BLACK BOX MULTIGRID FOR CONVECTION–DIFFUSION EQUATIONS ON ADVANCED COMPUTERS

by

VICTOR ALAN BANDY

M.S., University of Colorado at Denver, 1988
B.S., Oregon State University, 1983

A thesis submitted to the University of Colorado at Denver in partial fulfillment of the requirements for the degree of Doctor of Philosophy Department of Mathematics 1996
This thesis for the Doctor of Philosophy degree by

Victor Alan Bandy

has been approved for the

Department of

Mathematics

by

Jan Mandel

Joel E. Dendy, Jr.

Stephen McCormick

Leopoldo Franca

Gita Alaghband

Date ______________________
ABSTRACT

In this thesis we present Black Box Multigrid methods for the solution of convection-diffusion equations with anisotropic and discontinuous coefficients on advanced computers. The methods can be classified as either using standard or semi-coarsening for the generation of the coarse grids. The domains are assumed to be either two or three dimensional with a logically rectangular mesh structure being used for the discretization.

New grid transfer operators are presented and compared to earlier grid transfer operators. The new operators are found to be more robust for convection-diffusion equations.

Local mode and model problem analysis are used to examine several choices of iterative methods for the smoother and their relative effectiveness for the class of problems under consideration. The red/black alternating line Gauss-Seidel method and the incomplete line LU (ILLU) by lines-in-x methods were found to be the most robust for two dimensional domains, and red/black alternating plane Gauss-Seidel, using the the 2D black box multigrid method for the plane solves, was found to be the most robust and efficient smoother for 3D problems.

The Black Box Multigrid methods were developed to be portable, but optimized for either vector computers, such as the Cray Y-MP, or for parallel computers,
such as the CM-5. While the computer architectures are very different, they represent two of the main directions that supercomputer architectures are moving in today. Performance measures for a variety of test problems are presented for the two computers.

The vectorized methods are suitable for another large class of common computers that use superscalar pipelined processors, such as PCs and workstations. While the codes have not been optimized for these computers, especially when considering caching issues, they do perform quite well. Some timing results are presented for a Sun Sparc-5 for comparison with the supercomputers.

This abstract accurately represents the contents of the candidate’s thesis. I recommend its publication.

Signed

Joel E. Dendy, Jr.
To my Mom, Lee Buchanan, and everyone else who kept on asking

“When are you going to finish?”
CONTENTS

CHAPTER

1 INTRODUCTION .............................................. 1

1.1 Summary ..................................................... 1

1.1.1 Previous Results ........................................... 2

1.1.2 New Contributions ......................................... 6

1.2 Class of Problems ........................................... 9

1.3 Discretization of the Problem ............................... 10

1.4 Multigrid Overview ......................................... 13

1.4.1 Multigrid Cycling Strategies .............................. 19

1.5 Black Box Multigrid ....................................... 24

2 DISCRETIZATIONS: FINITE DIFFERENCE AND FINITE VOLUME ... 27

2.1 Finite Difference Discretization ............................... 28

2.2 Finite Volume Discretization ................................ 31

2.3 Cell Centered Finite Volume Discretization; Evaluation at the Vertices 34

2.3.1 Interior Finite Volumes ................................... 36

2.3.2 Dirichlet Boundary Condition ............................. 37

2.3.3 Neumann and Robin Boundary Conditions .................. 38

2.4 Cell Centered Finite Volume Discretization; Evaluation at the Cell Centers ................................................... 39

2.4.1 Interior Finite Volumes ................................... 40

2.4.2 Dirichlet Boundary Condition ............................. 41
2.4.3 Neumann and Robin Boundary Conditions ................. 42

2.5 Vertex Centered Finite Volume Discretization – Evaluation at the Vertices ................................................. 42
2.5.1 Interior Finite Volumes ........................................ 42
2.5.2 Edge Boundary Finite Volumes .............................. 43
2.5.3 Dirichlet Boundary Condition ............................... 43
2.5.4 Neumann and Robin Boundary Conditions ............... 43
2.5.5 Corner Boundary Finite Volumes ........................... 44
2.5.6 Dirichlet Boundary Condition ............................... 45
2.5.7 Neumann and Robin Boundary Conditions ............... 45

2.6 Vertex Centered Finite Volume Discretization – Evaluation at the Cell Vertices ................................................................. 46
2.6.1 Interior Finite Volumes ........................................ 46
2.6.2 Dirichlet Boundary Condition ............................... 47
2.6.3 Neumann and Robin Boundary Conditions ............... 47
2.6.4 Corner Boundary Finite Volumes ........................... 48
2.6.5 Dirichlet Boundary Condition ............................... 48
2.6.6 Neumann and Robin Boundary Conditions ............... 49

3 PROLONGATION AND RESTRICTION OPERATORS ............. 51
3.1 Prolongation .......................................................... 52
3.1.1 Prolongation Correction Near Boundaries ............... 55
3.2 Restriction ............................................................ 56
3.3 Overview .............................................................. 56
3.4 Symmetric Grid Operator $L^h$: Collapsing Methods ...... 59
3.5 Nonsymmetric Grid Operator $L^h$: Collapsing Methods ... 65
3.5.1 Prolongation Based on $\text{symm}(L^h)$ ................. 65
3.5.2 Prolongation Based on $L^h$ and $\text{symm}(L^h)$ 
3.5.3 Grid Transfer Operators Based on a hybrid form of $L^h$ and 
$\text{symm}(L^h)$ 
3.6 Nonsymmetric Grid Operators: Extension of Schaffer’s Idea 
3.6.1 Extension of Schaffer’s Idea to Standard Coarsening 
3.7 Conclusions Regarding Grid Transfer Operators 

4 BASIC ITERATION METHODS FOR SMOOTHERS 
4.1 Overview of Basic Iteration Methods 
4.2 Gauss-Seidel Relaxation 
4.2.1 Point Gauss-Seidel Iteration 
4.2.2 Line Gauss-Seidel Iteration by Lines in X 
4.2.3 Line Gauss-Seidel Iteration by Lines in Y 
4.2.4 Alternating Line Gauss-Seidel Iteration 
4.3 Incomplete Line LU Iteration 

5 FOURIER MODE ANALYSIS OF SMOOTHERS 
5.1 Introduction 
5.2 Motivation 
5.3 Overview of Smoothing Analysis 
5.4 2D Model Problems 
5.5 Local Mode Analysis for Point Gauss-Seidel Relaxation 
5.6 Local Mode Analysis for Line Gauss-Seidel Relaxation 
5.7 Local Mode Analysis for Alternating Line Gauss-Seidel and ILLU Iteration 
5.8 Local Mode Analysis Conclusions 
5.9 Other Iterative Methods Considered for Smoothers 

6 VECTOR ALGORITHM CONSIDERATIONS IN TWO DIMENSIONS
6.1 Cray Hardware Overview ...................................... 127
6.2 Memory Mapping and Data Structures ......................... 131
6.3 Scalar Temporaries ........................................... 132
6.4 In–Code Compiler Directives .................................. 133
6.5 Inlining ....................................................... 134
6.6 Loop Swapping ............................................... 135
6.7 Loop Unrolling ............................................... 135
6.8 Loops and Conditionals ....................................... 135
6.9 Scalar Operations ............................................ 136
6.10 Compiler Options ............................................ 136
6.11 Some Algorithmic Considerations for Smoothers .............. 137
   6.11.1 Point Gauss–Seidel Relaxation .......................... 137
   6.11.2 Line Gauss–Seidel Relaxation .......................... 138
6.12 Coarsest Grid Direct Solver ................................ 139
6.13 \(l_2\)-Norm of the Residual ................................ 140
6.14 2D Standard Coarsening Vector Algorithm .................... 144
   6.14.1 Coarsening ............................................ 144
   6.14.2 Data Structures ....................................... 144
   6.14.3 Smoothers ............................................. 145
   6.14.4 Coarsest Grid Solver .................................. 146
   6.14.5 Grid Transfer Operators .............................. 146
   6.14.6 Coarse Grid Operators ............................... 146
6.15 2D Semi–Coarsening Vector Algorithm ........................ 146
   6.15.1 Data Structures ....................................... 146
   6.15.2 Coarsening ............................................ 146
   6.15.3 Smoothers ............................................. 146
6.15.4 Coarsest Grid Solver ........................................... 147
6.15.5 Grid Transfer Operators ...................................... 147
6.15.6 Coarse Grid Operators ........................................ 147

7 2D NUMERICAL RESULTS ................................................. 148
7.1 Storage Requirements ............................................. 148
7.2 Vectorization Speedup ............................................. 151
7.3 2D Computational Work ......................................... 156
7.4 Timing Results for Test Problems ............................... 157
7.5 Numerical Results for Test Problem 8 ......................... 165
7.6 Numerical Results for Test Problem 9 ......................... 174
7.7 Numerical Results for Test Problem 10 ........................ 181
7.8 Numerical Results for Test Problem 11 ....................... 187
7.9 Numerical Results for Test Problem 13 ....................... 191
7.10 Numerical Results for Test Problem 17 ...................... 194
7.11 Comparison of 2D Black Box Multigrid Methods ............. 198

8 PARALLEL ALGORITHM CONSIDERATIONS IN TWO DIMENSIONS 203
8.1 CM-2 and CM-200 Parallel Algorithms ......................... 203
  8.1.1 Timing Comparisons ......................................... 206
8.2 CM-5 Hardware Overview ......................................... 207
8.3 CM-5 Memory Management ........................................ 215
8.4 Dynamic Memory Management Utilities ......................... 219
8.5 CM-5 Software Considerations .................................. 222
8.6 Coarsening and Data Structures in 2D ......................... 223
8.7 Coarse Grid Operators ........................................... 227
8.8 Grid Transfer Operators .......................................... 228
8.9 Smoothers ......................................................... 229
### 8.9.1 Parallel Line Gauss–Seidel Relaxation

- CM-5 Tridiagonal Line Solver Using Cyclic Reduction

### 8.10 Coarsest Grid Solver

### 8.11 Miscellaneous Software Issues

- Using Scalapack
- Poly-Shift Communication

### 8.12 2D Standard Coarsening Parallel Algorithm

- Data Structures
- Coarsening
- Smoothers
- Coarsest Grid Solver
- Grid Transfer Operators
- Coarse Grid Operators

### 8.13 2D Semi-Coarsening Parallel Algorithm

- Data Structures
- Coarsening
- Smoothers
- Coarsest Grid Solver
- Grid Transfer Operators
- Coarse Grid Operators

### 8.14 2D Parallel Timings

### 9 BLACK BOX MULTIGRID IN THREE DIMENSIONS

#### 9.1 Introduction

#### 9.1.1 Semi-Coarsening

### 10 3D DISCRETIZATIONS

#### 10.1 Finite Difference Discretization

---

229
230
233
236
236
237
237
238
238
239
239
239
240
240
240
241
241
241
241
241
250
250
251
253
254
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>Finite Volume Discretization</td>
<td>254</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Interior Finite Volumes</td>
<td>255</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Edge Boundary Finite Volumes</td>
<td>256</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Dirichlet Boundary Condition</td>
<td>257</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Neumann and Robin Boundary Conditions</td>
<td>257</td>
</tr>
<tr>
<td>11</td>
<td>3D NONSYMMETRIC: GRID TRANSFER OPERATORS</td>
<td>260</td>
</tr>
<tr>
<td>11.1</td>
<td>3D Grid Transfer Operations</td>
<td>262</td>
</tr>
<tr>
<td>11.2</td>
<td>3D Nonsymmetric Grid Operator $L^A$: Collapsing Methods</td>
<td>264</td>
</tr>
<tr>
<td>11.2.1</td>
<td>3D Grid Transfer Operator Variations</td>
<td>268</td>
</tr>
<tr>
<td>11.3</td>
<td>3D Coarse Grid Operator</td>
<td>268</td>
</tr>
<tr>
<td>12</td>
<td>3D SMOOTHERS</td>
<td>270</td>
</tr>
<tr>
<td>12.1</td>
<td>Point Gauss–Seidel</td>
<td>270</td>
</tr>
<tr>
<td>12.2</td>
<td>Line Gauss–Seidel</td>
<td>271</td>
</tr>
<tr>
<td>12.3</td>
<td>Plane Gauss–Seidel</td>
<td>272</td>
</tr>
<tr>
<td>13</td>
<td>LOCAL MODE ANALYSIS IN THREE DIMENSIONS</td>
<td>274</td>
</tr>
<tr>
<td>13.1</td>
<td>Overview of 3D Local Mode Analysis</td>
<td>274</td>
</tr>
<tr>
<td>13.2</td>
<td>Three Dimensional Model Problems</td>
<td>278</td>
</tr>
<tr>
<td>13.3</td>
<td>Local Mode Analysis for Point Gauss–Seidel Relaxation</td>
<td>280</td>
</tr>
<tr>
<td>13.4</td>
<td>Local Mode Analysis for Line Gauss–Seidel Relaxation</td>
<td>285</td>
</tr>
<tr>
<td>13.5</td>
<td>Local Mode Analysis for Plane Gauss–Seidel Relaxation</td>
<td>293</td>
</tr>
<tr>
<td>14</td>
<td>3D VECTOR ALGORITHM CONSIDERATIONS</td>
<td>308</td>
</tr>
<tr>
<td>14.1</td>
<td>3D Smoother</td>
<td>308</td>
</tr>
<tr>
<td>14.2</td>
<td>Data Structures and Memory</td>
<td>309</td>
</tr>
<tr>
<td>14.3</td>
<td>3D Standard Coarsening Vector Algorithm</td>
<td>313</td>
</tr>
<tr>
<td>14.3.1</td>
<td>Coarsening</td>
<td>313</td>
</tr>
<tr>
<td>14.3.2</td>
<td>Data Structures</td>
<td>313</td>
</tr>
</tbody>
</table>
FIGURES

FIGURE

1.1 Standard coarsening: superimposed fine grid $G^h$ and coarse grid $G^H$. . . 14
1.2 Semi-coarsening: superimposed fine grid $G^h$ and coarse grid $G^H$ . . . . 15
1.3 One V-cycle iteration for five grid levels. . . . . . . . . . . . . . . . . . . 20
1.4 One S-cycle iteration for four grid levels. . . . . . . . . . . . . . . . . . . 22
1.5 One W-cycle iteration for four grid levels. . . . . . . . . . . . . . . . . . . 22
1.6 One F-cycle iteration for five grid levels. . . . . . . . . . . . . . . . . . . 23
2.1 Vertex centered finite volume grid. . . . . . . . . . . . . . . . . . . . . . . 32
2.2 Cell centered finite volume grid. . . . . . . . . . . . . . . . . . . . . . . . . 33
2.3 Cell centered finite volume $\Omega_{i,j}$. . . . . . . . . . . . . . . . . . . . . 35
2.4 Vertex centered finite volume $\Omega_{i,j}$ at $y = 0$. . . . . . . . . . . . . . 43
2.5 Southwest boundary corner finite volume. . . . . . . . . . . . . . . . . . 44
3.1 Standard coarsening interpolation 2D cases . . . . . . . . . . . . . . . . . 53
6.1 Cray Y-MP hardware diagram . . . . . . . . . . . . . . . . . . . . . . . . . 128
6.2 Cray CPU configuration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
6.3 2D Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 145
7.1 Comparison of Setup time for BMGNS, SCBMG, and MGD9V . . . . . 154
7.2 Comparison of one V-cycle time for BMGNS, SCBMG, and MGD9V . . . 155
7.3 Domain $\Omega$ for problem 8. . . . . . . . . . . . . . . . . . . . . . . . . . . . 166
7.4 Domain $\Omega$ for problem 9. . . . . . . . . . . . . . . . . . . . . . . . . . . . 174
7.5 Domain $\Omega$ for problem 10. . . . . . . . . . . . . . . . . . . . . . . . . . . 181
7.6 Domain $\Omega$ for problem 11. ........................................ 187
7.7 Domain $\Omega$ for problem 13. ........................................ 191
7.8 Domain $\Omega$ for problem 17. ........................................ 195
8.1 CM-5 system diagram .................................................. 210
8.2 CM-5 processor node diagram ........................................ 212
8.3 CM-5 vector unit diagram ............................................. 214
8.4 CM-5 processor node memory map .................................... 217
8.5 Grid Data Structure Layout. .......................................... 225
9.1 Grid operator stencil in three dimensions. ....................... 252
11.1 Grid transfer operator’s stencil in three dimensions. ........ 261
14.1 3D FSS data structure ............................................... 311
## TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Smoothing factor $\mu$ for point Gauss–Seidel relaxation for anisotropic diffusion equations</td>
<td>109</td>
</tr>
<tr>
<td>5.2</td>
<td>Smoothing factor $\mu$ for point Gauss–Seidel relaxation for convection–diffusion equations</td>
<td>110</td>
</tr>
<tr>
<td>5.3</td>
<td>Smoothing factor $\mu$ for x– and y–line Gauss–Seidel relaxation for anisotropic diffusion equations</td>
<td>114</td>
</tr>
<tr>
<td>5.4</td>
<td>Smoothing factor $\mu$ for x– and y–line Gauss–Seidel relaxation for convection–diffusion equations</td>
<td>116</td>
</tr>
<tr>
<td>5.5</td>
<td>Smoothing factor $\mu$ for alternating line Gauss–Seidel relaxation and incomplete line LU iteration for anisotropic diffusion equations</td>
<td>119</td>
</tr>
<tr>
<td>5.6</td>
<td>Smoothing factor $\mu$ for alternating line Gauss–Seidel relaxation and incomplete line LU iteration for convection–diffusion equations</td>
<td>121</td>
</tr>
<tr>
<td>6.1</td>
<td>Cray Y-MP Timings for the Naive, Kahan, and Doubling Summation Algorithms</td>
<td>143</td>
</tr>
<tr>
<td>6.2</td>
<td>Sparc5 Timings for the Naive, Kahan, and Doubling Summation Algorithms</td>
<td>144</td>
</tr>
<tr>
<td>7.1</td>
<td>Memory storage requirements for the Cray Y-MP.</td>
<td>149</td>
</tr>
<tr>
<td>7.2</td>
<td>Storage requirements for BMGNS, SCBMG, and MGD9V.</td>
<td>150</td>
</tr>
<tr>
<td>7.3</td>
<td>Vectorization speedup factors for standard coarsening.</td>
<td>151</td>
</tr>
<tr>
<td>7.4</td>
<td>Vectorization speedup factors for semi-coarsening.</td>
<td>152</td>
</tr>
<tr>
<td>7.5</td>
<td>Operation count for standard coarsening setup.</td>
<td>156</td>
</tr>
</tbody>
</table>
7.23 Number of V–cycles for standard coarsening using the operator, L/L, grid transfer for problem 8 with ILLU. ......................................................... 172
7.24 Number of V–cycles for standard coarsening using the extension of Schaffer’s idea for problem 9. ......................................................... 175
7.25 Number of V–cycles for standard coarsening using the sL/L grid transfer for problem 9. ......................................................... 175
7.26 Number of V–cycles for standard coarsening using the hybrid sL/L grid transfer for problem 9. ......................................................... 176
7.27 Number of V–cycles for standard coarsening using the symmetric grid transfer for problem 9. ......................................................... 176
7.28 Number of V–cycles for standard coarsening using the operator, L/L, grid transfer for problem 9. ......................................................... 177
7.29 Number of V–cycles for standard coarsening using the extension of Schaffer’s idea for problem 9 with ILLU. ......................................................... 177
7.30 Number of V–cycles for standard coarsening using the sL/L grid transfer for problem 9 with ILLU. ......................................................... 178
7.31 Number of V–cycles for standard coarsening using the hybrid sL/L grid transfer for problem 9 with ILLU. ......................................................... 178
7.32 Number of V–cycles for standard coarsening using the symmetric grid transfer for problem 9 with ILLU. ......................................................... 179
7.33 Number of V–cycles for standard coarsening using the operator, L/L, grid transfer for problem 9 with ILLU. ......................................................... 179
7.34 Number of V–cycles for standard coarsening using the symmetric grid transfer for problem 9 with 4-direction PGS. ......................................................... 180
7.35 Number of V–cycles for standard coarsening using the extension of Schaffer’s idea for problem 10. ......................................................... 183
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.36</td>
<td>Number of (V)-cycles for standard coarsening using the (sL/L) grid transfer for problem 10.</td>
<td>183</td>
</tr>
<tr>
<td>7.37</td>
<td>Number of (V)-cycles for standard coarsening using the hybrid (sL/L) grid transfer for problem 10.</td>
<td>184</td>
</tr>
<tr>
<td>7.38</td>
<td>Number of (V)-cycles for standard coarsening using the symmetric grid transfer for problem 10.</td>
<td>184</td>
</tr>
<tr>
<td>7.39</td>
<td>Number of (V)-cycles for standard coarsening using the operator, (L/L), grid transfer for problem 10.</td>
<td>185</td>
</tr>
<tr>
<td>7.40</td>
<td>Number of (V)-cycles for standard coarsening using the extension of Schaffer’s idea for problem 10.</td>
<td>185</td>
</tr>
<tr>
<td>7.41</td>
<td>Number of (V)-cycles for standard coarsening using the (sL/L) grid transfer for problem 10.</td>
<td>185</td>
</tr>
<tr>
<td>7.42</td>
<td>Number of (V)-cycles for standard coarsening using the symmetric grid transfer for problem 10.</td>
<td>186</td>
</tr>
<tr>
<td>7.43</td>
<td>Number of (V)-cycles for MGD9V on problem 10.</td>
<td>186</td>
</tr>
<tr>
<td>7.44</td>
<td>Number of (V)-cycles for standard coarsening using the extension of Schaffer’s idea for problem 11.</td>
<td>188</td>
</tr>
<tr>
<td>7.45</td>
<td>Number of (V)-cycles for standard coarsening using the (sL/L) grid transfer for problem 11.</td>
<td>189</td>
</tr>
<tr>
<td>7.46</td>
<td>Number of (V)-cycles for standard coarsening using the hybrid (sL/L) grid transfer for problem 11.</td>
<td>189</td>
</tr>
<tr>
<td>7.47</td>
<td>Number of (V)-cycles for standard coarsening using the symmetric grid transfer for problem 11.</td>
<td>190</td>
</tr>
<tr>
<td>7.48</td>
<td>Number of (V)-cycles for standard coarsening using the operator, (L/L), grid transfer for problem 11.</td>
<td>190</td>
</tr>
</tbody>
</table>
7.49 Number of V–cycles for standard coarsening using the extension of Schaffer’s idea for problem 13. ........................................ 192
7.50 Number of V–cycles for standard coarsening using the hybrid sL/L grid transfer for problem 13. ........................................ 193
7.51 Number of V–cycles for standard coarsening using the symmetric grid transfer for problem 13. ........................................ 193
7.52 Number of V–cycles for standard coarsening using the extension of Schaffer’s idea for problem 17. ........................................ 194
7.53 Number of V–cycles for standard coarsening using the original collapsing method for problem 17. ........................................ 196
7.54 Number of V–cycles for standard coarsening using the extension of Schaffer’s idea for problem 17. ........................................ 197
7.55 Number of V–cycles for standard coarsening using the hybrid collapsing method for problem 17. ........................................ 197
7.56 Number of V–cycles for semi-coarsening for problem 17. .......... 197
7.57 Comparison for problem 8 on Cray Y-MP ............................. 199
7.58 Comparison for problem 9 on Cray Y-MP ............................. 201
8.1 Timing Comparison per V-cycle for semi-coarsening on the Cray Y-MP, CM-2, and CM-200. ........................................ 206
8.2 Timing Comparison per V-cycle for Standard Coarsening on the Cray Y-MP, CM-2, and CM-200. ........................................ 208
8.3 2D Standard coarsening 32 – 512 CM-5 nodes V-cycle timings .... 243
8.4 2D Standard coarsening 32 – 512 CM-5 nodes Setup timings ....... 244
8.5 2D Standard coarsening 32 – 512 CM-5 nodes parallel efficiency .... 244
8.6 2D Semi-coarsening 32 – 512 CM-5 nodes V-cycle timings ......... 245
8.7 2D Semi-coarsening 32 – 512 CM-5 nodes setup timings .......... 246
8.8 2D Semi-coarsening 32 – 512 CM-5 nodes parallel efficiency .......................... 246
8.9 2D Timing comparison between CM-5, Cray Y-MP, and Sparc-5 ............. 248
13.1 Smoothing factor for point Gauss–Seidel relaxation for anisotropic diffusion equations in 3D .................................................. 282
13.2 Smoothing factor for point Gauss–Seidel relaxation for convection–diffusion equations in 3D .................................................. 282
13.3 Smoothing factor for point Gauss–Seidel relaxation for convection–diffusion equations in 3D .................................................. 283
13.4 Smoothing factor for point Gauss–Seidel relaxation for convection–diffusion equations in 3D .................................................. 284
13.5 Smoothing factors for line Gauss–Seidel relaxation for anisotropic diffusion equations .................................................. 289
13.6 Smoothing factors for line Gauss–Seidel relaxation for convection–diffusion equations .................................................. 290
13.7 Smoothing factors for line Gauss–Seidel relaxation for convection–diffusion equations .................................................. 291
13.8 Smoothing factors for line Gauss–Seidel relaxation for convection–diffusion equations .................................................. 292
13.9 Smoothing factors for zebra line Gauss–Seidel relaxation for anisotropic diffusion equations .................................................. 293
13.10 Smoothing factors for zebra line Gauss–Seidel relaxation for convection–diffusion equations .................................................. 294
13.11 Smoothing factors for zebra line Gauss–Seidel relaxation for convection–diffusion equations .................................................. 295
13.12 Smoothing factors for zebra line Gauss–Seidel relaxation for convection–diffusion equations .................................................. 296
13.13 Smoothing factor $\mu$ for xy-, zz-, and yz-plane Gauss–Seidel relaxation for anisotropic diffusion equations .................................................. 300
13.14 Smoothing factor $\mu$ for xy-, zz-, and yz-plane Gauss–Seidel relaxation for convection–diffusion equations .................................................. 301
13.15 Smoothing factor $\mu$ for plane Gauss–Seidel (continued) ................. 302
13.16 Smoothing factor $\mu$ for plane Gauss–Seidel (continued) ................. 303
13.19 Smoothing factor for zebra plane Gauss–Seidel (continued) ................ 306
13.20 Smoothing factor for zebra plane Gauss–Seidel (continued) ................ 307
14.1 3D Multigrid Component Timing ...................................................... 317
14.2 Grid transfer timing comparison for standard and semi-coarsening ...... 318
14.3 Timing for various smoothers ............................................................ 319
14.4 Smoothing versus grid transfer timing ratios ........................................ 320
14.5 Numerical results for problem 1 in 3D. ............................................. 321
14.6 Numerical results for problem 1 in 3D. ............................................. 321
14.7 Numerical results for problem 1 in 3D. ............................................. 322
14.8 Numerical results for problem 1 in 3D. ............................................. 323
14.9 Numerical results for problem 1 in 3D. ............................................. 323
14.10 Numerical results for problem 1 in 3D. ............................................ 323
15.1 3D Standard coarsening 32, 64, 128 CM-5 nodes V-cycle timings ......... 327
15.2 3D Standard coarsening 32, 64, 128 CM-5 nodes Setup timings .......... 327
15.3 3D Standard coarsening 32, 64, 128 CM-5 nodes parallel efficiency ...... 328
15.4 3D Semi-coarsening 32, 64, 128 CM-5 nodes V-cycle timings ............. 329
15.5 3D Semi-coarsening 32, 64, 128 CM-5 nodes setup timings . . . . . . . . . . 329
15.6 3D Semi-coarsening 32, 64, 128 CM-5 nodes parallel efficiency . . . . . . . 329
15.7 3D Timing comparison between CM-5 and Cray Y-MP . . . . . . . . . . 330
ACKNOWLEDGMENTS
I would first like to thank my advisor Joel E. Dendy, Jr., Los Alamos National Laboratory, because without him none of this would have been possible; Thanks! In addition, at Los Alamos National Laboratory, I would like to thank Mac Hyman of Group T-7, the Center for Nonlinear Studies for their support and the Advanced Computing Laboratory and the CIC Division for the use of their computing facilities. This work was partially supported by the Center for Research on Parallel Computation through NSF Cooperative Agreement No. CCR-8809615.

I would also like to thank my PhD committee members at UCD, with a special thanks to professors Bill Briggs, Stan Payne, Roland Sweet and Jan Mandel. In addition, I would like to give a big thanks to Dr. Suely B. Oliveira for getting me back on track.

Finally, I would like to thank my mom, Lee Buchanan, my twin brother, Fred Bandy, my wife, Darlene Bandy, and all my friends for all their support and encouragement. Last but not least, a special thanks to Mark and Flavia Kuta, some very good friends, for letting me stay with them while I was in Denver.

On a sadder note, I would like to express a great debt that is owed to Seymour Cray, who died Oct. 5 1996. This thesis would never have been possible without Seymour Cray’s creativity, intelligence, and drive to conceive and develop supercomputers. His contributions to computing have changed the face of science and engineering and we will all miss him dearly.
CHAPTER 1

INTRODUCTION

1.1 Summary

The subject of this dissertation is the investigation of Black Box multigrid solvers for the numerical solution of second order elliptic partial differential equations in two or three dimensional domains. We place particular emphasis on efficiency on both vector and parallel computers, represented here by the Cray Y–MP and the Thinking Machines CM–5.

Black Box multigrid methods are sometimes referred to as geometric multigrid methods or, more recently, as automatic multigrid methods, in the literature. The methods can be considered to be a subclass of algebraic multigrid methods with several algorithmic restrictions. Geometric multigrid methods make \textit{a priori} assumptions about the domain and the class of problems that are to be solved, and in addition, it uses intergrid operators and coarse grid points based on the geometry and the order of grid equation operator. Algebraic multigrid, on the other hand, chooses both the coarse grid and intergrid operator based only on the coefficient matrix. Black box multigrid is in between these two, with the grids chosen geometrically, on logically rectangular grids, and the intergrid operators are chosen algebraically. There are other hybrid multigrid methods such as the unstructured grid method by Chan [22], which chooses the coarse grid based on graph theoretical considerations and the intergrid operator from the nodal coordinates (geometry), and the algebraic multigrid method of Vaněk
which uses kernels of the associated quadratics form in lieu of geometrical information. The algebraic multigrid method of Stüben and Ruge [66] [67] use almost the same construction of intergrid operator as Dendy [26] once the coarse has been chosen, while Van Řek’s work is based on a different idea. The assumptions and the components that make up the black box multigrid methods are spelled out in more detail in the following sections of this chapter.

We will examine the development of robust black box multigrid solvers using both standard and semi-coarsening. The methods are aimed at the solution of convection-diffusion equations with anisotropic and discontinuous coefficients (interface problems), such that the discrete system of equations need only be specified on a logically rectangular grid. A guiding principal in the design is that if the discrete system of equations is symmetric, then the multigrid coarse grid problems should preserve that symmetry.

1.1.1 Previous Results. The black box multigrid method was first introduced by Dendy [26]. The method is a practical implementation of a multigrid method for symmetric diffusion problems with anisotropic and discontinuous coefficients, represented by

$$-
abla \cdot (D \cdot \nabla U) + cU = f \quad \text{on } \Omega \subset \mathbb{R}^2. \quad (1.1)$$

The domain $\Omega$ is assumed to be embedded in a logically rectangular mesh and then discretized in such a manner as to yield a stencil which is no larger than a compact 9-point stencil. The method employs the Galerkin coarse grid approximation, $L^H = I_h^H L_k^H I_H^k$, to form the coarse grid operators, using the robust choice of grid transfer operators from Alcouffe et. al [1]. The robust choice of grid transfer operators is an operator induced formulation that, when $c = 0$, preserves the flux $\mu \cdot (D \cdot \nabla U)$ across interfaces. In [1] lexicographic point Gauss-Seidel relaxation and alternating
lexicographic line Gauss-Seidel relaxation were the choices available for smoothers. In subsequent extensions for vector machines, the choices available were red/black (or four color for nine point operators) point Gauss-Seidel and alternating red/black line Gauss-Seidel relaxation.

The black box multigrid method was extended to nonsymmetric elliptic convection–diffusion problems [27], for which the model problem is

$$-\varepsilon \Delta U + U_x + U_y = f \quad \text{on } \Omega \subset \mathbb{R}^2,$$

where $\varepsilon > 0$. The mesh is the same as before and the discretization is of the form

$$L^h_{\beta h} U_{i,j} = -\beta h \Delta_h U_{i,j} + D^x_{0,h} U_{i,j} + D^y_{0,h} U_{i,j} = F_{i,j},$$

where

$$\Delta_h U_{i,j} = \frac{1}{h^2} (U_{i,j-1} + U_{i-1,j} - 4U_{i,j} + U_{i+1,j} + U_{i,j+1}),$$

$$D^x_{0,h} U_{i,j} = \frac{1}{2h} (U_{i+1,j} - U_{i-1,j}),$$

$$D^y_{0,h} U_{i,j} = \frac{1}{2h} (U_{i,j+1} - U_{i,j-1}),$$

and where $\beta = \frac{1}{2}$ yields upstream differencing. A generalization of Galerkin coarse grid approximation is used to form the coarse grid operators. The prolongation operators are formed in the same way as they were for the symmetric method, but instead of being induced by $L^h$, they are induced by the symmetric part of the grid operator, $\text{symm}(L^h) = \frac{1}{2}((L^h)^* + L^h)$. It was found that instead of using $I^H_h = (I^H_H)^*$ to induce the restriction operator, a more robust choice is to form a new interpolation operator $J^h_H$ based on $(L^h)^*$ and then to define the restriction operator to be $I^H_h = (J^h_H)^*$. These choices were made to generalize the work of [26]. The choice of smoothers was also changed to include lexicographic point, line, and alternating line Kaczmarz relaxation.
The method performed well for the problems tested as long as \( \beta \geq 0.25 \), but since nonphysical oscillations begin to dominate for \( \beta < 0.25 \), this restriction is no difficulty.

The next development was the creation of a 3D black box multigrid solver for symmetric problems [29]. This method uses the same type of grid transfer operators as the earlier 2D symmetric method. Two different methods of forming the coarse grid operators were examined with nearly identical convergence results. The first method uses Galerkin coarse grid approximation with standard coarsening. The second method also uses Galerkin coarse grid approximation, but it does so by using auxiliary intermediate grids obtained by semi-coarsening successively in each of the three independent variables. For robustness, alternating red/black plane Gauss-Seidel relaxation was used for the smoother. The plane solves of the smoother were performed by using the 2D symmetric black box multigrid solver.

The 2D symmetric black box multigrid solver was then extended to solve singular and periodic diffusion problems [30]. The existence of a solution, in case \( c = 0 \), is assured by requiring that the equation be consistent; \( \int_{\Omega} F = 0 \). The periodic boundary conditions only impact the multigrid method by requiring the identification of the auxiliary grid point equations at setup, the identification of the auxiliary grid point unknowns after interpolation, and the identification of the auxiliary grid point residuals before restriction. The coarsest grid problem, if \( c = 0 \), is singular and cannot be solved by Gaussian elimination, but since the solution is determined only up to a constant, the arbitrary addition of the linearly independent condition that \( U_{i,j} = 0 \) for some coarse grid point \((i, j)\) allows solution by Gaussian elimination.

The first semi-coarsening black box multigrid solver was introduced for the solution of three dimensional petroleum reservoir simulations [33]. This method employs semi-coarsening in the z-direction and xy-plane relaxation for the smoother. Galerkin coarse grid approximation is used to form the coarse grid operators. Operator induced
grid transfer operators were used, but only after Schaffer’s paper [70] was it realized how to compute these in a robust manner; see section 3.6.

A two dimensional black box multigrid solver called MGD9V was developed by de Zeeuw [24]. This method was designed to solve the general elliptic convection–diffusion equation. The method used standard coarsening, an ILLU smoother, a $V(0,1)$-cycle (sawtooth), and a new set of operator induced grid transfer operators that were designed specifically for convection dominated problems. The method was found to be more robust than previous methods but was still divergent for problems with closed convection characteristics on large grids. The method of MGD9V was developed only for two dimensions and is not parallelizable.

The 2D symmetric black box multigrid solvers [26] [30] were updated to be portable, have consistent user interfaces, adhering to the SLATEC software guidelines [38], and provided with three new user interfaces by Bandy [9]. One of the interfaces included an automatic discretization routine, requiring the user to provide only a function which can evaluate the coefficients at the fine grid points. The interfaces all included extensive input parameter validation and memory management for workspace.

A parallel version of the semi-coarsening method for two dimensional scalar problems for the CM-2 was presented in [32]. A parallel version of semi-coarsening for two- and three- dimensional problems was presented in [75]. Both papers essentially relied on the algorithm from [33] and borrowed from Schaffer [69] [70] for the robust determination of grid transfer operators.

Fourier mode analysis has been used by many multigrid practitioners to find good smoothers for use in multigrid methods. The results of many of these analyses have been presented in the literature. Stüben and Trottenberg [78] present several fundamental results of Fourier mode analysis for a few selected 2D problems. Kettler [50] reports results for a range of 2D test problems and several lexicographic ordered
Gauss-Seidel methods along with several variations of ILU methods. Wesseling [84] reports a summary of smoothing analysis results for the 2D rotated anisotropic diffusion equation and the convection diffusion equation; however, the results are for only a limited number of worst case problems. Smoothing analysis results for the red/black ordered methods appear in many places in the literature, but they are only for a few selected problems. There are some results in the literature for 3D problems [79], but just like the 2D results, the analysis is not complete enough for our purposes.

1.1.2 New Contributions In this thesis we have developed and extended several black box multigrid methods for both two and three dimensional non-symmetric problems on sequential, vector, and parallel computing platforms. The new methods are based on a new implementation of the two dimensional nonsymmetric black box multigrid method [27] for vector computers. The new implementation was designed to take better advantage of developments in vector computing, while increasing portability and compatibility with sequential computers. The new implementation performs with a speedup factor of six over the earlier methods on vector computers, while providing identical functionality, and it also incorporates many of the ideas and software features from [9].

The new methods include the development of a three dimensional method, both vector and parallel versions, and a two dimensional parallel method for nonsymmetric problems. The new methods were also extended to handle periodic and singular problems using the modifications from [30].

In [27] a two dimensional nonsymmetric black box multigrid method was examined for a convection dominated problem with constant convection characteristics. In this work we investigate the new methods for a general convection-diffusion equation

\[-\nabla \cdot (D(\chi) \cdot \nabla U(\chi)) + b(\chi) \cdot \nabla U(\chi) + c(\chi) U(\chi) = f(\chi), \ \chi \in \Omega. \quad (1.4)\]
When the earlier method of [27] was applied to equation 1.4, but with more vectorizable
smoothers than those in [27], it was found to perform poorly, and even fail, for some
non-constant convection characteristic problems. This poor performance was caused
by both the new smoothers and by poor coarse grid correction. Several new grid
transfer operators are introduced to address these problems, of which two were found
to be robust; see chapter 3. The search for a more robust smoother was facilitated
by using local mode analysis, and led to the implementation of an incomplete line
LU factorization method (ILLU) for the smoother. The ILLU smoother made the
new methods more robust for convection dominated problems. A four-direction point
Gauss-Seidel method was also briefly considered for use as a smoother but was discarded
because it was not parallelizable nor suitable for anisotropic problems, even though it
was fairly robust for convection dominated problems.

A nonsymmetric black box multigrid method, using standard coarsening, was
created for three dimensional problems; previously only a semi-coarsening version ex-
isted. The new method is the three dimensional analogue of the new two dimensional
black box multigrid method, and it uses alternating red/black plane Gauss-Seidel as
a smoother for robustness. The 3D smoother uses one \( V(1,1) \)-cycle of the 2D non-
symmetric black box multigrid method to perform the required plane solves. The new
method was developed to use either the new grid transfer operators from the new 2D
nonsymmetric method or those from the 3D extension of Dendy's 2D nonsymmetric
black box multigrid method. The coarse grid operators are formed using the second
method from [29], which uses auxiliary intermediate grids obtained by successively
applying semi-coarsening in each of the independent variables. In addition, the new
method is designed to handle periodic and singular problems. Another use of local
mode analysis was in the design of robust three dimensional smoothers. Although
there are hints in the literature for how to perform local mode analysis for color relaxation in three dimensions, we are unaware of the appearance elsewhere of the detailed analysis presented in chapter 13.

The new methods are compared to a new implementation of the semi-coarsening method with a speedup factor of over 5 for the two dimensional method and speedup factor of 2 for the three dimensional method on vector computers. The grid transfer operators are based on Schaffer’s idea; see chapter 3. The 2D semi-coarsening method uses coarsening in the y-direction coupled with red/black x-line Gauss-Seidel relaxation for the smoother. The 3D semi-coarsening method uses coarsening in the z-direction coupled with red/black xy-plane Gauss-Seidel relaxation for the smoother. The new implementation also includes the ILLU smoother, not present in the original version.

Another aspect of this work was to compare de Zeeuw’s MGD9V with the black box multigrid methods. The idea was to mix and match components of the two approaches to investigate the strengths and weaknesses and to ascertain if a combination existed which was better than either. The results obtained from studying the algorithm components is that MGD9V obtains its robustness from the ILLU smoother and not from its grid transfer operators. If MGD9V uses alternating red/black line Gauss-Seidel for its smoother then performance similar to the black box multigrid methods’ is observed. Likewise, if ILLU is used as the smoother in the black box multigrid methods, then the performance is similar to that of MGD9V.

Parallel versions of the standard coarsening nonsymmetric black box multigrid methods are developed in this thesis and compared with the existing parallel version of semi-coarsening black box method. The 3D parallel version smoother uses a modified 2D nonsymmetric black box multigrid method to perform the simultaneous solution of all the planes of a single color.
A hybrid parallel black box multigrid method was developed that uses standard coarsening for grid levels with a VP (virtual processor) ratio, i.e. number of grid points per processor, greater than one, and semi-coarsening when the VP ratio is less than one. When the VP ratio is greater than one, standard coarsening reduces the number of grid points per processor, reducing the amount of serial work, faster than in semi-coarsening case. When the VP ratio is less than one, the semi-coarsening method is more efficient than standard coarsening because it keeps more processors busy that would otherwise be idle; in addition, tri-diagonal library routines, which are more efficient than we can write, are available for the data structures. The hybrid parallel method is the most efficient method on the CM-5 because it uses the most computationally efficient method for a given VP ratio.

1.2 Class of Problems

The class of problems that is being addressed is convection–diffusion equations with anisotropic and discontinuous coefficients on a two–or three–dimensional domain. These types of problems can be represented by the following equation and boundary conditions,

\[
L(\chi) \equiv -\nabla \cdot (D(\chi) \cdot \nabla U(\chi)) + b(\chi) \cdot \nabla U(\chi) + c(\chi)U(\chi) = f(\chi) \quad \chi \in \Omega \quad (1.5)
\]

\[
\nu(\chi) \cdot D(\chi)\nabla U(\chi) + \gamma(\chi)U(\chi) = 0 \quad \chi \in \partial \Omega, \quad (1.6)
\]

on a bounded domain \(\Omega \subset \mathbb{R}^d\) with boundary \(\partial \Omega\), where \(d\) is either 2 or 3, \(\chi = (x, y)\) or \((x, y, z)\), and \(D(\chi) = (D^1, D^2)\) or \((D^1, D^2, D^3)\), respectively. The term \(\nu(\chi)\) is the outward normal vector. It is assumed that \(D(\chi) > 0, c(\chi) \geq 0,\) and \(\gamma(\chi) \geq 0\) to ensure that upon discretization we end up with a positive definite system of equations. Anisotropies are also allowed, e.g. if \(\Omega \subset \mathbb{R}^2\) we have \(D = (D^1, D^2)\) where it is possible that \(D^1 \gg D^2\) in some subregion(s) while \(D^1 \ll D^2\) in other subregion(s). In addition,
\(D(\chi), c(\chi), \) and \(f(\chi)\) are allowed to be discontinuous across internal boundaries \(\Gamma \subset \Omega\). Moreover, let \(\mu(\chi)\) be a normal vector at \(\chi \in \Gamma\); then it is natural to assume also that

\[ U \text{ and } \mu \cdot (D\nabla U) \text{ are continuous at } \chi \text{ for almost every } \chi \in \Gamma. \] (1.7)

The “almost every” is necessary to exclude juncture points of \(\Gamma\), that is points where two pieces of \(\Gamma\) intersect and the continuity of \(\mu \cdot (D\nabla U)\) does not make any sense.

The boundary conditions permitted in (1.6) can be of three types: Dirichlet, Neumann, mixed. The periodic boundary condition is not considered, but can be handled by making a few adjustments and modifications to the black box multigrid codes. It should be noted that, for a problem with pure Neumann boundary conditions, a finite difference (volume or element) discretization may lead to a singular system of equations; the singularity can be propagated to the coarsest grid level and cause trouble for the direct solver, but a minor modification to the code circumvents this difficulty, allowing solution of the coarsest grid level problem.

1.3 Discretization of the Problem

Let the continuous problem represented by equation (1.5) be written in operator notation as

\[ Lu = f \quad \text{in } \Omega. \] (1.8)

The following discussion is valid for both two and three dimensions, but only the two dimensional case is presented. Suppose that, for all \(\chi = (x, y) \in \Omega\), \(a_x \leq x \leq b_x\) and \(a_y \leq y \leq b_y\). Let \(C^h\) define a rectangular grid on \([a_x, b_x] \times [a_y, b_y]\), partitioned with

\[ a_x = x_1 < x_2 < \cdots < x_{n_x} = b_x, \quad a_y = y_1 < y_2 < \cdots < y_{n_y} = b_y, \] (1.9)

and let the grid spacings be defined as

\[ h_{x_i} = x_{i+1} - x_i, \quad h_{y_j} = y_{j+1} - y_j \] (1.10)
Then the rectangular grid, $G^h$ is defined as

$$G^h = \{ x_i \in [a_x, b_x], y_j \in [a_y, b_y] \} \quad (1.11)$$

with the domain grid, $\Omega^h$, being defined as

$$\Omega^h = \Omega \cap G^h. \quad (1.12)$$

Before the discrete grid problem is defined we should first address the issue of domains with irregular boundaries. The black box multigrid solvers in two dimensions are intended to solve the equation (1.8) on logically rectangular grids, but for simplicity, we consider only rectangular grids. An irregular shaped domain can be embedded in the smallest rectangular grid, $G^h$, possible, $\Omega^h \subset G^h$. The problem is then discretized on $\Omega^h$ avoiding any coupling to the grid points not in $\Omega^h$. For grid points outside of $\Omega^h$, $\chi^h \in G^h - \Omega^h$, considered to be fictitious points, an arbitrary equation is introduced, such as $c_{i,j} u_{i,j} = f_{i,j}$, where $c_{i,j} \neq 0$ and $f_{i,j}$ are arbitrary. The problem is now rectangular and the solution to the discrete equations can be obtained at the points in the domain, while the solution $u_{i,j} = f_{i,j}/c_{i,j}$ is obtained for the other points. Problems with irregular domains in three dimensions can be handled in a similar fashion for a cuboid box grid.

Now the discrete grid problem approximating the continuous problem, (1.8) can be written as

$$L^h u^h = f^h \quad \text{in } G^h, \quad (1.13)$$

where the superscript $h$ refers to discretization with grid spacing $h$. Note that, for irregular domains the discrete solution $u^h(x)$ makes sense only for $\chi \in \Omega^h$; $u^h(\chi)$, for $\chi \in G^h \setminus \Omega^h$, is arbitrary.

We consider only discrete operators $L^h$ on rectangular grids that can be described by 5-point or 9-point box stencils. Suppose we discretize the equation (1.5)
using five points at the grid point \((x_i, y_j)\),

\[-S_{i,j}u_{i,j-1} - W_{i,j}u_{i-1,j} + C_{i,j}u_{i,j} - E_{i,j}u_{i+1,j} - N_{i,j}u_{i,j+1} = F_{i,j}.\]  

(1.14)

We use stencil notation to represent the 5 and 9 point cases, respectively:

\[
\begin{bmatrix}
N \\
W & C & E
\end{bmatrix}_{i,j}^h \quad \begin{bmatrix}
NW & N & NE \\
W & C & E \\
SW & S & SE
\end{bmatrix}_{i,j}^h
\]

(1.15)

where the stencil represents the coefficients for the discrete equation at the grid point 
\((x_i, y_j)\) on grid \(G^h\). The subscripts \(i, j\) can be dropped and it will be understood that
the stencil is centered at the grid point \((x_i, y_j)\). The superscript \(h\) can also be dropped
when the mesh spacing is clear from the context. The stencils are valid over the entire
grid including the boundary points because the coefficients are allowed to be zero.
Hence, any coefficients that reach out of the domain can be set to zero. Clearly, the
5-point stencil is a special case of the 9-point stencil, where the \(NW, NE, SW,\) and
\(SE\) coefficients are set to zero.

We illustrate the stencil notation for Poisson’s equation on a square domain
in two dimensions,

\[Lu(x, y) \equiv -u_{xx}(x, y) - u_{yy}(x, y) = f(x, y), \quad (x, y) \in \Omega = (0, 1)^2\]  

(1.16)

using 5- and 9-point finite difference discretizations. The 5-point stencil for the operator \(L\), using a central finite difference discretization on a uniform grid with grid spacing
\(h = 1/N\) for \(N = n_x = n_y\) is

\[
L^h = -\Delta^h = \frac{1}{h^2} \begin{bmatrix}
-1 \\
-1 & 4 & -1 \\
-1
\end{bmatrix}^h
\]

(1.17)
One 9-point discretization for $L$ in (1.16) has the stencil

$$L^h = \frac{1}{h^2} \begin{bmatrix} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{bmatrix}. \quad (1.18)$$

Many types of discretization can be considered: central finite differences, upstream finite differences, finite volumes, finite elements, etc.

The black box multigrid solvers actually allow for more general meshes than just the rectangular grids shown so far. The only requirement is that the mesh be logically rectangular. In two dimensions the logically rectangular grid $G$ can be defined as

$$G = \{x(i,j), y(i,j) : 1 \leq i \leq n_x, 1 \leq j \leq n_y \} \quad (1.19)$$

where the grid cell formed by

$$(x(i,j), y(i,j)), \quad (x(i+1,j+1), y(i+1,j+1))$$

$$(x(i,j), y(i+1,j)), \quad (x(i+1,j), y(i,j))$$

has positive area, $1 \leq i \leq n_x, 1 \leq j \leq n_y$.

The black box multigrid solvers which we consider require the discretization to be represented by a 9-point box stencil. However, just because the problem has a 9-point box stencil does not mean that it can be solved by the black box multigrid methods presented in this thesis. Such solutions are dependent on a number of factors which are problem dependent. We attempt to investigate these factors in this thesis.

### 1.4 Multigrid Overview

A two level multigrid method is presented first to illustrate the basic components and underlying ideas that will be expanded into the classical multigrid method.
Figure 1.1. Standard coarsening. Superimposed fine grid $G^h$ and coarse grid $G^H$, where the • indicates the coarse grid points in relation to the fine grid $G^h$.

Suppose that we have a continuous problem of the form

\begin{equation}
Lu(x, y) = f(x, y), \quad (x, y) \in \Omega \subset \mathbb{R}^2,
\end{equation}

where $L$ is a linear positive definite operator defined on an appropriate set of functions in $(0, 1)^2 = \Omega \subset \mathbb{R}^2$. Let $G^h$ and $G^H$ be two uniform grids for the discretization of $\Omega$; then

\begin{equation}
G^h = \{(x, y) \in \Omega : (x, y) = (ih, jh), \; i, j = 0, \ldots, n\}
\end{equation}

and

\begin{equation}
G^H = \{(x, y) \in \Omega : (x, y) = (iH, jH) = (i2h, j2h), \; i, j = 0, \ldots, \frac{n}{2}\},
\end{equation}

where the number of grid cells $n$ on $G^h$ is even with grid spacing $h = 1/n$, and where grid $G^H$ has $n/2$ grid cells with grid spacing $H = 2h$.

The coarse grid $G^H$ is often referred to as a standard coarsening of $G^h$; see figure 1.1. However, this choice is not the only one possible. Another popular choice is semi-coarsening, which coarsens in only one dimension; see figure 1.2. For the overview, only standard coarsening will be used.
Figure 1.2. Semi-coarsening. Superimposed fine grid $G^h$ and coarse grid $G^H$, where the $\bullet$ indicates the coarse grid points in relation to the fine grid $G^h$. 
The discrete problems now take the form
\[ L^h u^h = f^h \quad \text{on } G^h \]  \tag{1.23}
and
\[ L^H u^H = f^H \quad \text{on } G^H. \]  \tag{1.24}
We refer to \( L^h \) and \( L^H \) as the fine and coarse grid operators respectively. The grid operators are positive definite, linear operators
\[ L^h : G^h \to G^h, \]  \tag{1.25}
and
\[ L^H : G^H \to G^H. \]  \tag{1.26}

Let \( U^h \) be an approximation to \( u^h \) from equation (1.23). Denote the error \( e^h \) by
\[ e^h = U^h - u^h; \]  \tag{1.27}
thus \( e^h \) can also be regarded as a correction to \( U^h \). The residual (defect) of equation (1.23) is given by
\[ r^h = f^h - L^h U^h. \]  \tag{1.28}
The defect equation (error–residual equation) on grid \( G^h \)
\[ L^h e^h = r^h \]  \tag{1.29}
is equivalent to the original fine grid equation (1.23). The defect equation and its approximation play a central role in the development of a multigrid method.

The fine grid equation (1.23) can be approximately solved using an iterative method such as Gauss–Seidel. The first few iterations reduce the error quickly, but then the reduction in the error slows down for subsequent iterations. The slowing down in
the reduction of the error after the initial quick reduction is a property of most regular splitting methods and of most basic iterative methods. These methods reduce the error associated with high frequency (rough) components of the error quickly, but the low frequency (smooth) components are reduced very little. Hence, the methods seem to converge quickly for the first few iterations, as the high frequency error components are eliminated, but then the convergence rate slows down towards its asymptotic value as the low frequency components are slowly reduced. The idea behind the multigrid method is to take advantage of this behavior in the reduction of the error components. The point is that a few iterations of the relaxation method on $G^h$ effectively eliminate the high frequency components of the error.

Further relaxation on the fine grid results in little gain towards approximating the solution. However, the smooth components of the error on the fine grid are high frequency components with respect to the coarse grid. So, let us project the defect equation, since it is the error that we are interested in resolving, onto the coarse grid from the fine grid. This projection is done by using a restriction operator to project the residual, $r^h$, onto the coarse grid, where we can form a new defect equation

$$L^H v^H = I^H r^h = f^H, \quad (1.30)$$

where $I^H$ is the restriction operator. We can now solve this equation for $v^H$. Having done so, we can project the solution back up to the fine grid with a prolongation (interpolation) operator, $I_H^h$, and correct the solution on the fine grid, $G^h$,

$$U^h \leftarrow U^h + I_H^h v^H \quad (1.31)$$

We call this process (of projecting the error from the coarse grid to the fine grid and correcting the solution there) the coarse grid correction step. The process of projecting the error from a coarse grid to a fine grid introduces high frequency errors. The high
frequencies introduced by prolongation can be eliminated by applying a few iterations of a relaxation scheme. The relaxation scheme can be applied to the projection of the error, \( I_h^v v^H \), or to the approximation to the solution, \( U^h \), after the correction. It is desirable to apply the relaxation to \( U^h \) instead of \( I_h^v v^h \) since then additional reduction of the smooth components of the error in the solution may be obtained.

The projection operator from the fine grid to the coarse grid is called the restriction operator, while the projection operator from the coarse grid to the fine grid is called the prolongation operator or, interchangeably, the interpolation operator. These two operators are referred to as the grid transfer operators.

In the two level scheme just described, it can be seen that the coarse grid problem is the same, in form, as the fine grid problem with \( u^h \) and \( f^h \) being replaced by \( v^H \) and \( f^H = I_h^H r^h \) respectively. We can now formulate the classical multigrid method by applying the above two level scheme recursively. In doing so, we no longer solve the coarse grid defect equation exactly. Instead, we use the relaxation scheme on the coarse grid problem, where now, the smooth (low) frequencies from the fine grid appear to be higher frequencies with respect to the coarse grid. The relaxation scheme now effectively reduces the error components of these, now, higher frequencies. The coarse grid problem now looks like the fine grid problem, and we can project the coarse grid residual to an even coarser grid where a new defect equation is formed to solve for the error. The grid spacing in this yet coarser grid is \( 2H \). After sufficiently many recursions of the two level method, the resulting grid will have too few grid points to be reduced any further. We call this grid level the coarsest grid. We can either use relaxation or a direct solver to solve the coarsest grid problem. The approximate solution is then propagated back up to the fine grid, using the coarse grid correction step recursively.

What we have described informally is one multigrid V-cycle. More formally,
let us number the grid levels from 1 to $M$, where grid level 1 is the coarsest and grid level $M$ is the finest.

**Algorithm 1.4.1** ($\text{MGV}(k, \nu_1, \nu_2, h)$)

1. relax $\nu_1$ times on $L^k U^k = F^k$
2. compute the residual, $r^k = F^k - L^k U^k$
3. restrict the residual $I_{k-1}^{k-1} r^k$ to $G^{k-1}$, $F^{k-1} = I_{k-1}^{k-1} r^k$ and form the coarse grid problem (defect equation) $L^{k-1} U^{k-1} = F^{k-1}$, where $v^k = I_{k-1}^k U^{k-1}$ and $h^{k-1} = 2 h^k$.
4. IF $(k - 1) \neq 1$ THEN call Algorithm $\text{MGV}(k - 1, \nu_1, \nu_2, H)$
5. solve $L^{k-1} U^{k-1} = F^{k-1}$ to get the solution $u^{k-1}$
6. interpolate the defect (coarse grid solution) to the fine grid, and correct the fine grid solution, $U^k \leftarrow U^k + I_{k-1}^k u^{k-1}$
7. relax $\nu_2$ times on $L^k U^k = F^k$
8. IF (finest grid) THEN Stop

This algorithm describes the basic steps in the multigrid method for one iteration of a V-cycle. If the algorithm uses bi-linear (tri-linear in 3D) interpolation, it is called the classical multigrid method. This algorithm assumes that the coarsening is done by doubling the fine grid spacing, which can be seen in step 3 of the algorithm. However, the algorithm is valid for any choice of coarsening, $h^{k-1} = m h^k$, where $m$ is any integer greater than one.

**1.4.1 Multigrid Cycling Strategies** There are many different types of cycling strategies that are used in multigrid methods besides the V-cycle. We illustrate the different cycling types with the use of a few pictures and brief descriptions.
Figure 1.3. One V-cycle iteration for five grid levels, where the • represent a visit to a grid level.
The V-cycle is illustrated graphically in figure 1.3. The • represents a visit to a particular grid level. A slanting line connection between two grid levels indicates that smoothing work is to be performed. A vertical line connection between grid levels means that no smoothing is to take place between grid level visits. The grid levels are indicated by a numerical value listed on the left side of the figure, where grid level 1 is the coarsest grid level and is always placed at the bottom of the diagram.

The mechanics of the V-cycle were described in the multigrid algorithm in the last section. The V-cycle is one of the most widely used multigrid cycling strategies. Its best performance can be realized when there is an initial guess of the solution available. When a guess is not available a common choice is to use a zero initial guess or to use an F-cycle (see below).

The S-cycle is illustrated in figure 1.4. The “S” stands for “sawtooth”, because that is what it resembles; it is clearly a V(0, 1)-cycle and thus a special case of a V-cycle. The S-cycle is what De Zeeuw’s MGD9V [24] black box multigrid code uses for its cycling strategy. The S-cycle usually requires a smoother with a very good smoothing factor in order to be efficient and competitive with other cycling strategies.

The W-cycle is illustrated in figure 1.5. The W-cycle is sometimes called a 2-cycle; similarly, a V-cycle can be called a 1-cycle. From the figure 1.5, one can see the W type structure. It is called a 2-cycle because there must be two visits to the coarsest grid level before ascending to the next finer intermediate fine grid level. An intermediate fine grid level is one that is not the finest nor coarsest grid level and where the algorithm switches from ascending to descending based on the number times the grid level has been visited since the residual was restricted to it from a finer grid.

The F-cycle is illustrated in figure 1.6 and is called a full multigrid cycle. The figure shows a full multigrid V-cycle, that is, each sub-cycle that visits the coarsest grid level is a V-cycle. An F-cycle can also be created using a W-cycle, or any other
Figure 1.4. One S-cycle iteration for four grid levels, where the \( \bullet \) represent a visit to a grid level.

Figure 1.5. One W-cycle iteration for four grid levels, where the \( \bullet \) represent a visit to a grid level.
Figure 1.6. One F-cycle iteration for five grid levels, where the • represent a visit to a grid level.
type of cycling, for its sub-cycle. The F-cycle is very good when an initial guess for the multigrid iteration is not available, since it constructs its own initial guess. The F-cycle first projects the fine grid problem down to the coarsest grid level and then proceeds to construct a solution by using sub-cycles. After the completion of each sub-cycle the solution on an intermediate fine grid level is interpolated up to the next finer grid level where a new sub-cycle begins. This process is continued until the finest grid level is reached and its own V-cycle completed. At this point if more multigrid iterations are needed then the V-cycling is continued at the finest grid level.

1.5 Black Box Multigrid

Black box multigrid is also called geometric multigrid by some and is a member of the algebraic multigrid method (AMG) family. The distinguishing feature of black box multigrid is that the black box approach makes several assumptions about the type of problem to be solved and the structure of the system of equations. The black box multigrid methods also have a predetermined coarsening scheme where the coarse grid has roughly half as many grid points as the fine grid does in one or more of the coordinate directions. For a uniform grid, this means that $H = 2h$. Both methods automatically generate the grid transfer operators, prolongation $I_{k-1}^k$ and restriction $I_{k-1}^k$ for $2 \leq k \leq M$, and the coarse grid operators $L^k$ for $1 \leq k < M - 1$. The coarse grid operators are formed using the Galerkin coarse grid approximation,

$$L_{k-1}^k = I_{k-1}^k L^k I_{k-1}^k,$$

where $k = 1, \ldots, M - 1$. The algebraic multigrid methods deal with the system of equations in a purely algebraic way. The coarsening strategy for general AMG is not fixed nor is the formation of the grid transfer operators, resulting in methods that can be highly adaptable. However, the more adaptable a method is, the more complex its
implementation is likely to be, and it may also be less efficient due to its complexity. Another disadvantage of general AMG methods is that the coarse grid problems are usually not structured even when the fine grid problem is; moreover, the unstructured matrices on coarser levels tend to become less and less sparse, the coarser the grid level.

To define the black box multigrid method we need to define several of the multigrid components, such as the grid transfer operators, the coarse grid operators, the type of smoother employed, and the coarsest grid solver. We can also mention the type of cycling strategies that are available and other options.

There are several different grid transfer operators that we have developed and used in our codes. They are of two basic types. The first type collapses the stencil of the operator in a given grid coordinate direction to form three point relations, and the second is based on ideas from S. Schaffer [69]. The details of the grid transfer operators will be presented in chapter 3.

The coarse grid operators are formed by using the Galerkin coarse grid approximation given in equation (1.32).

There are several choices for the smoothing operator available in our codes. The smoothers that we have chosen are all of the multi–color type, except for incomplete line LU. For standard coarsening versions, the choices are point Gauss–Seidel, line Gauss–Seidel, alternating line Gauss–Seidel, and incomplete line LU. The semi–coarsening version uses either line Gauss–Seidel by lines in the x-direction or incomplete line LU. The smoothers will be presented in more detail in chapter 4.

In the standard coarsening codes, the coarsest grid solver is a direct solver using LU factorization. The semi–coarsening version allows the option of using line Gauss–Seidel relaxation.

There are several cycling strategies that are allowed, and they are chosen by input parameters. The most important choice is whether to choose full multigrid
cycling or not. There is also a choice for $N$-cycling, where $N = 1$ is the standard $V$-cycle and $N = 2$ is the $W$-cycle, etc... For more details, see section (1.4.1) above.
CHAPTER 2

DISCRETIZATIONS: FINITE DIFFERENCE AND FINITE VOLUME

This chapter presents some of the discretizations that can be used on the convection–diffusion equation. We present only some of the more common finite difference and finite volume discretizations. Although this section may be considered elementary, it was thought to be important for two reasons. First, it shows some of the range of discrete problems that can be solved by the black box multigrid methods. Secondly, it gives sufficient detail for others to be able to duplicate the results presented in this thesis. The sections on the finite volume method present more than is needed, but because there is very little on this topic in the current literature and because of its importance for maintaining $O(h^2)$ accurate discretizations for interface problems, we have decided to include it. For references on the finite volume discretization see [85] and [52].

The continuous two dimensional problem is given by

$$-\nabla \cdot (D \cdot \nabla u) + b \cdot \nabla u + cu = f, \quad \text{in } \Omega = (0, M_x) \times (0, M_y) \quad (2.1)$$

where $D$ is a $2 \times 2$ tensor,

$$D = \begin{bmatrix} D_x & D_{xy} \\ D_{yx} & D_y \end{bmatrix} \quad (2.2)$$

and $\text{det } D > 0$, $c \geq 0$. In general, $D_{xy} \neq D_{yx}$, but we only consider either $D_{xy} = D_{yx}$ or $D_{xy} = D_{yx} = 0$. In addition, $D$, $c$, and $f$ are allowed to be discontinuous across
internal interfaces in the domain $\Omega$. The boundary conditions are given by
\[
\frac{\partial u}{\partial n} + a u = g, \quad \text{on } \partial \Omega
\] (2.3)
where $a$ and $g$ are functions, and $n$ is the outward unit normal vector. This allows us to represent Dirichlet, Neumann, and Robin boundary conditions.

The domain is assumed to be rectangular, $\Omega = (0, M_x) \times (0, M_y)$, and is then divided into uniform cells of length $h_x = M_x/N_x$ by $h_y = M_y/N_y$, where $N_x$ and $N_y$ are the number of cells in the $x$- and $y$-directions respectively. A uniform grid is not required, but we will use it to simplify our discussions.

It should be noted that finite elements on a regular triangulation can also be used to derive the discrete system of equations to be solved by the black box multigrid methods. However, we will not present any details on how to derive these equations.

2.1 Finite Difference Discretization

The finite difference approach to discretization is well known. Finite difference approximation is based on Taylor’s series expansion. In one dimension, if a function $u$ and its derivatives are single valued, finite, and continuous functions of $x$, then we have the Taylor’s series expansions,

\[
u(x + h) = u(x) + hu'(x) + \frac{1}{2}h^2u''(x) + \frac{1}{6}h^3u'''(x) + \ldots \] (2.4)

and

\[
u(x - h) = u(x) - hu'(x) + \frac{1}{2}h^2u''(x) - \frac{1}{6}h^3u'''(x) + \ldots \] (2.5)

If we add equations (2.4) and (2.5) together we get an approximation to the second derivative of $u$, given by,

\[
u''(x) \approx \frac{1}{h^2} (u(x + h) - 2u(x) + u(x - h)) \] (2.6)
where the leading error term is $O(h^2)$. Subtracting equation (2.5) from (2.4) gives
\[
u'(x) \approx \frac{1}{h} (u(x + h) - u(x - h)),
\]
(2.7)
with an error of $O(h^2)$. Both equations (2.6) and (2.7) are said to be central difference approximations. We also derive a forward and backward difference approximation to the first derivative from equations (2.4) and (2.5):
\[
u'(x) \approx \frac{1}{h} (u(x + h) - u(x))
\]
(2.8)
and
\[
u'(x) \approx \frac{1}{h} (u(x) - u(x - h))
\]
(2.9)
respectively, with an error of $O(h)$.

The above approximations can be extended to higher dimensions easily and form the basis for finite difference approximation. We illustrate the finite difference discretization, using stencil notation, by way of examples for some of the types of problems that we are interested in. There are many references on finite differences if one is interested in more details; see for instance [74] [39].

The first example is for the anisotropic Poisson’s equation on a square domain,
\[
Lu \equiv -\varepsilon u_{xx} - u_{yy} = f \quad \text{in } \Omega = (0,1)^2,
\]
(2.10)
where $u$ and $f$ are functions of $(x,y) \in \Omega$. Using central finite differences and discretization on a uniform grid with grid spacing $h = 1/N$ for $N = n_x = n_y$, gives the 5-point stencil,
\[
L^h \approx \frac{1}{h^2} \begin{bmatrix}
-1 \\
-\varepsilon & 2(1 + \varepsilon) & -\varepsilon \\
-\varepsilon & -1
\end{bmatrix}
\]
(2.11)
The second example is for the convection–diffusion equation on a square domain,

$$Lu \equiv -\varepsilon \Delta u + b_x u_x + b_y u_y = f \quad (x, y) \in \Omega = (0, 1)^2 \quad (2.12)$$

where $u$, $b_x$, $b_y$, and $f$ are functions of $x$ and $y$. Using a mix of upstream and central finite differences and discretizing on a uniform grid with grid spacing $h = 1/N$ for $N = n_x = n_y$, gives the 5-point stencil,

$$L^h \approx \frac{1}{h^2} \begin{bmatrix}
-\varepsilon + b_y h \mu_y \\
-\varepsilon + b_x h (\mu_x - 1) & \sum \mu_x & -\varepsilon + b_x h \mu_x \\
-\varepsilon + b_y h (\mu_y - 1)
\end{bmatrix} \quad (2.13)$$

where

$$\sum = 4\varepsilon + b_x h(2\mu_x - 1) + b_y h(2\mu_y - 1) \quad (2.14)$$

and

$$\mu_x = \begin{cases}
\frac{\varepsilon}{2b_x h} & b_x h \geq \varepsilon \\
1 + \frac{\varepsilon}{2b_x h} & b_x h < \varepsilon \\
\frac{1}{2} & |b_x h| \leq \varepsilon
\end{cases} \quad \mu_y = \begin{cases}
\frac{\varepsilon}{2b_y h} & b_y h \geq \varepsilon \\
1 + \frac{\varepsilon}{2b_y h} & b_y h < -\varepsilon \\
\frac{1}{2} & |b_y h| \leq \varepsilon \quad (2.15)
\end{cases}$$

The third example is the rotated anisotropic diffusion equation on a square domain. It has this name because it is obtained from the second example by rotating the axes through an angle of $\theta$. The equation is given by

$$Lu \equiv -\varepsilon c^2 + s^2 \frac{\partial^2 u}{\partial x^2} - 2(\varepsilon - 1) cs \frac{\partial^2 u}{\partial x \partial y} - (\varepsilon s^2 + c^2) \frac{\partial^2 u}{\partial y^2} = 0 \quad (2.16)$$

where $c = \cos \theta$, $s = \sin \theta$, and $\varepsilon > 0$. There are two parameters, $\varepsilon$ and $\theta$, that can be varied. There are two popular discretizations of this equation which are seen in real
world applications. They differ only in the discretization of the cross derivative term. Let

\[ \alpha = (\varepsilon c^2 + s^2) \quad \beta = (\varepsilon - 1) cs \quad \gamma = (\varepsilon s^2 + c^2); \]  

then if the grid spacing is \( h = 1/N \) for \( N = n_x = n_y \), the first, a 7-point finite difference stencil, is

\[ L^h \approx \frac{1}{h^2} \begin{bmatrix} \beta & -\beta - \gamma \\ -\alpha - \beta & 2(\alpha + \beta + \gamma) & -\alpha - \beta \\ -\beta - \gamma & \beta \end{bmatrix}. \]  

The second, a 9-point finite difference stencil, is,

\[ L^h \approx \frac{1}{h^2} \begin{bmatrix} \frac{1}{2} \beta & -\gamma & -\frac{1}{2} \beta \\ -\alpha & 2(\alpha + \gamma) & -\alpha \\ -\frac{1}{2} \beta & -\gamma & \frac{1}{2} \beta \end{bmatrix}. \]  

The fourth example is the convection–diffusion equation on a square domain,

\[ Lu \equiv -\varepsilon \Delta u + cu_x + su_y = 0 \quad (x, y) \in \Omega = (0, 1)^2 \]  

where \( c = \cos \theta, \ s = \sin \theta, \) and \( \varepsilon > 0 \). Upstream finite differences and discretization on a uniform grid with grid spacing \( h = 1/N \) for \( N = n_x = n_y \), yields

\[ L^h \approx \frac{1}{h^2} \begin{bmatrix} -\varepsilon + \frac{h}{2} (s - |s|) \\ -\varepsilon - \frac{h}{2} (c + |c|) & 4\varepsilon + h (|c| + |s|) & -\varepsilon + \frac{h}{2} (c - |c|) \\ -\varepsilon - \frac{h}{2} (s + |s|) \end{bmatrix}. \]  

### 2.2 Finite Volume Discretization

There are two types of computational grids that will be considered. The first type is the vertex centered grid \( G_v \), defined as
Figure 2.1. Vertex centered finite volume grid, where the \( \bullet \) indicates where the discretization is centered and the dashed lines delineate the finite volumes.
where \(N_x\) and \(N_y\) are the number of cells in the \(x\) and \(y\) directions respectively, see figure 2.1. The second type is the cell centered grid \(G_c\) which is defined by

\[
G_c = \left\{(x_i, y_j) : \begin{array}{l}
x_i = (i - \frac{1}{2}) h_x, \quad i = 1, \ldots, N_x, \\
y_j = (j - \frac{1}{2}) h_y, \quad j = 1, \ldots, N_y \end{array} \right\},
\] (2.23)

where \(N_x\) and \(N_y\) are the number of cells in the \(x\) and \(y\) directions respectively, see figure 2.2.

There are two other somewhat common finite volume grids that will not be discussed here, but can be used to derive the discrete system of equations to be solved by the black box multigrid methods. These grids are defined by placing the finite volume cell centers on the grid lines in one of the coordinate directions and centered between the grid lines in the other coordinate direction. For instance, align the cell centers with the \(y\) grid lines and centered between \(x\) grid lines. The cell edges will then correspond with \(x\) grid lines and centered between \(y\) grid lines.

We will present finite volume discretization for both vertex and cell centered finite volumes where the coefficients are evaluated at either the vertices or cell centers.
The coefficients could be evaluated at other points, such as cell edges, but we will not show the development of such discretizations because they follow easily from the descriptions given below.

2.3 Cell Centered Finite Volume Discretization; Evaluation at the Vertices

For the cell centered finite volume discretization the cell has its center at the point \((i - \frac{1}{2})h_x, (j - \frac{1}{2})h_y\) and the cell is called the finite volume, \(\Omega_{i,j}\), for the point \((i, j)\) on the computational grid \(G_c\), where \(i = 1, \ldots, N_x\) and \(j = 1, \ldots, N_y\); see equation (2.23). A finite volume is shown in figure 2.3. The approximation of \(u\) in the center of the cell is called \(u_{i,j}\). The coefficients are approximated by constant values in the finite volume \(\Omega_{i,j}\). This discretization is useful when the discontinuities are not aligned with the finite volume cell boundaries.

Assume that \(D_{xy} = D_{yx} = 0\) and that \(b = 0\) for now. If we integrate equation (2.1) over the finite volume \(\Omega_{i,j}\) and use Green’s theorem we get

\[
- \int_{\partial \Omega_{i,j}} D_x \frac{\partial u}{\partial x} n_x + D_y \frac{\partial u}{\partial y} n_y d\Gamma + \int_{\Omega_{i,j}} c u d\Omega = \int_{\Omega_{i,j}} f d\Omega, \tag{2.24}
\]

where \(n_x\) and \(n_y\) are the components of the outward normal vector to the boundary \(\partial \Omega_{i,j}\).

We proceed by developing the equations for the interior points \(u_{i,j}\), and then for the boundary points, where we present the modifications that are needed for the three types of boundary conditions that we consider. We refer to figure 2.3 to aid in the development of the finite volume discretization.
Figure 2.3. Cell centered finite volume $\Omega_{i,j}$, where $P$ has the coordinates \((i - \frac{1}{2})h_x, (j - \frac{1}{2})h_y\).
2.3.1 Interior Finite Volumes  Referring to figure 2.3, we write the line integral from equation (2.24) as

\[-\int_{\partial\Omega_{i,j}} D_x \frac{\partial u}{\partial x} n_x + D_y \frac{\partial u}{\partial y} n_y \, d\Gamma = \int_{sw}^{se} D_y \frac{\partial u}{\partial y} \, dx - \int_{se}^{ne} D_x \frac{\partial u}{\partial x} \, dy]

\[+ \int_{ne}^{nw} D_y \frac{\partial u}{\partial y} \, dx - \int_{nw}^{sw} D_x \frac{\partial u}{\partial x} \, dy. \tag{2.25}\]

The integral from (sw) to (se) can be approximated by

\[\int_{sw}^{se} D_y(sw) \frac{\partial u}{\partial y} \, dx + \int_{se}^{sw} D_y(se) \frac{\partial u}{\partial y} \, dx \approx \frac{h_x}{2h_y} (D_y(sw) + D_y(se)) (u_{i,j} - u_{i,j-1}) \tag{2.26}\]

\[= \frac{h_x}{h_y} \alpha_{i,j-1}^x (u_{i,j} - u_{i,j-1})\]

where \(\alpha_{i,j}^x = \frac{1}{4} (D_y i,j + D_y i-1,j)\), and \(D_y i,j\) is the value of \(D_y\) at the point \((i,j)\).

The other line integrals of \(\Omega_{i,j}\), \((se)\) to \((ne)\), \((ne)\) to \((nw)\), and \((nw)\) to \((sw)\), can be approximated in a similar fashion.

The surface integrals in equation (2.24) can be approximated by:

\[\int_{\Omega_{i,j}} cu \, d\Omega = h_x h_y \tilde{c}_{i,j} u_{i,j} \tag{2.27}\]

and

\[\int_{\Omega_{i,j}} f \, d\Omega = h_x h_y \tilde{f}_{i,j}, \tag{2.28}\]

where \(\tilde{c}_{i,j}\) and \(\tilde{f}_{i,j}\) are approximations of \(c\) and \(f\), respectively, at the grid point \(((i - \frac{1}{2})h_x, (j - \frac{1}{2})h_y)\), given by

\[\tilde{c}_{i,j} = \frac{1}{4} (c_{i,j} + c_{i-1,j} + c_{i-1,j-1} + c_{i,j-1}) \tag{2.29}\]

and

\[\tilde{f}_{i,j} = \frac{1}{4} (f_{i,j} + f_{i-1,j} + f_{i-1,j-1} + f_{i,j-1}) \tag{2.30}\]
respectively. The resulting stencil for interior points is

\[
\begin{pmatrix}
\frac{h_z}{h_x} \alpha^y_{i,j} \\
-\frac{h_x}{h_y} \alpha^x_{i-1,j} + \sum h_x h_y \tilde{c}_{i,j} - \frac{h_y}{h_x} \alpha^x_{i,j} \\
-\frac{h_x}{h_y} \alpha^y_{i,j-1}
\end{pmatrix},
\]

(2.31)

where

\[
\alpha^x_{i,j} = \frac{1}{2} (D_{x,i,j-1} + D_{x,i,j})
\]

(2.32)

\[
\alpha^y_{i,j} = \frac{1}{2} (D_{y,i-1,j} + D_{y,i,j})
\]

(2.33)

and

\[
\sum = \frac{h_x}{h_y} (\alpha^y_{i,j-1} + \alpha^y_{i,j}) + \frac{h_y}{h_x} (\alpha^x_{i-1,j} + \alpha^x_{i,j}).
\]

(2.34)

At an interface, the diffusivity is given as an arithmetic mean of the diffusion coefficients of adjacent finite volumes. The arithmetic makes sense because the interface passes through the finite volume. This discretization is most accurate when the interface passes directly through the cell of the finite volume.

When the finite volume \( \Omega_{i,j} \) has an edge on the boundary, the line integral in equation (2.24) for that edge has to be treated differently. We examine what needs to be done for each of the three different types of boundary conditions. We examine the changes that are needed only on one boundary edge, and the other changes needed for the other boundary edges follow in a similar fashion.

### 2.3.2 Dirichlet Boundary Condition

Let us examine the south boundary, \((sw) - (se)\), where we have

\[
u_{(s)} = g_{(s)}.
\]

(2.35)
The line integral from \((sw)\) to \((se)\) is approximated by
\[
\int_{sw}^{se} D_y \frac{\partial u}{\partial y} dx \approx \frac{1}{2} \frac{h_x}{h_y} (D_{y,i,j-1} + D_{y,i-1,j-1}) \left( u_{i,j} - u_{i,j-1} \right). 
\] (2.36)

This gives the stencil
\[
\begin{pmatrix}
\frac{h_x}{h_y} \alpha_{i,j}^x \\
\frac{h_y}{h_x} \alpha_{i-1,j}^y \\
\sum + h_x h_y \tilde{c}_{i,j} - \frac{h_y}{h_x} \alpha_{i,j}^y \\
0
\end{pmatrix},
\] (2.37)

where \(\sum\) is defined in equation (2.34) and \(\alpha\) is defined by equation (2.32) and (2.33).

### 2.3.3 Neumann and Robin Boundary Conditions

We examine the south boundary, \((sw)\)--\((se)\), where
\[
\left( \frac{\partial u}{\partial n} + a \ u \right)_{(s)} = g_{(s)}. 
\] (2.38)

We then make the approximation
\[
u_{(s)} - u_{(p)} \approx \frac{1}{2} \frac{h_x}{h_y} \frac{\partial u}{\partial n}_{(s)}
\]
\[
= \frac{1}{2} h_y \left( g_{(s)} - a_{(s)} u_{(s)} \right). 
\] (2.39)

Solving for \(u_{(s)}\) gives
\[
u_{(s)} = \frac{\frac{1}{2} h_y g_{(s)} + u_{(p)}}{1 + \frac{1}{2} h_y a_{(s)}},
\] (2.40)

The line integral is then approximated as
\[
\int_{sw}^{se} D_y \frac{\partial u}{\partial y} dx \approx \frac{h_x a_{i,j-1}^y}{h_y} \left( g_{(s)} - a_{(s)} u_{(s)} \right) 
\] (2.41)
Now we substitute equation (2.40) to obtain

\[
\int_{sw}^{se} D_y \frac{\partial u}{\partial y} \, dx \approx \frac{2}{2 + h_y a_{(s)}} \alpha_{i,j-1}^y \left( a_{(s)} u_{i,j} - g_{(s)} \right)
\]

(2.42)

The resulting stencil for the south boundary is

\[
\begin{bmatrix}
\frac{h_y \alpha_{i,j}^y}{h_x} & \sum h_x h_y c_{i,j} + \frac{2}{2 + h_x a_{0,j}} \alpha_{i-1,j}^x - \frac{h_y}{h_x} \alpha_{i,j}^x \\
-\frac{h_x \alpha_{i-1,j}^x}{h_y} & 0
\end{bmatrix}
\]

(2.43)

where \( \alpha \) is defined in equations (2.32) and (2.33), and \( \sum \) is now given by

\[
\sum = \frac{h_x \alpha_{i,j}^y}{h_y} + \frac{h_y}{h_x} \left( \alpha_{i-1,j}^x + \alpha_{i,j}^x \right)
\]

(2.44)

The other boundaries can be handled in the same way. We have now defined the cell centered finite volume discretization where the coefficients are evaluated at the grid vertices.

### 2.4 Cell Centered Finite Volume Discretization; Evaluation at the Cell Centers

This discretization is better suited to problems when the interfaces align with the boundaries of the finite volumes. The discretization is very similar to what was done in section 2.3, except that now the coefficients are evaluated at the cell centers, \((i - \frac{1}{2})h_x, (j - \frac{1}{2})h_y\), of the finite volume \( \Omega_{i,j} \). The coefficients are approximated by constant values in the finite volume \( \Omega_{i,j} \). We need to approximate the integrals in equation (2.24).
2.4.1 Interior Finite Volumes

We have the line integral, as in equation (2.25), and the integral from \((sw)\) to \((se)\) can be approximated by

\[
\int_{sw}^{se} D_y \frac{\partial u}{\partial y} dx \approx \frac{2}{h_y} h_x D_{y,i,j} \left( u_{i,j} - u_{(s)} \right)
\]  

(2.45)

where \(D_{y,i,j}\) is the value of \(D_y\) at the point \((i,j)\). We still need to approximate \(u_{(s)}\), and to do this we will use the continuity of \(u\) and \(D_y \frac{\partial u}{\partial y}\),

\[
D_{y,i,j} \left( u_{i,j} - u_{(s)} \right) = D_{y,i,j-1} \left( u_{(s)} - u_{i,j-1} \right),
\]

(2.46)

yielding

\[
u_{(s)} = \frac{D_{y,i,j} u_{i,j} + D_{y,i,j-1} u_{i,j-1}}{D_{y,i,j} + D_{y,i,j-1}}.
\]

(2.47)

We can now substitute equation (2.46) into equation (2.45) to get

\[
\int_{sw}^{se} D_y \frac{\partial u}{\partial y} dx \approx \frac{h_x}{h_y} \alpha_{i,j-1}^y \left( u_{i,j} - u_{i,j-1} \right),
\]

(2.48)

where \(\alpha_{i,j-1}^y\) is now given by

\[
\alpha_{i,j-1}^y = \frac{2 D_{y,i,j} D_{y,i,j-1}}{D_{y,i,j} + D_{y,i,j-1}}.
\]

(2.49)

The other line integrals of \(\Omega_{i,j}\), \((se)\) to \((ne)\), \((ne)\) to \((nw)\), and \((nw)\) to \((sw)\), can be approximated in a similar fashion.

The surface integrals are approximated in the same way as before,

\[
\int_{\Omega_{i,j}} c u d \Omega = h_x h_y c_{i-\frac{1}{2},j-\frac{1}{2}} u_{i,j}
\]

(2.50)

and

\[
\int_{\Omega_{i,j}} f d \Omega = h_x h_y f_{i-\frac{1}{2},j-\frac{1}{2}};
\]

(2.51)

but instead of \(c_{i,j}\) and \(f_{i,j}\) we have \(c_{i-\frac{1}{2},j-\frac{1}{2}}\) and \(f_{i-\frac{1}{2},j-\frac{1}{2}}\).
The resulting stencil for interior points is

\[
\begin{bmatrix}
-\frac{h_y}{h_x} \alpha^y_{i,j} \\
-\frac{h_x}{h_y} \alpha^x_{i-1,j} + \sum h_x h_y c_{i,j} - \frac{h_y}{h_x} \alpha^x_{i,j} \\
-\frac{h_x}{h_y} \alpha^y_{i,j-1}
\end{bmatrix},
\]

where

\[
\alpha^x_{i-1,j} = \frac{2}{D_{x,i,j} + D_{x,i-1,j}} D_{x,i,j}
\]

(2.53)

\[
\alpha^y_{i,j-1} = \frac{2}{D_{y,i,j} + D_{y,i,j-1}} D_{y,i,j}
\]

(2.54)

and

\[
\sum = \frac{h_y}{h_x} (\alpha^x_{i-1,j} + \alpha^x_{i,j}) + \frac{h_x}{h_y} (\alpha^y_{i,j-1} + \alpha^y_{i,j})
\]

(2.55)

At an interface, the diffusivity is given as a harmonic average of the diffusion coefficients of the adjacent finite volumes.

### 2.4.2 Dirichlet Boundary Condition

For the south boundary, \((sw)\) to \((se)\), the Dirichlet boundary condition, \(u_{(s)} = g_{(s)}\). The line integral is approximated by

\[
\int_{sw}^{se} D_y \frac{\partial u}{\partial y} dx \approx \frac{2}{h_y} D_{y,i,j} \left( u_{i,j} - g_{(s)} \right).
\]

(2.56)

The stencil is then given by

\[
\begin{bmatrix}
-\frac{h_y}{h_x} \alpha^y_{i,j} \\
-\frac{h_x}{h_y} \alpha^x_{i-1,j} + \sum h_x h_y c_{i,j} + \frac{2 h_x}{h_y} D_{y,i,j} - \frac{h_y}{h_x} \alpha^x_{i,j} \\
0
\end{bmatrix},
\]

(2.57)

where \(\sum\) is given in equation (2.55) and \(\alpha\) is given by equation (2.53) and (2.54).
2.4.3 Neumann and Robin Boundary Conditions

The Neumann and Robin boundary conditions can be handled in the same way as in section 2.3.3. The line integral for the south boundary is

$$\int_{sw}^e D_y \frac{\partial u}{\partial y} dx \approx \frac{2}{2 + h_y a_{(s)}} D_{y,i,j} \left( a_{(s)} u_{i,j} - g_{(s)} \right).$$  \hfill (2.58)

The resulting stencil is now

$$
\begin{bmatrix}
-\frac{h_x}{h_y} \alpha_{i,j}^y \\
-\frac{h_y}{h_x} \alpha_{i-1,j}^x \\
\sum + h_x h_y c_{i,j} + \frac{2}{2 + h_y a_{(s)}} D_{y,i,j} - \frac{h_y}{h_x} \alpha_{i,j}^x \\
0
\end{bmatrix},$$  \hfill (2.59)

where $\sum$ is given in equation (2.55) and $\alpha$ is given by equation (2.53) and (2.54).

2.5 Vertex Centered Finite Volume Discretization – Evaluation at the Vertices

In this discretization $D$, $c$, and $f$ are approximated by constant values in finite volume, $\Omega_{i,j}$, whose centers are at the vertices. This discretization is useful when the discontinuities align with the boundaries of the finite volumes.

2.5.1 Interior Finite Volumes

The development is done the same as before for the cell centered cases; see section (2.3.1). The stencil, when $D_{xy} = D_{yx} = 0$ and $b = 0$, is given by

$$
\begin{bmatrix}
-\frac{h_x}{h_y} \alpha_{i,j}^y \\
-\frac{h_y}{h_x} \alpha_{i-1,j}^x \\
\sum + h_x h_y c_{i,j} - \frac{h_y}{h_x} \alpha_{i,j}^x \\
-\frac{h_y}{h_x} \alpha_{i,j-1}
\end{bmatrix}.$$  \hfill (2.60)
where

\[ \alpha_{i,j}^x = \frac{2 D_{x,i,j} D_{x,i+1,j}}{D_{x,i,j} + D_{x,i+1,j}} \]  \hspace{1cm} (2.61) \\

\[ \alpha_{i,j}^y = \frac{2 D_{y,i,j} D_{y,i+1,j}}{D_{y,i,j} + D_{y,i+1,j}} \]  \hspace{1cm} (2.62)

and

\[ \sum = \frac{h_x}{h_y} \left( \alpha_{i,j-1}^y + \alpha_{i,j}^y \right) + \frac{h_y}{h_x} \left( \alpha_{i-1,j}^x + \alpha_{i,j}^x \right) \]  \hspace{1cm} (2.63)

where \( c \) and \( f \) are evaluated at the grid point \((i h_x, j h_y)\).

### 2.5.2 Edge Boundary Finite Volumes

Let the finite volume \( \Omega_{i,j} \) have its southern edge, \( (sw)-(se) \) at the southern boundary \( (y = 0) \) of the domain; see figure 2.4.

### 2.5.3 Dirichlet Boundary Condition

For the Dirichlet boundary condition we have \( u_{(s)} = g_{(s)} \), and we can just eliminate the unknown \( u_{(s)} \) and move it to the right-hand side of the equation.

### 2.5.4 Neumann and Robin Boundary Conditions

The line integral along the boundary is approximated by

\[ \int_{sw}^{ee} D_y \frac{\partial u}{\partial y} \, dx \approx h_x D_{y,i,j} \frac{\partial u}{\partial y}_{(s)} \]
Figure 2.5. Southwest corner finite volume, where the • indicates where the discretization is centered.

\[
= -h_x D_{y,i,j} \left( g(s) - a(s)u_{i,j} \right) \tag{2.64}
\]

and now we need to look at the surface integrals

\[
\int_{\Omega_{i,j}} c u \, d\Omega \approx \frac{1}{2} h_x h_y c_{i,j} u_{i,j} \tag{2.65}
\]

and similarly for \( f \). The stencil for the edge boundary is given by

\[
\begin{bmatrix}
-\frac{h_x}{h_y} \alpha_{i,j} \\
-\frac{h_y}{h_x} \alpha_{i-1,j} \\
\sum + \frac{1}{2} h_x h_y c_{i,j} + h_x a(s) D_{y,i,j} - \frac{h_y}{h_x} \alpha_{i,j} \\
0
\end{bmatrix}, \tag{2.66}
\]

where

\[
\sum = \frac{h_x}{h_y} \alpha_{i,j} + \frac{h_y}{h_x} \left( \alpha_{i-1,j} + \alpha_{i,j} \right), \tag{2.67}
\]

and \( \alpha \) is defined by equations (2.61) and (2.62).

2.5.5 Corner Boundary Finite Volumes The corner finite volume discretization will be shown for the southwest corner of the computational grid; see figure (2.5).
2.5.6 Dirichlet Boundary Condition  In the Dirichlet boundary condition case, the unknown \(u_{(sw)}\) is eliminated by the boundary condition equation,

\[ u_{(sw)} = g_{(sw)}. \tag{2.68} \]

The term \(g_{(sw)}\) is incorporated into the right hand side of the discrete system of equations. The stencil for the southwest corner is

\[
\begin{bmatrix}
-\frac{h_y}{2h_y} \alpha^y_{i,j} \\
0 + \frac{h_x}{2h_x} \alpha^x_{i,j} - \frac{h_y}{2h_y} \alpha^y_{i,j} \\
0
\end{bmatrix}
\tag{2.69}
\]

where \(\sum\) is defined as

\[
\sum = -\frac{h_x}{2h_y} \alpha^y_{i,j} - \frac{h_y}{2h_x} \alpha^x_{i,j}
\]

and \(\alpha\) is defined by equations (2.61) and (2.62).

2.5.7 Neumann and Robin Boundary Conditions  In the Neumann and Robin boundary condition cases, we have

\[
\begin{align*}
\left[ -\frac{\partial u}{\partial x} + a_w u \right]_{(sw)} &= g_w, \tag{2.71} \\
\left[ -\frac{\partial u}{\partial y} + a_s u \right]_{(sw)} &= g_s, \tag{2.72}
\end{align*}
\]

where the subscripts \((sw)\) means evaluation at the \(sw\)-point; see figure 2.5. The line integrals around the finite volume are approximated by

\[
\begin{align*}
\int_{sw}^{se} D_y \frac{\partial u}{\partial x} \, dx &\approx \frac{1}{2} h_x D_{y,i,j} \frac{\partial u_{(sw)}}{\partial y} \\
&\approx \frac{1}{2} h_x D_{y,i,j} (a_w (sw) u_{i,j} - g_w (sw)) 
\tag{2.73}
\end{align*}
\]

\[
\begin{align*}
\int_{sw}^{nw} D_x \frac{\partial u}{\partial x} \, dy &\approx \frac{1}{2} h_y D_{x,i,j} \frac{\partial u_{(sw)}}{\partial y} \\
&\approx \frac{1}{2} h_y D_{x,i,j} (a_w (sw) u_{i,j} - g_w (sw)) 
\tag{2.74}
\end{align*}
\]

45
\[- \int_{x_x}^{x_{x+1}} D_x \frac{\partial u}{\partial x} \, dx \approx \frac{1}{2h_x} \alpha^{x}_{i,j} (u_{i,j} - u_{i+1,j}) \]  
(2.75)
\[- \int_{x_y}^{x_{y+1}} D_y \frac{\partial u}{\partial y} \, dy \approx \frac{1}{2h_y} \alpha^{y}_{i,j} (u_{i,j} - u_{i,j+1}). \]  
(2.76)

The stencil for the southwest corner is
\[
\begin{bmatrix}
- \frac{h_y}{2h_y} \alpha^{y}_{i,j} \\
0 \sum + \frac{1}{4} h_x h_y c_{i,j} + \bar{B}C - \frac{h_y}{2h_x} \alpha^{x}_{i,j} \\
0
\end{bmatrix}
\]  
(2.77)

where \( \sum \) is defined in equation (2.70), \( \alpha \) is defined by equations (2.61) and (2.62), and
\[
\bar{B}C = \frac{1}{2} (h_x D_{y,i,j} a_i(sw) + h_y D_{x,i,j} a_w(sw)).
\]  
(2.78)

### 2.6 Vertex Centered Finite Volume Discretization – Evaluation at the Cell Vertices

In this discretization \( D, c, \) and \( f \) are approximated by constant values in finite volume, \( \Omega_{i,j} \), whose centers are at the vertices. This discretization is useful when the discontinuities pass through the interior of the finite volumes, and best when the interface passes through the cell center.

#### 2.6.1 Interior Finite Volumes

The development is the same as for the previous section on vertex centered finite volumes; see section 2.5. The stencil, when \( D_{xy} = D_{yx} = 0 \) and \( b = 0 \), is given by
\[
\begin{bmatrix}
- \frac{h_x}{h_y} \alpha^{y}_{i,j} \\
- \frac{h_y}{h_x} \alpha^{x}_{i-1,j} \\
0 \sum + h_x h_y c_{i,j} - \frac{h_y}{h_x} \alpha^{x}_{i,j} \\
- \frac{h_x}{h_y} \alpha^{y}_{i,j-1}
\end{bmatrix}
\]  
(2.79)
where

\[
\alpha_{i,j}^x = \frac{1}{2} (D_{x,i+1,j} + D_{x,i+1,j+1})
\]  
(2.80)

\[
\alpha_{i,j}^y = \frac{1}{2} (D_{y,i,j+1} + D_{y,i+1,j+1})
\]  
(2.81)

and

\[
\sum = \frac{h_x}{h_y} (\alpha_{i,j-1}^y + \alpha_{i,j}^y) + \frac{h_y}{h_x} (\alpha_{i-1,j}^x + \alpha_{i,j}^x)
\]  
(2.82)

and where \( c \) and \( f \) are evaluated at the grid point \((i h_x, j h_y)\).

\[
\tilde{c}_{i,j} = \frac{1}{4} (c_{i-1,j-1} + c_{i+1,j-1} + c_{i-1,j+1} + c_{i+1,j+1})
\]  
(2.83)

and

\[
\tilde{f}_{i,j} = \frac{1}{4} (f_{i-1,j-1} + f_{i+1,j-1} + f_{i-1,j+1} + f_{i+1,j+1})
\]  
(2.84)

Let the finite volume \( \Omega_{i,j} \) have its southern edge, \((sw)-(se)\) at the southern boundary \((y = 0)\) of the domain; see figure 2.4.

### 2.6.2 Dirichlet Boundary Condition

For the Dirichlet boundary condition we have \( u_{(s)} = g_{(s)} \), and we can just eliminate the unknown \( u_{(s)} \) and move it to the right-hand side of the equation.

### 2.6.3 Neumann and Robin Boundary Conditions

The line integral along the boundary is approximated by

\[
\int_{sw}^{ne} D_y \frac{\partial u}{\partial y} dx \approx -h_x \alpha_{i,j}^y \frac{\partial u}{\partial n}_{(s)}
\]

\[
= -h_x \alpha_{i,j}^y \left( g_{(s)} - a_{(s)} u_{i,j} \right),
\]  
(2.85)

\[
\int_{se}^{ne} D_x \frac{\partial u}{\partial x} dy \approx -\frac{h_y}{2h_x} D_{y,i+1,j} (u_{i+1,j} - u_{i,j})
\]
and similarly for the line integral from \((sw)-(nw)\), and the line integral from \((nw)-(ne)\) is done as before for the interior.

The surface integrals are now given by

\[
\int_{\Omega_{i,j}} c \ u \ d\Omega \approx \frac{1}{2} h_x h_y \tilde{c}_{i,j} u_{i,j}
\]

where

\[
\tilde{c}_{i,j} = \frac{1}{2} (c_{i,j+1} + c_{i+1,j+1})
\]

and similarly for \(\tilde{f}\). The stencil for the edge boundary is given by

\[
\begin{bmatrix}
-\frac{h_y}{h_x} \alpha_{i,j}^y \\
-\frac{h_x}{2h_x} D_{x,i-1,j} + \frac{h_x}{2h_x} \sum + \frac{h_y}{2h_x} \tilde{c}_{i,j} + h_x a_{i,j} \alpha_{i,j}^y - \frac{h_x}{2h_x} D_{x,i,j} \\
0
\end{bmatrix},
\]

where

\[
\sum = \frac{h_x}{h_y} \alpha_{i,j}^y + \frac{h_y}{2h_x} (D_{x,i-1,j} + D_{x,i,j}),
\]

and \(\alpha\) is defined by equations (2.80) and (2.81).

### 2.6.4 Corner Boundary Finite Volumes

The corner finite volume discretization will be shown for the southwest corner of the computational grid; see figure (2.5).

### 2.6.5 Dirichlet Boundary Condition

In the Dirichlet boundary condition case, the unknown \(u_{(sw)}\) is eliminated by the boundary condition equation, \(u_{(sw)} = g_{(sw)}\). The term \(g_{(sw)}\) is incorporated into the right hand side of the discrete
system of equations. The stencil for the southwest corner is

\[
\begin{pmatrix}
  -\frac{h_x}{2h_y} D_{y,i,j} \\
  0 \\
  0
\end{pmatrix} + \frac{1}{4} h_x h_y \tilde{c}_{i,j} - \frac{h_y}{2h_x} D_{x,i,j}
\]

(2.90)

where \( \Sigma \) is defined as

\[
\sum = -\frac{h_x}{2h_y} D_{y,i,j} - \frac{h_y}{2h_x} D_{x,i,j}
\]

(2.91)

and \( \alpha \) is defined by equations (2.80) and (2.81).

### 2.6.6 Neumann and Robin Boundary Conditions

In the Neumann and Robin boundary condition cases, we have

\[
\begin{alignat}{2}
\left[-\frac{\partial u}{\partial x} + a_w u\right]_{(sw)} &= g_w \quad (2.92) \\
\left[-\frac{\partial u}{\partial y} + a_s u\right]_{(sw)} &= g_s, \quad (2.93)
\end{alignat}
\]

where the subscripts \((sw)\) means evaluation at the \(sw\)-point; see figure 2.5. The line integrals around the finite volume are approximated by

\[
\int_{sw}^{nw} D_y \frac{\partial u}{\partial y} dx \approx \frac{1}{2} h_x D_{y,i+1,j+1} \frac{\partial u(sw)}{\partial y}
\]

\[
\approx \frac{1}{2} h_x D_{y,i+1,j+1} (a_w(sw)u_{i,j} - g_w(sw)) \quad (2.94)
\]

\[
\int_{sw}^{nw} D_x \frac{\partial u}{\partial x} dy \approx \frac{1}{2} h_y D_{x,i+1,j+1} \frac{\partial u(sw)}{\partial x}
\]

\[
\approx \frac{1}{2} h_y D_{x,i+1,j+1} (a_w(sw)u_{i,j} - g_w(sw)) \quad (2.95)
\]

\[
- \int_{se}^{ne} D_x \frac{\partial u}{\partial x} dy \approx \frac{h_y}{2h_x} D_{x,i+1,j+1} (u_{i,j} - u_{i+1,j}) \quad (2.96)
\]

\[
- \int_{nw}^{ne} D_y \frac{\partial u}{\partial y} dx \approx \frac{h_x}{2h_y} D_{y,i+1,j+1} (u_{i,j} - u_{i,j+1}). \quad (2.97)
\]
The surface integrals are approximated by

\[
\int_{\Omega_{i,j}} cu d\Omega \approx \frac{1}{4} h_x h_y c_{i+1,j+1} u_{i,j}
\]  

and similarly for \( f \). The stencil for the southwest corner is

\[
\begin{bmatrix}
-\frac{h_x}{2h_y} D_{y,i+1,j+1} \\
0 & \sum +\frac{1}{2} h_x h_y c_{i+1,j+1} + BC - \frac{h_y}{2h_x} D_{x,i+1,j+1} \\
0
\end{bmatrix}
\]

(2.99)

where \( \sum \) is defined in equation (2.91), \( \alpha \) is defined by equations (2.80) and (2.81), and

\[
BC = \frac{1}{2} \left( h_x D_{y,i+1,j+1} a_s (sw) + h_y D_{x,i+1,j+1} a_w (sw) \right).
\]

(2.100)
CHAPTER 3

PROLONGATION AND RESTRICTION OPERATORS

Suppose that we have an elliptic linear operator \( L \) on a two dimensional rectangular domain \( \Omega \):

\[
Lu = f \quad \text{in} \quad \Omega \subset \mathbb{R}^2.
\]  \hfill (3.1)

This problem can be discretized using finite differences (or other discretization) on a rectangular grid \( G^h \) with grid spacing \( h \), given by

\[
L^h u^h = f^h \quad \text{in} \quad G^h,
\]  \hfill (3.2)

\[
G^h = \{(x_i, y_j) : x_i = x_0 + i h, \quad y_j = y_0 + j h\}
\]  \hfill (3.3)

We assume that the discretization is represented in stencil notation as

\[
\left[ L^h \right]_{(i,j)} = \begin{bmatrix}
NW & N & NE \\
W & C & E \\
SW & S & SE
\end{bmatrix}_{(i,j)}
\]  \hfill (3.4)

where \( NW, N, NE, \ldots \) are the coefficients of the discretization stencil centered at \((x_i, y_j)\).

The size of the fine grid operator’s stencil is important to remember because we require that the coarser grid operator’s stencil not be any larger than the largest allowable fine grid operator stencil. By keeping the grid operator stencil fixed at a maximum of 9-points, we ensure that the implementation will be easier and more efficient by maintaining the sparsity of the operators. This consideration is important.
when discussing the formation of the grid transfer operators since we use the Galerkin coarse grid approximation approach to form the coarse grid operators. The formulation of the coarse grid operators involves the multiplication of three matrices, and if their stencils are at most 9-point, then the coarse grid operator will also be at most 9-point. If we use grid transfer operators with larger stencils, the size of the coarse grid operator stencil can grow without bound, as the grids levels became coarser, until the stencils either become the size of the full matrix or we run out of grid levels.

Another guiding principal that we follow is that if we are given a symmetric fine grid operator we would like all the coarser grid operators to be symmetric also. In order to follow this principal the interpolation and restriction operators must be chosen with care.

Before getting started it would be best to show where and how the operators are used to transfer components between grid levels. We assume the layout of coarse and fine grids shown in figure 1.1. We refer to coarse grid points with indices \((i_c, j_c)\) and fine grid points with indices \((i_f, j_f)\).

### 3.1 Prolongation

We interpolate the defect correction (error) from the coarse grid level to the fine grid level, where it is added as a correction to the approximation of the fine grid solution. There are four possible interpolation cases for standard coarsening in two dimensions. The four cases are illustrated in figure 3.1, where the thick lines represent coarse grid lines, thin lines represent the fine grid lines, circles represent coarse grid points, X represents the fine grid interpolation point, and the subscripts \(f\) and \(c\) distinguish the fine and coarse grid indices respectively. Figure 3.1(a) represents interpolation to fine grid points that coincide with coarse grid points. Figure 3.1(b) represents interpolation to fine grid points that do not coincide with coarse grid points,
Figure 3.1. The four 2D standard coarsening interpolation cases, where • represents the coarse grid points used to interpolate to the fine grid point represented by ×. The thick lines represent coarse grid lines.
but lie on coarse grid lines in the x-direction. Figure 3.1(c) represents interpolation to fine grid points that do not coinciding with coarse grid points, but lie on coarse grid lines in the y-direction. Figure 3.1(d) represents interpolation to fine grid points that do not align with any coarse grid lines either horizontally or vertically.

The fine grid points that are also coarse grid points, case (a), use the identity as the interpolation operator. The coarse grid correction is then given by

$$u^h_{i,j} = u^h_{i,j} + u^H_{i,j}$$  \hspace{1cm} (3.5)$$

where \((x_{i,j}, y_{i,j}) = (x_{i_c}, y_{j_c})\) on the grid; here the interpolation coefficient is 1.

The fine grid points that are between two coarse grid points that share the same \(y_j\) coordinate, case (b), use a two point relation for the interpolation. The coarse grid correction is given by

$$u^h_{i-1,j} = u^h_{i-1,j} + I^w_{i-1,j} \cdot u^H_{i-1,j} + I^e_{i,j} \cdot u^H_{i,j}$$  \hspace{1cm} (3.6)$$

where \(y_{j_c} = y_{j_f}\) and \(x_{i-1,j} < x_{i,j} < x_{i_c}\) on the grid, and the interpolation coefficients are \(I^w_{i-1,j}\) and \(I^e_{i,j}\).

The fine grid points that are between two coarse grid points that share the same \(x_i\) coordinate, case (c), use a similar two point relation for the interpolation. The coarse grid correction is then given by

$$u^h_{i,j-1} = u^h_{i,j-1} + I^n_{i,j-1} \cdot u^H_{i,j-1} + I^s_{i,j} \cdot u^H_{i,j},-1$$  \hspace{1cm} (3.7)$$

where \(x_{i_c} = x_{i_f}\) and \(y_{j-1,j} < y_{j,f} < y_{j_c}\) on the grid, and the interpolation coefficients are \(I^n_{i,j-1}\) and \(I^s_{i,j}\).

The last set of fine grid points are those that do not share either a \(x_i\) or a \(y_j\) coordinate with the coarse grid, case (d). We use a four point relation for the interpolation in this case, and the coarse grid correction is given by

$$u^h_{i-1,j-1} = u^h_{i-1,j-1}$$
where \(x_{i_e} < x_{i_f} < x_{i_e+1}\) and \(y_{j_e} < y_{j_f} < y_{j_e+1}\), and the interpolation coefficients are \(I_{i_e-1,j_e-1}^{se}\), \(I_{i_e-1,j_e}^{ne}\), and \(I_{i_e,j_e-1}^{se}\). The interpolation operator’s coefficients can also be represented in stencil notation, just like the grid operator, as

\[
I^h_H = \begin{bmatrix}
I^{nw} & I^n & I^{ne} \\
I^w & 1 & I^n \\
I^{sw} & I^s & I^{se}
\end{bmatrix}_H^h.
\] (3.9)

3.1.1 Prolongation Correction Near Boundaries In the black box multigrid solvers, the right hand side of the grid equation next to the boundary can contain boundary data, in which case the above interpolation formulas can lead to \(O(1)\) interpolation errors. To improve this error we can use a correction term that contains the residual to bring the interpolation errors back to \(O(h^2)\); [26]. The correction term is \(O(h^2)\) for the interior grid points, and in general will not improve the error on the interior, but near the boundary the correction term can be of \(O(1)\). The correction term takes the form of the residual divided by the diagonal of the grid equation coefficient matrix; the correction term is equal to \(r_{i,j}/C_{i,j}\), where the residual was computed for the grid before restriction. The correction term is added to equations 3.6, 3.7, and 3.8, which are for interpolating to fine grid points that are not coarse grid points. Applying the correction is similar to performing an additional relaxation sweep along the boundary, and it does not affect the size of the prolongation stencil.
3.2 Restriction

The restriction operator restricts the residual from the fine grid level to the coarse grid level, where it becomes the right-hand-side of the defect equation (error-residual equation). The restriction equation is

\[
\begin{align*}
\mathbf{f}^H_{i_0,j_e} &= J^w_{i_0,j_e} \cdot r^h_{i_f,j_f+1} + J^e_{i_0,j_e} \cdot r^h_{i_f,j_f-1} \\
&+ J^s_{i_0,j_e} \cdot r^h_{i_f+1,j_f} + J^n_{i_0,j_e} \cdot r^h_{i_f,j_f-1} \\
&+ J^{sw}_{i_0,j_e} \cdot r^h_{i_f+1,j_f+1} + J^{nw}_{i_0,j_e} \cdot r^h_{i_f+1,j_f-1} \\
&+ J^{ne}_{i_0,j_e} \cdot r^h_{i_f-1,j_f} + J^{sw}_{i_0,j_e} \cdot r^h_{i_f-1,j_f+1} \\
&+ r^h_{i_f,j_f},
\end{align*}
\]

(3.10)

where the restriction coefficients are \(J^w, J^e, J^n, J^{sw}, J^{nw}, J^{ne}, J^{se}\), and 1. The restriction coefficients can also be represented in stencil notation as

\[
I^H_h = \begin{bmatrix}
J^{nw} & J^n & J^{ne} \\

J^w & 1 & J^e \\
J^{sw} & J^s & J^{se}
\end{bmatrix}^H,
\]

(3.11)

where the restriction is centered at the fine grid point \((x_{i_f}, y_{j_f}) = (x_{i_e}, y_{j_e})\).

3.3 Overview

In the following sections we present several different interpolation operators by exhibiting the coefficients needed to represent the operator’s stencil. In most cases, we omit the indices of the operators, it being be understood that the grid operator is given at the fine grid point \((x_{i_f}, y_{j_f})\). The grid transfer operators can be split into two groups based upon how the operators are computed.

The first class of grid transfer operators is based on using a collapse (lumping) in one of the coordinate directions, yielding a simple three point relation that can be
solved. The second class of grid transfer operators is based on an idea from Schaffer’s semi-coarsening multigrid [69]. Both these methods for operator induced grid transfer operators are an approximation to the Schur complement, that is, they try to approximate the block Gaussian elimination of the unknowns that are on the fine grid but not on the coarse grid. The collapsing methods are a local process while Schaffer’s idea is to apply the procedure to a block (line) of unknowns.

We start by presenting the grid transfer operators used in the symmetric versions of the black box multigrid solvers. Then we present several different grid transfer operators that are used in the nonsymmetric black box multigrid solvers.

In classic multigrid methods, the grid transfer operators are often taken to be bilinear interpolation and full weighting; injection is also popular. To see why we do not use these choices, we need to look at the type of problems that we are hoping to solve. These problems are represented by the convection–diffusion equation,

$$-\nabla \cdot (D \cdot \nabla u) + b \cdot \nabla u + c u = f,$$

where $D$, $c$, and $f$ are allowed to be discontinuous across internal boundaries. The black box multigrid solvers are aimed at solving these problems when $D$ is strongly discontinuous. The classical multigrid grid transfer operators perform quite well when $D$ jumps by an order of magnitude or less, but when $D$ jumps by several orders of magnitude, the classical methods can exhibit extremely poor convergence, since these methods are based on the continuity of $\nabla u$ and the smoothing of the error in $\nabla u$. However, it is $D \cdot \nabla u$ that is continuous, not $\nabla u$. Hence, if $D$ has jumps of more than an order of magnitude across internal boundaries, then it is more appropriate to use grid transfer operators that approximate the continuity of $D \cdot \nabla u$ instead of the continuity of $\nabla u$. It is important to remember that we are using the Galerkin coarse grid approximation approach to form the coarse grid operators. We want the coarse
grid operators to approximate the continuity of $D \cdot \nabla u$. This goal is accomplished by basing the grid transfer operators on the grid operator $L^h$.

Before proceeding with the definitions of the first class of grid transfer operators, we need to define a few terms and make a few explanations.

**Definition 3.3.1** Using the grid operator’s stencil notation, define $R_\sigma$, row sum, at a given grid point, $(x_i, y_j)$, to be

$$R_\Sigma = C + NW + N + NE + W + E + SW + S + SE,$$

where the subscript $(i, j)$ has been suppressed.

The row sum is used to determine when to switch between two different ways of computing the grid transfer coefficients at a given point. The switch happens when the grid operator is marginally diagonally dominant, or in other words, when the row sum is small in some sense.

We recall what is meant by the symmetric part of the operator.

**Definition 3.3.2** Define the symmetric part of the operator, $L$, as

$$\sigma L = \text{symm}(L) = \frac{1}{2} (L + L^*)$$

where $L^*$ is the adjoint of the grid operator $L$.

The notation applies equally to the grid operator’s coefficients, for example:

$$\sigma N_{i,j} = \frac{1}{2} (N_{i,j} + S_{i,j+1})$$

and

$$\sigma SW_{i,j} = \frac{1}{2} (SW_{i,j} + NE_{i-1,j-1})$$

In addition, we can give some examples of the adjoint (transpose) of the grid

58
operators coefficients are:

\[(W_{i,j})^* = E_{i-1,j},\]
\[(\sigma SE_{i,j})^* = \sigma NW_{i+1,j-1},\]
and
\[(\sigma C_{i,j})^* = C_{i,j}.\]  \hfill (3.16)

### 3.4 Symmetric Grid Operator \(L^h\): Collapsing Methods

The interpolation operator is based upon the discrete grid operator \(L^h\), while the restriction operator is based on \((L^h)^*\).

We want to preserve the flux \(\mu \cdot (D \cdot \nabla U)\) across interfaces, which can be done by using the grid operator \(L^h\). Assume that \(L^h\) has a 5-point stencil, then

\[W(U_{i,j} - U_{i-1,j}) = E(U_{i+1,j} - U_{i,j}),\]  \hfill (3.17)

which gives the interpolation formula

\[U_{i,j} = \frac{W}{W + E} U_{i-1,j} + \frac{E}{W + E} U_{i+1,j}.\]  \hfill (3.18)

When \(L^h\) has a 9-point stencil, the idea is to integrate the contributions from the other coefficients (NW, NE, SW, and SE), which can be done by summing (collapsing) the coefficients to get the three point relation,

\[A_- v_{i-1,j} + A_o v_{i,j} + A_+ v_{i+1,j} = 0\]  \hfill (3.19)

where \(A_- = (NW + W + SW), \ A_o = (N + C + S), \) and \(A_+ = (NE + E + SE).\)

The computation of the \(I^w\) and \(I^e\) coefficients are done by collapsing the grid operator in the y-direction to get a three point relation on the x-grid lines. Let the interpolation formula be given by

\[A_{i-1} v_{i-1} + A_i v_{i} + A_{i+1} v_{i+1} = 0\]  \hfill (3.20)
where \( v_k \) is written for \( v_{k,j} \), and \( A_{i-1} = (NW + W + SW)_{i,j} \), \( A_i = (N + C + S)_{i,j} \), and \( A_{i+1} = (NE + E + SE)_{i,j} \). We now solve the equation for \( v_i \) to get the interpolation formula in an explicit form.

\[
v_i = -A_i^{-1}A_{i-1}v_{i-1} - A_i^{-1}A_{i+1}v_{i+1}.
\]

(3.21)

The interpolation coefficients \( I^w \) and \( I^e \) are then given by

\[
I^w = -A_i^{-1}A_{i-1} \quad \text{and} \quad I^e = -A_i^{-1}A_{i+1}
\]

(3.22)

Writing out the coefficients explicitly gives

\[
I^w = -\frac{NW + W + SW}{N + C + S},
\]

(3.23)

\[
I^e = -\frac{NE + E + SE}{N + C + S},
\]

(3.24)

where \( I^w \) and \( I^e \) are evaluated at \((i_c - 1, j_c)\) and \((i_c, j_c)\) respectively, and the other coefficients on the right hand side are evaluated at \((i_f - 1, j_f)\). If, however, the row sum number, \( R_i \) (see 3.13), is small (see 3.28) then instead of \((N + C + S)_i \) for \( A_i \) we use \(-(NW + W + SW + NE + E + SE)_i \). These two formulas give the same result when the row sum is zero, which is the case for an operator with only second order terms away from the boundary. This idea is observed to lead to better convergence, and it is due to Dendy [30]. The coefficients are then defined by

\[
I^w = \frac{NW + W + SW}{NW + W + SW + NE + E + SE},
\]

(3.25)

and

\[
I^e = \frac{NE + E + SE}{NW + W + SW + NE + E + SE},
\]

(3.26)

where \( I^w \) and \( I^e \) are evaluated at \((i_c - 1, j_c)\) and \((i_c, j_c)\) respectively, and the other coefficients on the right hand side are evaluated at \((i_f - 1, j_f)\).
Let
\[ \gamma = \min \{ NW + W + SW, |NE + E + SE|, 1. \}. \] (3.27)

Then by \textit{small} we mean that
\[ R_{\Sigma} < -\gamma \cdot (NW + W + SW + N + S + NE + E + SE), \] (3.28)
where \( R_{\Sigma} \) is the row sum defined above.

The computation of the \( I^s \) and \( I^n \) coefficients is done by collapsing the grid operator in the \( x \)-direction to get the three point relation on the \( y \)-grid line. Let the interpolation formula be given by
\[ A_{j-1}v_{j-1} + A_jv_j + A_{j+1}v_{j+1} = 0 \] (3.29)
where \( v_{j-1} = \{ v_{i,j-1} : i = 1, \ldots, nx \}, v_i = \{ v_{i,j} : i = 1, \ldots, nx \}, v_{j+1} = \{ v_{i,j+1} : i = 1, \ldots, nx \}, \) and \( A_{j+1} = (NW + N + NE)_{i,j+1}, A_j = (W + C + E)_{i,j}, \) and \( A_{j-1} = (SW + S + SE)_{i,j-1}. \) We now solve the equation for \( v_j \) to get the interpolation formula in an explicit form:
\[ v_j = -A_j^{-1}A_{j-1}v_{j-1} - A_j^{-1}A_{j+1}v_{j+1} \] (3.30)

The interpolation coefficients \( I^s \) and \( I^n \) are given by
\[ I^s = -A_j^{-1}A_{j-1} \quad \text{and} \quad I^n = -A_j^{-1}A_{j+1} \] (3.31)

Writing out the coefficients explicitly gives
\[ I^s = -\frac{SW + S + SE}{W + C + E}, \] (3.32)
\[ I^n = -\frac{NE + N + NE}{W + C + E}. \] (3.33)
If, however, the row sum, $R_\Sigma$, is small, then instead of $(W + C + E)_j$ for $A_j$ we use $-(NW + N + NE + SW + S + SE)_j$. The coefficients are then defined by

$$I^s = \frac{SW + S + SE}{NW + N + NE + SW + S + SE}, \quad (3.34)$$

$$I^n = \frac{NW + N + NE}{NW + N + NE + SW + S + SE}, \quad (3.35)$$

where $I^s$ and $I^n$ are evaluated at $(i_c, j_c - 1)$ and $(i_c, j_c)$ respectively, and the other coefficients on the right hand side are evaluated at $(i_f, j_f - 1)$. Let

$$\gamma = \min \{|NW + N + NE|, |SW + S + SE|, 1.\}. \quad (3.36)$$

Then by small we mean that

$$R_\Sigma < -\gamma \cdot (NW + N + NE + SW + S + SE), \quad (3.37)$$

where $R_\Sigma$ is the row sum.

The computation of the interpolation coefficients $I^{sw}$, $I^{nw}$, $I^{ne}$, and $I^{se}$ is similar to that of the coefficients that have already been computed. Let the interpolation formula be given by

$$A_{i-1,j+1}v_{i-1,j+1} + A_{i,j+1}v_{i,j+1} + A_{i+1,j+1}v_{i+1,j+1}$$

$$+ A_{i-1,j}v_{i-1,j} + A_{i,j}v_{i,j} + A_{i+1,j}v_{i+1,j}$$

$$+ A_{i-1,j-1}v_{i-1,j-1} + A_{i,j-1}v_{i,j-1} + A_{i+1,j-1}v_{i+1,j-1} = 0. \quad (3.38)$$

where the $A_{s,s}$ are just the corresponding grid operator coefficients. We can now solve for $v_{i,j}$ to get the interpolation formula.

$$v_{i,j} = -A_{i,j}^{-1} \left( A_{i-1,j+1}v_{i-1,j+1} + A_{i,j+1}v_{i,j+1} + A_{i+1,j+1}v_{i+1,j+1} ight.$$ 

$$+ A_{i-1,j}v_{i-1,j} + A_{i,j}v_{i,j} + A_{i+1,j}v_{i+1,j}$$

$$+ A_{i-1,j-1}v_{i-1,j-1} + A_{i,j-1}v_{i,j-1} + A_{i+1,j-1}v_{i+1,j-1} \right) \quad (3.39)$$

$$62$$
Notice that $v_{i,j-1}$, $v_{i-1,j}$, $v_{i+1,j}$, and $v_{i,j+1}$ are unknowns. However, we can use their interpolated values that we computed above, being careful to note that their stencils are all centered at different grid points. After performing the substitutions and collecting the terms for $v_{i-1,j-1}$, $v_{i+1,j-1}$, $v_{i-1,j+1}$, and $v_{i+1,j+1}$ we get

$$v_{i,j} = I^{sw}v_{i-1,j-1} + I^{nw}v_{i-1,j+1} + I^{se}v_{i,j-1} + I^{ne}v_{i,j+1}, \quad (3.40)$$

where instead of having to compute everything all over again, it can be seen that $I^{sw}$, $I^{nw}$, $I^{ne}$, and $I^{se}$ can be expressed in terms of the previous four coefficients, $I^w$, $I^e$, $I^s$, and $I^n$. However, we must now explicitly write the subscripts for the coefficients $I^w$, $I^e$, $I^s$, and $I^n$ to indicate where their stencils are centered relative to the interpolated point's stencil, which is centered at $(i,j)$. The formulas for the four coefficients are

$$I^{sw} = -\frac{SW + S \cdot I^w_{i,j-1} + W \cdot I^s_{i-1,j}}{C}, \quad (3.41)$$

where $I^{sw}$ is evaluated at $(x_{i,-1}, y_{j,-1})$,

$$I^{nw} = -\frac{NW + N \cdot I^w_{i,j+1} + W \cdot I^n_{i-1,j}}{C}, \quad (3.42)$$

where $I^{nw}$ is evaluated at $(x_{i,-1}, y_{j,+1})$,

$$I^{ne} = -\frac{NE + N \cdot I^e_{i,j+1} + E \cdot I^n_{i+1,j}}{C}, \quad (3.43)$$

where $I^{ne}$ is evaluated at $(x_{i,+1}, y_{j,+1})$,

$$I^{se} = -\frac{SE + S \cdot I^e_{i,j-1} + E \cdot I^s_{i+1,j}}{C}, \quad (3.44)$$

where $I^{se}$ is evaluated at $(x_{i,+1}, y_{j,-1})$, and the other stencil coefficients are evaluated at $(x_{i,j}, y_{j})$. If, however, $R^\Sigma$ is small, then

$$I^{sw}_{i,-1,j,-1} = \frac{SW + S \cdot I^w_{i,-1,j-1} + W \cdot I^s_{i-1,j}}{NW + N + NE + W + E + SW + S + SE}, \quad (3.45)$$

$$I^{sw}_{i,+1,j,+1} = \frac{NW + N \cdot I^w_{i,+1,j+1} + W \cdot I^n_{i-1,j}}{NW + N + NE + W + E + SW + S + SE}, \quad (3.46)$$
\begin{align}
I^{ne}_{i,j} &= \frac{NE + N \cdot I^x_{i,j+1} + E \cdot I^y_{i+1,j}}{NW + N + NE + W + E + SW + S + SE}, \\
I^{nc}_{i,j-1} &= \frac{SE + S \cdot I^x_{i,j-1} + E \cdot I^y_{i+1,j}}{NW + N + NE + W + E + SW + S + SE},
\end{align}

and where \(NW, N, NE, W, C, E, SW, S,\) and \(SE\) are evaluated at \((x_{ij}, y_{ij})\). Let

\[
\gamma = \min \left\{ |SW + W + NW|, \quad |NW + N + NE|, \right. \\
\left. \quad |NE + E + SE|, \quad |SE + S + SW|, \quad 1. \right\}
\]  

(3.49)

Then by small we mean that

\[
R^\Sigma < -\gamma \cdot (NW + N + NE + W + E + SW + S + SE).
\]  

(3.50)

The interpolation correction terms are \(A^{-1}_i r^H, A^{-1}_j r^H,\) or \(A^{-1}_{i,j} r^H\) for the corresponding interpolation formulas above, where \(r^H\) is the residual on the coarse grid.

Note that the \(A\)'s change depending on whether \(R^\Sigma\) is small or not.

The computation of the interpolation coefficients in this way was used in the BOXMG, BOXMGP, BBMG, and BBMGP codes for symmetric problems \([1, 26, 30, 10]\). Similar computations have also been used for most black box, geometric, and algebraic multigrid solvers for symmetric problems arising from finite difference and finite volume discretizations using either a 5-point or a 9-point standard stencil \([7, 23, 29, 31, 52, 54, 53, 55, 63, 85, 24]\).

The computation of the restriction operator’s coefficients is closely related to that of the interpolation coefficients. In fact, in the symmetric case, the restriction coefficients for the symmetric grid operator \(L^h\) can be taken to be equal to the interpolation coefficients,

\[
I^H = \kappa \cdot (I^H) = \kappa \in \mathbb{R}.
\]  

(3.51)
3.5 Nonsymmetric Grid Operator $L^h$: Collapsing Methods

The interpolation coefficients can be computed in the same way as in the symmetric case except that we replace all of the grid operator's coefficients with their equivalent symmetric stencil coefficients, denoted by $\sigma(\cdot)$. However, the row sum $R_{\Sigma}$ definition remains unchanged.

3.5.1 Prolongation Based on $\text{symm}(L^h)$ The computation of the $I^w$ and $I^e$ coefficients is given by

\[
I^w = -\frac{\sigma NW + \sigma W + \sigma SW}{\sigma N + \sigma C + \sigma S}, \quad (3.52)
\]

\[
I^e = -\frac{\sigma NE + \sigma E + \sigma SE}{\sigma N + \sigma C + \sigma S}. \quad (3.53)
\]

If, however, $R_{\Sigma}$ is small, then

\[
I^w = \frac{\sigma NW + \sigma W + \sigma SW}{\sigma NW + \sigma W + \sigma SW + \sigma NE + \sigma E + \sigma SE}. \quad (3.54)
\]

\[
I^e = \frac{\sigma NE + \sigma E + \sigma SE}{\sigma NW + \sigma W + \sigma SW + \sigma NE + \sigma E + \sigma SE}. \quad (3.55)
\]

In (3.52)-(3.55) $I^w$ and $I^e$ are evaluated at $(x_{i,-1}, y_{j_c})$ and $(x_{i_c}, y_{j_c})$ respectively, and the other coefficients on the right hand side are evaluated at $(x_{i_f-1}, y_{j_f})$ for the $L^h$ components. Let

\[
\gamma = \min \{ |\sigma NW + \sigma W + \sigma SW|, |\sigma NE + \sigma E + \sigma SE|, 1.\}. \quad (3.56)
\]

Then by small we mean that

\[
R_{\Sigma} < -\gamma \cdot (\sigma NW + \sigma W + \sigma SW + \sigma N + \sigma S + \sigma NE + \sigma E + \sigma SE) \quad (3.57)
\]
The formulas for the $I^n$ and $I^s$ coefficients are

\[ I^n = \frac{\sigma NW + \sigma N + \sigma NE}{\sigma W + \sigma C + \sigma E}, \quad (3.58) \]

\[ I^s = \frac{\sigma SW + \sigma S + \sigma SE}{\sigma W + \sigma C + \sigma E}. \quad (3.59) \]

If, however, $R_\Sigma$ is small, then

\[ I^n = \frac{\sigma NW + \sigma N + \sigma NE}{\sigma NW + \sigma N + \sigma NE + \sigma SW + \sigma S + \sigma SE}, \quad (3.60) \]

\[ I^s = \frac{\sigma SW + \sigma S + \sigma SE}{\sigma NW + \sigma N + \sigma NE + \sigma SW + \sigma S + \sigma SE}, \quad (3.61) \]

where $I^n$ and $I^s$ are evaluated at $(x_{i,c}, y_{j,c})$ and $(x_{i,c}, y_{j,c}-1)$ respectively, and the other coefficients on the right hand side are evaluated at $(x_{i,f}, y_{j,f})$ for the $L^h$ components. Let

\[ \gamma = \min \{ |\sigma NW + \sigma N + \sigma NE|, |\sigma SW + \sigma S + \sigma SE|, 1. \}. \quad (3.62) \]

Then by small we mean that

\[ R_\Sigma < -\gamma \cdot (\sigma NW + \sigma N + \sigma NE + \sigma SW + \sigma S + \sigma SE). \quad (3.63) \]

The computation of the interpolation coefficients $I^{nw}$, $I^{nw}$, $I^{ne}$, and $I^{se}$ can be expressed in terms of the other four coefficients:

\[ I_{i-1,j-1}^{nw} = -\frac{\sigma SW + \sigma S \cdot I_{i,j-1}^{w} + \sigma W \cdot I_{i-1,j}^{s}}{C}, \quad (3.64) \]

\[ I_{i-1,j}^{nw} = -\frac{\sigma NW + \sigma N \cdot I_{i,j+1}^{w} + \sigma W \cdot I_{i-1,j}^{n}}{C}, \quad (3.65) \]

\[ I_{i,j+1}^{ne} = -\frac{\sigma NE + \sigma N \cdot I_{i,j+1}^{e} + \sigma E \cdot I_{i+1,j}^{n}}{C}, \quad (3.66) \]

\[ I_{i,j+1}^{se} = -\frac{\sigma SE + \sigma S \cdot I_{i,j-1}^{e} + \sigma E \cdot I_{i+1,j}^{s}}{C}. \quad (3.67) \]
If, however, $R_\Sigma$ is small, then

$$I_{i,j}^{sw} = \frac{\sigma SW + \sigma S \cdot I_{i,j-1}^{sw} + \sigma W \cdot I_{i,j}^{se}}{\sigma NW + \sigma N + \sigma NE + \sigma W + \sigma E + \sigma SW + \sigma S + \sigma SE},$$

$$I_{i,-1,j}^{nw} = \frac{\sigma NW + \sigma N \cdot I_{i+1,j}^{nw} + \sigma W \cdot I_{i,-1,j}^{nw}}{\sigma NW + \sigma N + \sigma NE + \sigma W + \sigma E + \sigma SW + \sigma S + \sigma SE},$$

$$I_{i,-1,j}^{ne} = \frac{\sigma NE + \sigma N \cdot I_{i+1,j}^{ne} + \sigma E \cdot I_{i,-1,j}^{ne}}{\sigma NW + \sigma N + \sigma NE + \sigma W + \sigma E + \sigma SW + \sigma S + \sigma SE},$$

$$I_{i,j,-1}^{se} = \frac{\sigma SE + \sigma S \cdot I_{i,j-1}^{se} + \sigma E \cdot I_{i,j+1}^{se}}{\sigma NW + \sigma N + \sigma NE + \sigma W + \sigma E + \sigma SW + \sigma S + \sigma SE},$$

where $\sigma NW$, $\sigma N$, $\sigma NE$, $\sigma W$, $\sigma C$, $\sigma E$, $\sigma SW$, $\sigma S$, and $\sigma SE$ are evaluated at $(x_i, y_j)$ for the $L^h$ components. Let

$$\gamma = \min \left\{ \frac{|\sigma SW + \sigma W + \sigma NW|}{|\sigma NE + \sigma E + \sigma SE|}, \frac{|\sigma NW + \sigma N + \sigma NE|}{|\sigma SE + \sigma S + \sigma SW|}, 1 \right\}. \quad (3.72)$$

Then by small we mean that

$$R_\Sigma < -\gamma \cdot (\sigma NW + \sigma N + \sigma NE + \sigma W + \sigma E + \sigma SW + \sigma S + \sigma SE). \quad (3.73)$$

It has been found in practice that the restriction operator $I_h^r$ need not be based on the same operator as the interpolation operator, so we change its symbol to be $J_h^r$ to reflect this change. The restriction operator’s coefficients are based on $(L^h)^T$ instead of $\sigma L^h$. The restriction coefficients are computed in exactly the same way as the interpolation coefficients except that all of the grid operator’s coefficients in the computations are replaced by their transposes. The computations for the restriction coefficients are now straightforward and will not be written out.

The grid transfer operators have been computed in this fashion for the black box multigrid solver for nonsymmetric problems [27]. It should be noted that when the grid operator $L^h$ is symmetric, then the computations given here for both the symmetric case and nonsymmetric case yield the same grid transfer coefficients.
3.5.2 Prolongation Based on $L^h$ and symm($L^h$) The third possibility for computing the grid transfer operators is one that uses the same form of the computations as above, see section 3.5.1. This prolongation is a point collapse approximation to Schaffer’s ideas; see section 3.6. The only difference in the above computations for the nonsymmetric case is that for the denominators, $A_i^{-1}$ and $A_j^{-1}$, we use the coefficients based on $L^h$ instead of $\sigma L^h$. The test for small is still in the same form as before except that $L^h$ is used, but $\gamma$ is still based on $\sigma L^h$.

The restriction operator coefficients are computed as before, but the denominator is now based on $L^h$ instead of on $(L^h)^T$.

3.5.3 Grid Transfer Operators Based on a hybrid form of $L^h$ and symm($L^h$) The prolongation operator coefficients are computed the same as in the last section 3.5.2. However, the computation of the restriction operator coefficients has been modified into a hybrid form that uses both $L^T$ and $L$.

The difference in the computation of the restriction coefficients comes into play when the switch is made in the denominator, $A_i^{-1}$ and $A_j^{-1}$, because the row sum is small. When the row sum is large we modify the denominator by adding in two coefficients from the grid operator $L$. We can illustrate this modification by computing the restriction coefficients $J^w$ and $J^e$.

\[
J^w = -\frac{(NW)^T + (W)^T + (SW)^T}{N + C + S}, \tag{3.74}
\]

\[
J^e = -\frac{(NE)^T + (E)^T + (SE)^T}{N + C + S}. \tag{3.75}
\]

If, however, $R_\Sigma$ is small, then

\[
J^w = \frac{(NW)^T + (W)^T + (SW)^T}{(NW)^T + (W)^T + (SW)^T + N + S + (NE)^T + (E)^T + (SE)^T}, \tag{3.76}
\]

68
\[ J^c = \frac{(NE)^T + (E)^T + (SE)^T}{(NW)^T + (W)^T + (SW)^T + N + S + (NE)^T + (E)^T + (SE)^T}. \] (3.77)

In (3.74)–(3.76) \( J^n \) and \( J^c \) are evaluated at \((x_{i-1}, y_{j})\) and \((x_i, y_{j+1})\) respectively, and the other coefficients on the right hand side are evaluated at \((x_{i-1}, y_{j+1})\) for the \( L^h \) components. Let

\[ \gamma = \min \left\{ \left| (NW)^T + (W)^T + (SW)^T \right|, \left| (NE)^T + (E)^T + (SE)^T \right|, 1. \right\}. \] (3.78)

Then by *small* we mean that

\[ R_{ij}^c < -\gamma \cdot (NW)^T + (W)^T + (SW)^T + (N)^T + (S)^T + N + S \]
\[ + (NE)^T + (E)^T + (SE)^T \] (3.79)

The restriction coefficients \( J^n \) and \( J^c \) are computed in a similar way.

The motivation behind these modifications was to try to get the coarse grid operator to approximate the one obtained when using the extension of Schaffer’s idea; see section 3.6. The grid operators from section 3.5.2 above were computed to approximate the grid transfer coefficients based on an extension of Schaffer’s idea; while the method in this section attempts to do the same thing, it also makes some modifications so that the coarse grid operator more closely approximates the one obtained in section 3.6.1.

### 3.6 Nonsymmetric Grid Operators: Extension of Schaffer’s Idea

The second class of grid transfer routines is based on Schaffer’s idea for grid transfer operators in his semi-coarsening multigrid method [70]. Schaffer’s idea is to approximate a full matrix by a diagonal matrix to compute the grid transfer operators.
Schaffer’s idea was used in the development of the semi-coarsening black box multi-grid method [32]. We took Schaffer’s idea and extended it to apply to the standard coarsening grid transfer operators.

The ideas used in the semi-coarsening method are as follows. Suppose that coarsening takes place only in the y-direction. Then the interpolation to points on the fine grid can be represented by

\[ A_{j-1}v_{j-1} + A_jv_j + A_{j+1}v_{j+1} = 0 \]  

where \( v_k = \{v_{i,k} : i = 1, \ldots, nx, j = j-1, j, j+1\} \). The tridiagonal matrices \( A_{j-1}, A_j, \) and \( A_{j+1} \) represent the nine point grid operator on the \( j-1, j, \) and \( j+1 \) grid lines respectively;

\[ A_{j+1} = \text{tridiag}[\text{NW, N, NE}]_{j+1} \]
\[ A_j = \text{tridiag}[\text{W, C, E}]_j \]
\[ A_{j-1} = \text{tridiag}[\text{SW, S, SE}]_{j-1} . \]

As before, we solve this equation for \( v_j \) to get,

\[ v_j = -A_j^{-1}A_{j-1}v_{j-1} - A_j^{-1}A_{j+1}v_{j+1} , \]  

where we have assumed that \( A_j^{-1} \) exists and can be stably inverted. This assumption can not always be guaranteed, but Schaffer’s and our methods allow line relaxation as a smoother, where these assumptions are necessary. The methods would fail if the assumptions did not hold, so in that sense we can say that the assumptions hold.

From equation (3.81), we form the quantities \( -A_j^{-1}A_{j-1} \) and \( -A_j^{-1}A_{j+1} \), leading to a non-sparse interpolation operator. If the interpolation operator is not sparse, that is, involves only \( v_{i,j-1} \) and \( v_{i,j+1} \) for interpolation at the point \((i,j)\), then the coarse grid operators formed by the Galerkin coarse grid approximation approach will grow

70
beyond a 9-point stencil. This is a property that we would very much like to avoid, since it would lead to full operators on the coarser grid levels. Schaffer’s idea, also arrived at independently by Dendy, is to approximate these quantities with diagonal matrices $B_{j-1}$ and $B_{j+1}$. This is accomplished by solving the following relations

\begin{align*}
-A_j^{-1}A_{j-1}e &= B_{j-1}e \\
-A_j^{-1}A_{j+1}e &= B_{j+1}e,
\end{align*}

where $e = (1, 1, \ldots, 1)^T$. They can be solved quickly because they are tridiagonal equations. After solving, the entries (diagonals) in $B_{j-1}$ and $B_{j+1}$ are just the interpolation coefficients $I^s$ and $I^n$ respectively.

In the semi-coarsening case the restriction operator is still based on the transpose of the nonsymmetric grid operator $L^h$. This is done by replacing $A_{j-1}$, $A_j$, and $A_{j+1}$ by their transpose to get $(A_{j-1})^*$, $(A_j)^*$, and $(A_{j+1})^*$ respectively.

3.6.1 Extension of Schaffer’s Idea to Standard Coarsening The above was presented in a manner suitable for the symmetric case. It can be modified for the nonsymmetric case, as we did for the collapsing methods, by using the symmetric part of the operator. We can do this by replacing $A_s$ with $\sigma A_s$ in equation (3.82) to get,

\begin{align*}
-(\text{symm}(A_j))^{-1} \text{symm}(A_{j-1}) e &= B_{j-1}e \\
-(\text{symm}(A_j))^{-1} \text{symm}(A_{j+1}) e &= B_{j+1}e.
\end{align*}

Schaffer constructs his grid transfer operators in a different manner and his construction for variable coefficient problems can yield a nonsymmetric coarse grid operator $L^H$ even if $L^h$ is symmetric. We would like the coarse grid operators to be symmetric whenever the fine grid operator is symmetric. We can do this is several
ways, but a more efficient construction is to replace equation (3.83) with

\[-A_j^{-1} \text{symm}(A_{j-1}) e = B_{j-1} e \]  

(3.84)

\[-A_j^{-1} \text{symm}(A_{j+1}) e = B_{j+1} e .\]

The advantage of this form is that it can use the same tridiagonal system solver that we are already using for the line solves for the multigrid smoother. Equation (3.83) will require an additional tridiagonal solve for \( \text{symm}(A_j) \) and additional storage if the LU factors are to be saved.

To extend these ideas to the standard coarsening case is quite easy. We first compute the grid transfer coefficients for semi-coarsening in the \( y \)-direction, and define

\[ v_k = \{ v_{i,k} : i = 1, \ldots, nx, k = j - 1, j, j + 1 \} \]

and the tridiagonal matrices

\[ A_{j+1} = \text{tridiag}[\sigma NW, \sigma N, \sigma NE]_{j+1} \]

\[ A_j = \text{tridiag}[W, C, E]_j \]

\[ A_{j-1} = \text{tridiag}[\sigma SW, \sigma S, \sigma SE]_{j-1} . \]

We save the diagonals of \( B_{j-1} \) and \( B_{j+1} \) associated with coarse grid lines in the \( x \)-direction as the \( I^s \) and \( I^n \) interpolation coefficients respectively.

To obtain the coefficients for the \( y \)-direction, we compute the grid transfer coefficients for semi-coarsening in the \( x \)-direction and define

\[ v_k = \{ v_{k,j} : k = i - 1, i, i + 1, j = 1, \ldots, ny \} \]

and the tridiagonal matrices

\[ A_{i+1} = \text{tridiag}[\sigma SW, \sigma W, \sigma NW]_{i+1} \]

\[ A_i = \text{tridiag}[S, C, N]_i \]

\[ A_{i-1} = \text{tridiag}[\sigma SE, \sigma E, \sigma NE]_{i-1} . \]
We save the diagonals of $B_{i-1}$ and $B_{i+1}$ associated with coarse grid lines in the x-direction as the $I^w$ and $I^e$ interpolation coefficients respectively.

Finally, we can then combine the semi-coarsening coefficients from the X and Y lines to obtain the $I^{sw}$, $I^{nw}$, $I^{ne}$, and $I^{se}$ interpolation coefficients. They can be computed as the product of the coefficients that have already been computed,

$$
I^{nw} = I^n \cdot I^w \quad I^{ne} = I^n \cdot I^e
$$
$$
I^{se} = I^s \cdot I^w \quad I^{se} = I^s \cdot I^e,
$$

or elimination can be used as before.

The restriction operator for the extension to the standard coarsening case is computed as above, but the transpose of the grid operator is used instead of the symmetric part of the operator. This is done by replacing $A_{j-1}$ and $A_{j+1}$ by their transpose to get $(A_{j-1})^t$ and $(A_{j+1})^t$ respectively.

### 3.7 Conclusions Regarding Grid Transfer Operators

Many other grid transfer operators were tried in the standard coarsening black box multigrid method in addition to the those presented above. However, only three were deemed to be robust and efficient enough to include in a release version of the solver. The three choices for grid transfer operators are the original nonsymmetric collapsing method described in section 3.5.1, the nonsymmetric hybrid collapsing method described in section 3.5.3, and the nonsymmetric extension to Schaffer’s ideas described in section 3.6.1. While all three of these choices are good, better results were obtained for the later two for all test and application problems run to date.

Most of the other grid transfer operators, that were tried had good performance on some of the test problems but failed on others. There does appear to be enough good results to cover all the test problems, with the exception of re-entrant flows. However, to unify these into one set of grid transfer operators would be much
more expensive to compute and may also introduce trouble when combining the various types of grid transfer operators.

The grid transfer operators from section 3.5.2, which use a collapsing method to try to approximate the extension of Schaffer’s ideas for nonsymmetric problems, were a disappointment. While they seemed to be a good idea, they turned out to not be very robust and in several cases actually caused divergence of the multigrid method. This bad behavior prompted examination of the coarse grid operators and grid transfer operators. After comparing the operators with those obtained from Schaffer’s ideas, it was noticed that several things were wrong, but with the modifications described in section 3.5.3, these problems were overcome. These new grid transfer operators extended Schaffer’s ideas to standard coarsening very well.
CHAPTER 4

BASIC ITERATION METHODS FOR SMOOTHERS

In this chapter we examine several basic iteration schemes for use as smoothers in the Black Box Multigrid solvers. Fourier mode analysis is used to identify which scheme makes the best smoother for a given type of model problem in two dimensions.

In this chapter we will be using parentheses around a superscript to denote an iteration index. For example: $u^{(n)}$ means the $n^{th}$ iterate.

4.1 Overview of Basic Iteration Methods

All of the methods in this section can be characterized in the following way. The algebraic system of equations to be solved is given by the matrix equation

$$Lu = f$$

(4.1)

The matrix $L$ is an $N_{xy} \times N_{xy}$ matrix, where $N_{xy} = n_x n_y$. The computational grid is two dimensional with $n_x$ and $n_y$ grid points in the $x$- and $y$-directions respectively. The matrix $L$ can be split as

$$L = M - N,$$

(4.2)

where $M$ is non-singular and assumed easy to invert. Then a basic iteration method for the solution of equation (4.1) is given by

$$Mu^{(n+1)} = Nu^{(n)} + f$$

(4.3)
or as

$$u^{(n+1)} = Su^{(n)} + M^{-1} f,$$  \hspace{1cm} (4.4)

where $S = M^{-1}N$ is called the iteration matrix. The basic iteration method can also be damped, and if the damping parameter is $\omega$, then the damped method is given by

$$u^{(n+1)} = \omega \left( M^{-1} Nu^{(n)} + M^{-1} f \right) + (1 - \omega) u^{(n)}$$  \hspace{1cm} (4.5)

or by

$$u^{(n+1)} = \tilde{S}u^{(n)} + \omega M^{-1} f,$$  \hspace{1cm} (4.6)

where $\tilde{S}$ is now given by

$$\tilde{S} = \omega M^{-1}N + (1 - \omega) I,$$  \hspace{1cm} (4.7)

and $I$ is the identity matrix. When $\omega = 1$ we recover the undamped basic iterative method.

The eigenvalues of the damped basic iteration matrix $\tilde{S}$ can be given in terms of the eigenvalues of the undamped basic iteration matrix $S$. They are related by

$$\lambda(\tilde{S}) = \omega \lambda(S) + 1 - \omega,$$  \hspace{1cm} (4.8)

where $\omega$ is the damping parameter and $\lambda(S)$ on the right hand side of the equation is an eigenvalue of $S$, the undamped iteration matrix.

The error after the $n$th iteration is

$$e^{(n)} = u^{(n)} - u$$  \hspace{1cm} (4.9)

where $u$ is a solution (unique if $L$ is non-singular) to equation (4.1). The error at the $(n + 1)$st iteration is related to the error at the $n$th iteration by

$$e^{(n+1)} = Se^{(n)},$$  \hspace{1cm} (4.10)
where $S$ is the iteration matrix defined above ($\bar{S}$ can also replace $S$ in the equation). From equation (4.10), it follows by induction, that $e^{(n)}$ can be written in terms of the original error, $e^{(0)}$, as

$$e^{(n)} = S^n e^{(0)} \quad (4.11)$$

where the superscript, $n$ on $S$ is now an exponent and $n \geq 0$. In terms of vector norms, we have

$$\|e^{(n)}\| = \|S^n e^{(0)}\| \leq \|S^n\| \|e^{(0)}\| \quad (4.12)$$

where $\| \cdot \|$ is any vector norm with induced matrix norm for $\|S^n\|$. The term $\|S\|$ is called the contraction number of the basic iterative method.

The spectral radius of $S$ is defined as

$$\rho(S) = \max_{1 \leq i \leq N_{xy}} |\lambda_i(S)| \quad (4.13)$$

The basic iterative method is said to be convergent if

$$\rho(S) < 1, \quad (4.14)$$

in which case, we have that

$$\lim_{n \to \infty} \|S^n\| = 0. \quad (4.15)$$

If the method is convergent, then this also implies that

$$\lim_{n \to \infty} \|e^n\| = 0 \quad (4.16)$$

for any initial choice of $e^{(0)}$ if and only if $\rho(S) = \max |\lambda(S)| < 1$. Some other useful results, that are similar to those found in Varga [82], are:
Theorem 4.1.1 If $\rho(S)$ is the spectral radius of $S$, then if

$$\rho(S) \leq \|S\|$$ (4.17)

then

$$[\rho(S)]^n = \rho(S^n) \leq \|S^n\|.$$(4.18)

Theorem 4.1.2 If $\rho(S) < 1$ then

$$\lim_{n \to \infty} (\|S^n\|)^{\frac{1}{n}} = \rho(S)$$ (4.19)

for any suitable induced matrix norm.

We can also define the reduction factor and the rate of convergence for basic iterative methods.

Definition 4.1.1 The reduction factor is defined as

$$\tau = \frac{\|e^{(n+1)}\|}{\|e^{(n)}\|} \leq \|S\|$$ (4.20)

where $\tau$ is the reduction factor per iteration.

Definition 4.1.2 The average reduction factor, $\bar{\tau}$, is given by

$$\bar{\tau} = \left(\frac{\|e^{(n)}\|}{\|e^{(0)}\|}\right)^{\frac{1}{n}} \leq \|S^n\|^{\frac{1}{n}}$$ (4.21)

after $n$ iterations.

Definition 4.1.3 The average rate of convergence can be defined as

$$R(S^n) \equiv -\frac{1}{n} \log (\|S^n\|)$$ (4.22)

where $n > 0$ and $\|S^n\| < 1$.

Definition 4.1.4 Then the asymptotic rate of convergence is defined to be

$$R_{\infty}(S) \equiv \lim_{n \to \infty} R(S^n) = -\log \rho(S).$$ (4.23)
**Theorem 4.1.3** If \( \|S^n\| < 1 \) for \( n > 0 \), then

\[
R(S^n) \leq R_\infty(S).
\] (4.24)

Since \( L \) is a linear operator, we can write the defect equation for (4.1) above as

\[
Le^{(n)} = r^{(n)},
\] (4.25)

where \( r^{(n)} = f - Lu^{(n)} \) is the residual after the \( n \)th iteration. We can also relate \( r^{(n+1)} \) to \( r^{(n)} \) by using equation (4.10) to get

\[
r^{(n+1)} = \hat{S}r^{(n)}
\] (4.26)

where \( \hat{S} = NM^{-1} \) is the iteration matrix for the residual. Since the residual and the error are related by the defect equation (4.25), then as the residual, \( r^{(n)} \), goes to the zero vector the error, \( e^{(n)} \), will also go to zero as long as \( L \) is non-singular. If \( L \) is singular then the error, \( e^{(n)} \), will tend to a vector in the null space of \( L \). This can and does happen for pure Neumann boundary conditions, and any solution obtained is only unique up to a constant. In this case, special care, usually through normalization, needs to be taken to ensure that a solution is obtained.

### 4.2 Gauss–Seidel Relaxation

In this section we define several relaxation methods which we have considered for inclusion in the black box multigrid methods. We will also discuss there appropriateness for inclusion in both the vector and parallel black box multigrid solvers.

We need to define some notation concerning the grid point equations. The grid that was used to generate the system of equations is the one that was defined in
section 1.3, and is the rectangular grid,

\[
G^h = \left\{(x_i, y_j): \begin{array}{l}
x_i = a_x + \sum_{k=1}^{i} h_{x_k}, \quad i = 1, \ldots, n_x, \\
y_j = a_y + \sum_{k=1}^{j} h_{y_k}, \quad j = 1, \ldots, n_y
\end{array} \right\}, \quad (4.27)
\]

where \(G^h = \Omega^h\). The grid point equation on grid \(G\), suppressing the superscript \(h\), is then

\[
S u_{i,j-1} + W u_{i-1,j} + C u_{i,j} + E u_{i+1,j} + N u_{i,j+1} = F_{i,j} \quad (4.28)
\]

where the subscript \((i, j)\) on the coefficients has been suppressed. Likewise, the 9-point stencil is given by

\[
\begin{align*}
SW u_{i-1,j-1} &+ S u_{i,j-1} + SE u_{i+1,j-1} \\
W u_{i-1,j} &+ C u_{i,j} + E u_{i+1,j} \\
NW u_{i-1,j+1} &+ N u_{i,j+1} + NE u_{i+1,j+1} = F_{i,j}
\end{align*} \quad (4.29)
\]

4.2.1 Point Gauss–Seidel Iteration  We are using a multi-color point Gauss–Seidel relaxation method. If the stencil for the grid operator \(L^h\) is a 5-point stencil, then a red/black version is used, and if the stencil is a 9-point stencil, then a 4-color version is used.

The point Gauss–Seidel relaxation method without multi-coloring for the 5-point stencil is given by

\[
S u_{i,j-1}^{(n+1)} + W u_{i-1,j}^{(n+1)} + C u_{i,j}^{(n+1)} + E u_{i+1,j}^{(n)} + N u_{i,j+1}^{(n)} = F_{i,j} \quad (4.30)
\]

and for the 9-point stencil is given by

\[
\begin{align*}
SW u_{i-1,j-1}^{(n+1)} &+ S u_{i,j-1}^{(n+1)} + SE u_{i+1,j-1}^{(n+1)} \\
W u_{i-1,j}^{(n+1)} &+ C u_{i,j}^{(n+1)} + E u_{i+1,j}^{(n)} \\
NW u_{i-1,j+1}^{(n)} &+ N u_{i,j+1}^{(n)} + NE u_{i+1,j+1}^{(n)} = F_{i,j}
\end{align*} \quad (4.31)
\]
This method does not vectorize or parallelize if only one color is employed regardless of the ordering of the equations. Vectorization fails because there will always be a vector feedback dependency (e.g. \( u(i, j) \) depends on \( u(i-1, j) \) for \( i = 1, 2, \cdots \)). It is possible to write down single color orderings that appear to allow vector and/or parallel processing, but one finds upon closer examination that they are equivalent to multi-color orderings. For our purposes, when we say one color ordering we mean that the equations have a lexicographic ordering. Parallelization is not possible for lexicographic ordering because of the data dependencies that exist by sweeping through the equations.

The multi-color point Gauss-Seidel relaxation method for the 5-point stencil has two colors: red and black. The coloring can be defined in terms of the grid point indices,

\[
\text{Red : } \quad i + j \text{ even } \quad (4.32) \\
\text{Black : } \quad i + j \text{ odd}.
\]

This method proceeds in two half steps, one for each color of points being used. The grid point equation for the red grid points (first half step) is

\[
S u_{i,j}^{(n)} + W u_{i-1,j}^{(n)} + C u_{i,j}^{(n+\frac{1}{2})} + E u_{i+1,j}^{(n)} + N u_{i,j+1}^{(n)} = F_{i,j},
\]

where \( i + j \) is even, and for the black grid points (second half step)

\[
S u_{i,j}^{(n+\frac{1}{2})} + W u_{i-1,j}^{(n+\frac{1}{2})} + C u_{i,j}^{(n+1)} + E u_{i+1,j}^{(n+\frac{1}{2})} + N u_{i,j+1}^{(n+\frac{1}{2})} = F_{i,j},
\]

where \( i + j \) is odd.

The multi-color point Gauss-Seidel relaxation method for the 9-point stencil has four colors: red, black, green, and yellow. The coloring can be defined in terms of
the grid point indices,

Red: \( i \) odd, \( j \) odd

Black: \( i \) even, \( j \) odd

Green: \( i \) odd, \( j \) even

Yellow: \( i \) even, \( j \) even .

This method proceeds in four quarter steps, one for each color of points being used.

The grid point equation for the red grid points (first quarter step) is

\[
\begin{align*}
SW \ u_{i-1,j-1}^{(n)} & \quad + S \ u_{i,j-1}^{(n)} & \quad + SE \ u_{i+1,j-1}^{(n)} \\
W \ u_{i-1,j}^{(n)} & \quad + C \ u_{i,j}^{(n+\frac{1}{4})} & \quad + E \ u_{i+1,j}^{(n)} \\
NW \ u_{i-1,j+1}^{(n)} & \quad + N \ u_{i,j+1}^{(n)} & \quad + NE \ u_{i+1,j+1}^{(n)} = F_{i,j},
\end{align*}
\]

where \( i \) is odd and \( j \) is odd. The grid point equation for the black grid points (second quarter step) is

\[
\begin{align*}
SW \ u_{i-1,j-1}^{(n)} & \quad + S \ u_{i,j-1}^{(n)} & \quad + SE \ u_{i+1,j-1}^{(n)} \\
W \ u_{i-1,j}^{(n+\frac{1}{4})} & \quad + C \ u_{i,j}^{(n+\frac{1}{4})} & \quad + E \ u_{i+1,j}^{(n+\frac{1}{4})} \\
NW \ u_{i-1,j+1}^{(n)} & \quad + N \ u_{i,j+1}^{(n)} & \quad + NE \ u_{i+1,j+1}^{(n)} = F_{i,j},
\end{align*}
\]

where \( i \) is even and \( j \) is odd. The grid point equation for the green grid points (third quarter step) is

\[
\begin{align*}
SW \ u_{i-1,j-1}^{(n+\frac{1}{4})} & \quad + S \ u_{i,j-1}^{(n+\frac{1}{4})} & \quad + SE \ u_{i+1,j-1}^{(n+\frac{1}{4})} \\
W \ u_{i-1,j}^{(n+\frac{1}{2})} & \quad + C \ u_{i,j}^{(n+\frac{1}{2})} & \quad + E \ u_{i+1,j}^{(n+\frac{1}{2})} \\
NW \ u_{i-1,j+1}^{(n+\frac{1}{4})} & \quad + N \ u_{i,j+1}^{(n+\frac{1}{4})} & \quad + NE \ u_{i+1,j+1}^{(n+\frac{1}{4})} = F_{i,j},
\end{align*}
\]

where \( i \) is odd and \( j \) is even. And finally, the grid point equation for the yellow grid points (fourth quarter step) is

\[
\begin{align*}
SW \ u_{i-1,j-1}^{(n+\frac{1}{4})} & \quad + S \ u_{i,j-1}^{(n+\frac{1}{4})} & \quad + SE \ u_{i+1,j-1}^{(n+\frac{1}{4})} \\
W \ u_{i-1,j}^{(n+\frac{1}{2})} & \quad + C \ u_{i,j}^{(n+\frac{1}{2})} & \quad + E \ u_{i+1,j}^{(n+\frac{1}{2})} \\
NW \ u_{i-1,j+1}^{(n+\frac{1}{4})} & \quad + N \ u_{i,j+1}^{(n+\frac{1}{4})} & \quad + NE \ u_{i+1,j+1}^{(n+\frac{1}{4})} = F_{i,j},
\end{align*}
\]
where \( i \) is even and \( j \) is even.

The addition of multi-coloring for the ordering of the sweeps through the grid equations leads to highly vectorizable and parallelizable solvers. Vectorization is obtained because sweeping through either red or black equations no longer contains vector feedback dependencies. Parallelization is obtained because the red and black points are now decoupled and all the equations of a single color can be computed independently of the other color. In other words, there are no longer any data dependencies among equations of the same color. However, the computations are performed not in one parallel operation, but in a number of parallel operations equal to the number of independent colors. As will be seen in the next chapter, in addition to the gains in vector and parallel performance, we also obtain, generally, better convergence and smoothing properties with multiple color ordering.

4.2.2 Line Gauss–Seidel Iteration by Lines in \( \mathbf{X} \) The line Gauss–Seidel relaxation by lines in the \( \mathbf{x} \)-direction is used in our code with red/black zebra coloring. This relaxation method is good when there are strong connections (locally) on the grid in the \( \mathbf{x} \)-direction. This method requires the use of a tridiagonal solver to solve for each line of unknowns.

We present the details for only the 9-point stencil because the 5-point stencil is just a special case where the \( \text{NW}, \text{NE}, \text{SW}, \) and \( \text{SE} \) grid operator coefficients are zero.

The line Gauss–Seidel relaxation by \( \mathbf{x} \)-lines method without multi-coloring for the 9-point stencil is given by

\[
\begin{align*}
\text{SW} \ u_{i-1,j-1}^{(n+1)} + \text{S} \ u_{i,j-1}^{(n+1)} + \text{SE} \ u_{i+1,j-1}^{(n+1)} \\
\text{W} \ u_{i-1,j}^{(n+1)} + \text{C} \ u_{i,j}^{(n+1)} + \text{E} \ u_{i+1,j}^{(n+1)} \quad \text{(4.40)} \\
\text{NW} \ u_{i-1,j+1}^{(n)} + \text{N} \ u_{i,j+1}^{(n)} + \text{NE} \ u_{i+1,j+1}^{(n)} = F_{i,j}.
\end{align*}
\]
The method is vectorizable and parallelizable only in the multiple line solves. By this we mean that we can loop over all the lines to obtain vectorization or we can solve all the lines simultaneously for parallelization. The parallel line solves are done on the CM-5 using odd-even cyclic reduction.

When the red/black zebra coloring is used, the colors are defined to be

\[
\begin{align*}
\text{Red} : & \quad j \text{ odd} \\
\text{Black} : & \quad j \text{ even}
\end{align*}
\] (4.41)

The line relaxation is done in two half steps, first for the red points, and second for the black points. The line relaxation for the 9-point stencil is

\[
\begin{align*}
SW \ u_{i-1,j-1}^{(n)} & + S \ u_{i,j-1}^{(n)} & + SE \ u_{i+1,j-1}^{(n)} \\
W \ u_{i-1,j}^{(n+\frac{1}{2})} & + C \ u_{i,j}^{(n+\frac{1}{2})} & + E \ u_{i+1,j}^{(n+\frac{1}{2})} \\
NW \ u_{i-1,j+1}^{(n)} & + N \ u_{i,j+1}^{(n)} & + NE \ u_{i+1,j+1}^{(n)} = F_{i,j},
\end{align*}
\] (4.42)

where \( j \) is odd for the first half step, and for the second half step,

\[
\begin{align*}
SW \ u_{i-1,j-1}^{(n+\frac{1}{2})} & + S \ u_{i,j-1}^{(n+\frac{1}{2})} & + SE \ u_{i+1,j-1}^{(n+\frac{1}{2})} \\
W \ u_{i-1,j}^{(n+1)} & + C \ u_{i,j}^{(n+1)} & + E \ u_{i+1,j}^{(n+1)} \\
NW \ u_{i-1,j+1}^{(n+\frac{1}{2})} & + N \ u_{i,j+1}^{(n+\frac{1}{2})} & + NE \ u_{i+1,j+1}^{(n+\frac{1}{2})} = F_{i,j},
\end{align*}
\] (4.43)

where \( j \) is even.

The zebra coloring method allows both vectorization and parallelization at not only for the lines solves, as before, but also by decoupling the red and black lines. For parallelization this decoupling means that all the lines of one color can be solved simultaneously. As will be seen, the convergence factors for zebra coloring are usually better than those for lexicographic ordering.

4.2.3 Line Gauss–Seidel Iteration by Lines in Y  The line Gauss–Seidel relaxation by lines in the \( y \)-direction is used in our code with red/black zebra
coloring. The method requires the use of a tridiagonal solver to solve for each line of unknowns. The line relaxation method is good when there are strong connections (locally) on the grid in the y-direction.

We will present the details for only the 9-point stencil, since the 5-point stencil is just a special case with the NW, NE, SW, and SE coefficients are zero.

The line Gauss–Seidel relaxation by y-lines method without multi-coloring for the 9-point stencil is given by

\[
\begin{align*}
SW u_{i-1,j-1}^{(n+1)} + S u_{i,j-1}^{(n+1)} + & SE u_{i+1,j-1}^{(n)} \\
W u_{i-1,j}^{(n+1)} + & C u_{i,j}^{(n+1)} + E u_{i+1,j}^{(n)} \\
NW u_{i-1,j+1}^{(n+1)} + N u_{i,j+1}^{(n+1)} + & NE u_{i+1,j+1}^{(n)} = F_{i,j} .
\end{align*}
\]

When the red/black zebra coloring is used, the colors are defined to be

\[
\begin{align*}
\text{Red} : & \quad i \text{ odd} \\
\text{Black} : & \quad i \text{ even} .
\end{align*}
\]

The line relaxation is done in two half steps, first for the red points and second for the black points. The first half step for the line relaxation for the 9-point stencil is

\[
\begin{align*}
SW u_{i-1,j-1}^{(n)} + S u_{i,j-1}^{(n+\frac{1}{2})} + & SE u_{i+1,j-1}^{(n)} \\
W u_{i-1,j}^{(n)} + & C u_{i,j}^{(n+\frac{1}{2})} + E u_{i+1,j}^{(n)} \\
NW u_{i-1,j+1}^{(n)} + N u_{i,j+1}^{(n+\frac{1}{2})} + & NE u_{i+1,j+1}^{(n)} = F_{i,j} ,
\end{align*}
\]

where \( i \) is odd and for the second half step, it is given by

\[
\begin{align*}
SW u_{i-1,j-1}^{(n+\frac{1}{2})} + S u_{i,j-1}^{(n+1)} + & SE u_{i+1,j-1}^{(n+\frac{1}{2})} \\
W u_{i-1,j}^{(n+\frac{1}{2})} + & C u_{i,j}^{(n+1)} + E u_{i+1,j}^{(n+\frac{1}{2})} \\
NW u_{i-1,j+1}^{(n+1)} + N u_{i,j+1}^{(n+1)} + & NE u_{i+1,j+1}^{(n+\frac{1}{2})} = F_{i,j} ,
\end{align*}
\]

where \( i \) is even.

The same comments about vectorization and parallelization that were made about x-line Gauss–Seidel apply to y-line Gauss–Seidel.
4.2.4 Alternating Line Gauss–Seidel Iteration  The alternating line Gauss–Seidel method performs zebra colored line Gauss–Seidel by lines in the $x$–direction followed by zebra colored line Gauss–Seidel by lines in the $y$–direction. For the details see the two previous sections 4.2.2 and 4.2.3 above.

4.3 Incomplete Line LU Iteration

The incomplete line LU iteration (ILLU) is also referred to in the literature as incomplete block LU iteration. We consider ILLU by lines in the $x$–direction. The method presented is for the 9–point stencil with the grid points ordered lexicographically. There are two parts to using this smoother, the first being the factorization and the second the actual iterations. In this section we change the notation we have been using for our problem from $Lu = f$ to $Au = f$. This is because we want to use the symbol $L$ to represent the lower triangular part of a matrix.

The ILLU factorization assumes that the matrix $A$ of the system to be solved is in tridiagonal block matrix form,

$$A = \begin{pmatrix}
B_1 & U_1 \\
L_2 & B_2 & U_2 \\
& \ddots & \ddots & \ddots \\
& & \ddots & \ddots & U_{n_y-1} \\
& & & L_{n_y} & B_{n_y}
\end{pmatrix}$$

(4.48)

where $L_j$, $B_j$, and $U_j$ are $n_x \times n_x$ tridiagonal matrices. Then there exists a matrix $D$, derived below, such that

$$A = (L + D)D^{-1}(D + U) ,$$

(4.49)
where

$$L = \begin{pmatrix}
0 & & & \\
L_2 & \ddots & & \\
& \ddots & \ddots & \\
& & \ddots & L_{ny} \\
& & & 0
\end{pmatrix},$$

$$U = \begin{pmatrix}
0 & U_1 & & \\
& \ddots & \ddots & \\
& & \ddots & U_{ny} \\
& & & 0
\end{pmatrix},$$

(4.50)

$$D = \begin{pmatrix}
D_1 & & & \\
& D_2 & & \\
& & \ddots & \\
& & & D_{ny}
\end{pmatrix},$$

and $D$ is block diagonal consisting of $n_x \times n_x$ tridiagonal blocks $D_j$. The factorization of $A$ in equation (4.49) is called the line LU factorization of $A$. The blocks $L$, $D$, and $U$ correspond to horizontal grid lines in the $x$–direction on the computational grid $G$. The equation (4.49) can also be written as

$$A = L + D + U + LD^{-1}U$$

(4.51)

and the last term is the block–diagonal matrix,

$$LD^{-1}U = \begin{pmatrix}
0 & & & \\
L_2D_1^{-1}U_1 & & & \\
& \ddots & \ddots & \\
& & \ddots & L_{ny}D_{ny-1}^{-1}U_{ny-1}
\end{pmatrix}$$

(4.52)
From the last two equations a recursion formula to compute \( D \) can be obtained

\[
\begin{align*}
D_1 &= B_1, \\
D_j &= B_j - L_j D_{j-1}^{-1} U_j, & j = 2, 3, \ldots, n_y
\end{align*}
\] (4.53)

assuming that \( D_{j-1}^{-1} \) exists. This shows that the splitting in equation (4.49) holds when \( D \) is computed by equation (4.53).

The problem with this splitting is that the \( D_j \)'s are full. There are many variations for the incomplete iteration that have been proposed in the literature. For more on the description and theory of ILLU iteration see [4], [3], [2], [76], [34], [44], [84], [85].

The variation that we are using is obtained by replacing the term \( L_j D_{j-1}^{-1} U_j \) in equation (4.53) by its tridiagonal part, to get

\[
\begin{align*}
D_1 &= B_1, \\
D_j &= B_j - \text{tridiag} \left( L_j D_{j-1}^{-1} U_j \right)
\end{align*}
\] (4.54)

where \( j = 2, 3, \ldots, n_y \). The ILLU factorization is then defined as

\[ A = M - N \] (4.55)

where \( M = (L + D)D^{-1}(D + U) \), and \( D \) is computed using equation (4.54).

The iteration for the system of equations

\[ Au = f \] (4.56)

is given by the splitting in equation (4.55). The iteration is then given by

\[ Mu^{(n+1)} = Nu^{(n)} + f \] (4.57)

or as

\[ u^{(n+1)} = Su^{(n)} + M^{-1} f , \] (4.58)
where \( S = M^{-1}N \) is called the iteration matrix. This iteration then becomes

\[
r = f - Au^{(n)}
\]

\[
(L + D)D^{-1}(D + U)u^{(n+1)} = r
\]

\[
u^{(n+1)} \leftarrow u^{(n)} + u^{(n+1)}
\]

for computing purposes. The center equation in (4.59) above is solved in the following way:

solve \((L + D)u^{(n+1)} = r\)

\[
r = Du^{(n+1)}
\]

and the first equation in (4.60) above can be solved by

\[
D_1u_1^{(n+1)} = r_1,
\]

\[
D_ju_j^{(n+1)} = r_j - L_{j-1}u_j^{(n)}, \quad j = 2, 3, \ldots, n_y
\]

where \(U_j\) and \(r_j\) are \(n_x\) dimensional vectors corresponding to block \(j\). The last equation in (4.60) above is computed similarly to the first equation in (4.61).

For completeness, we could have looked at ILLU by lines in the \(y\)-direction, or even alternating ILLU. This has not been found to be necessary because the smoothing properties of the ILLU by lines in either direction are so good that we can get away with only using the ILLU by lines in the \(x\)-direction. However, this is only true in two dimensions, and this smoother is not robust in three dimensions.

We still need to comment about the vector and parallel aspects for ILLU. The ILLU method does not easily lend itself to vectorization or parallelization, but we have
been able to get some reasonable performance on the Cray Y-MP. The vector method
is the same as the one used by De Zeeuw [24], but the implementation is different.
However, this method does not parallelize. Many people are working on parallel ILLU
methods, but so far we are not aware of any that are efficient enough on the CM-5 to
compete with zebra alternating line Gauss-Seidel relaxation as a multigrid smoother.
Here efficiency is in the sense of convergence factor per unit of execution time.

As a final note, it was found that De Zeeuw’s MGD9V obtains its robustness
from the use of the ILLU smoother and not from his new grid transfer operators,
although the do contribute to the robustness of the method. The ILLU smoother
was replaced with an alternating red/black line Gauss-Seidel smoother in MGD9V and
experiments showed it to perform only marginally better than Dendy’s nonsymmetric
black box multigrid method. Likewise, the ILLU smoother was placed in Dendy’s
nonsymmetric black box multigrid method, and experiments showed that it performed
about the same as MGD9V.
CHAPTER 5

FOURIER MODE ANALYSIS OF SMOOTHERS

5.1 Introduction

To understand which relaxation schemes make good smoothers for our Black Box Multigrid code, we can use Fourier mode analysis, which is also known as local mode analysis or local Fourier analysis.

We will use Fourier mode analysis to help guide us in finding robust and efficient relaxation methods for use as the multigrid smoother. We want to find methods that will reduce the high frequency error components for a range of test problems that include anisotropic and convection dominated operators.

Results of local mode analysis have been reported in the literature. However, most of the reports have been for only a few selected problems and smoothing methods. Since the literature lacks adequate coverage of smoothing analysis for our range of test problems we have presented many of the results from our own smoothing analysis investigation.

The smoother plays an important part of the multigrid process by reducing the high frequency error components. The coarse grid correction complements this process by eliminating the low frequency error components. It is hoped that our choice of coarse grid and intergrid operators will meet the requirement, for coarse grid correction, that the range of prolongation contains low frequency components; see 5.3. When it does, the smoothing factor will give a reasonable approximation of the multigrid convergence
factor for definite elliptic problems. However, the smoothing factors do not generally predict the exact performance of the two-level algorithm, since the intergrid operators are neglected, as well as the differences between the fine and coarse grid operators.

A two-level analysis can often give more information than smoothing analysis; both are performed by using Fourier mode analysis. Two-level analysis attempts to approximates the spectral radius of \( (I - I_H^h (L^H)^{-1} I_H^h L^h) S^h \) for the two-level algorithm, while smoothing analysis computes the spectral radius of \( S^h \), the convergence factor of high frequencies for the multigrid smoother. We have used Galerkin coarse grid approximation, which can produce a different coarse grid operator on each grid level, causing the two-level analysis to be valid only for those levels. For constant coefficient zero row-sum problems, the collapsing method intergrid operators become bi-linear interpolation and full weighting restriction, in which case the two-level analysis is straightforward [78]. Variable coefficient problems are handled by performing the analysis for the extreme cases with frozen coefficients and using a continuity argument. For highly variable or discontinuous coefficient problems, it is not clear how to perform two-level analysis, especially when Galerkin coarse grid approximation is being used. Because of the above mentioned difficulties for performing a two-level analysis we have chosen to use local mode analysis to analyze only the smoothing factors.

5.2 Motivation

When working on parallel and vector computers, it can often pay to consider methods that one would not usually think of using on a sequential computer. Likewise, there are tried and true algorithms that work well on a sequential computer but that do not vectorize or parallelize to any great extent.

We have used the CM–5, which is a massively parallel SIMD computer. It can also be used in a MIMD mode, but we will not be using it in this way. The CM–5
uses a small vector length of sixteen (until late 1995 it was eight), which even though it is short, favors algorithms which vectorize. We have also used the Cray Y-MP, which is a vector computer with a vector length of 64.

For a five point stencil, the relaxation scheme on the CM-5 one might consider using is a Gauss–Seidel point relaxation implemented by using multiple colors. For instance, one might use red/black Gauss–Seidel. This scheme takes two sweeps over the data, one for the red points and the other for the black points. If one implements Jacobi relaxation, one finds that it looks exactly like a Gauss–Seidel relaxation on a sequential computer (this is because of the way the synchronous SIMD operations work on the CM computers). It requires only one sweep across the data, and hence one can do two iterations of Jacobi for the price of one red/black Gauss–Seidel relaxation. The Jacobi method on the CM-5 does not need the extra storage space or index switching that is needed on a sequential computer. However, a nine point stencil requires a four color Gauss–Seidel relaxation scheme, which takes four sweeps through the data, per iteration, as much work as four sweeps of Jacobi relaxation.

It is known that the Jacobi relaxation is not very good, but we can use a damping factor to make it better. In this case, we may find that the Jacobi relaxation becomes more competitive with the Gauss–Seidel relaxation on parallel computers.

So, when considering smoothers, it is important to take into account the amount of work that must be done in parallel and to remember to consider those methods which may seem to be outdated.

In order for the Fourier mode analysis to be technically valid, the operator \( L \) must be constant coefficient with periodic boundary conditions. However, it can still provide useful bounds for other types of boundary conditions. If the problem has variable coefficients, then it should be analyzed for a fixed set of coefficients that are sampled across the domain, and it should include the extreme cases in amplitude of
the coefficients. The behavior of the problem should then be bounded by the extreme bounds found in the analysis.

Fourier (local) mode analysis has been the topic of debate for many years as to how robust and rigorous it can be, and under what circumstances it can be successfully applied. However, it has proven itself to be quite useful and is used by most practitioners. Achi Brandt and several of his students have proposed that it can be rigorous in its construction and use, if proper care is taken; see Brandt’s paper on *Rigorous Local Mode Analysis* [17].

### 5.3 Overview of Smoothing Analysis

A good introduction to Fourier smoothing analysis can be found in the 1982 paper by Stüben and Trottenberg [78] and in the book by P. Wesseling [85]. The presentation in this section uses similar notation and is patterned after Wesseling’s book chapter 7.

Let the grid $G$ be defined by

$$
G = \left\{ (x_i, y_j) \in \mathbb{R} : \begin{array}{l}
x_i = i \cdot h_x, \quad i = 1, \ldots, n_x, \quad h_x = \frac{1}{n_x - 1} \\
y_j = j \cdot h_y, \quad j = 1, \ldots, n_y, \quad h_y = \frac{1}{n_y - 1}
\end{array} \right\}
$$

(5.1)

The continuous problem is discretized into a system of algebraic equations

$$
Lu = f
$$

(5.2)

where $L$ is represented by the stencil

$$
[L]_{i,j} = \begin{bmatrix}
NW & N & NE \\
W & C & E \\
SW & S & SE
\end{bmatrix}_{i,j}.
$$

(5.3)

For the most part, smoothing methods are basic iterative methods. That is, they are
splittings (usually a regular splitting) of the form
\[ L = M - N. \] (5.4)

The details for these methods are given in chapter 4. For the basic type of iterative methods the error amplification matrix is given by
\[ S = I - M^{-1}L \] (5.5)
without damping, and with damping it becomes
\[ S = I - \omega M^{-1}L \] (5.6)

If the continuous problem has constant coefficients and periodic boundary conditions, then the stencils of \([L]\), \([M]\), and \([N]\) are independent of the grid point \((i,j)\). We will assume that \(S\) has a complete set of eigenfunctions (local modes). The error before, \(e^{(0)}\), and after, \(e^{(1)}\) smoothing is given by
\[ e^{(1)} = S e^{(0)}, \] (5.7)
which gives us the relation
\[ S \Phi(\theta) = \lambda(\theta)\Phi(\theta) \] (5.8)
where \(\lambda(\theta)\) is an eigenvalue associated with the eigenfunction \(\Phi(\theta)\). The eigenfunctions of \(S\) are
\[ \Phi_{i,j}(\theta) = e^{i(\theta_x + j\theta_y)} \quad \theta \in \Theta \] (5.9)
where \(\iota = \sqrt{-1}, \theta = (\theta_x, \theta_y)\), and \(\Theta\) is defined as
\[ \Theta = \left\{ (\theta_x, \theta_y) : \begin{array}{l}
\theta_x = \frac{2\pi k_x}{n_x}, \quad k_x = -\frac{n_x}{2}, -\frac{n_x}{2} - 1, \ldots, \frac{n_x}{2} \\
\theta_y = \frac{2\pi k_y}{n_y}, \quad k_y = -\frac{n_y}{2}, -\frac{n_y}{2} - 1, \ldots, \frac{n_y}{2}
\end{array} \right\} \] (5.10)
If \( n_x \) and \( n_y \) are assumed to be even, then the corresponding eigenvalues of \( S \) are

\[
\lambda(\theta) = \frac{\sum_\kappa [N]_\kappa e^{(k_x\theta_x + k_y\theta_y)}}{\sum_\kappa [M]_\kappa e^{(k_x\theta_x + k_y\theta_y)}} \quad \theta \in \Theta,
\]

where \( \kappa = (k_x, k_y) \) is a vector.

The eigenvalue, \( \lambda(\theta) \), is called the amplification factor of the Fourier mode \( \Phi(\theta) \). If under-relaxation or over-relaxation is used then

\[
\lambda(\theta) \equiv \omega \lambda(\theta) + (1 - \omega)
\]

where \( \omega \) is the relaxation parameter, and the \( \lambda(\theta) \) on the right hand side is the eigenvalue from the undamped amplification matrix.

The Fourier representation of a periodic grid function \( u_{i,j} \) is

\[
u_{i,j} = \sum_{\theta \in \Theta} c_\theta \Phi_{i,j}(\theta),
\]

and the error is

\[
e^\alpha = \sum_{\theta \in \Theta} c_\theta^\alpha \Phi(\theta), \quad \alpha = 0, 1
\]

which then gives

\[
c_\theta^1 = \lambda(\theta) c_\theta^0.
\]

Next, we define the sets of rough and smooth frequencies, that is, the high and low frequencies respectively relative to the grid \( G \). We have assumed that the ratio between the fine and coarse grid spacings is two. The smooth frequencies are defined as

\[
\Theta_s = \Theta \cap \left( -\frac{\pi}{2}, \frac{\pi}{2} \right)^2
\]

and the rough frequencies are defined as

\[
\Theta_r = \Theta \setminus \Theta_s
\]
where the \( \setminus \) means "the set minus". The Fourier smoothing factor is now defined to be

\[
\mu = \max_{\theta \in \Theta_r} \{|\lambda(\theta)|\}.
\]  

(5.18)

Now we need to consider the effect of the boundary conditions, in particular, the Dirichlet boundary condition. For problems with Dirichlet boundary conditions, we know that the error at the boundary is always zero, and hence, we can ignore the wave numbers where \( \theta_x = 0 \) and/or \( \theta_y = 0 \). Then the set of rough wave numbers, in the Dirichlet boundary condition case, is defined to be

\[
\Theta^D_r = \Theta_r \cap \{ \theta \in \Theta : \theta_x \neq 0 \text{ and/or } \theta_y \neq 0 \},
\]  

(5.19)

and then the corresponding smoothing factor, \( \mu^D \), is given by

\[
\mu^D = \max_{\theta \in \Theta^D_r} \{|\lambda(\theta)|\}.
\]  

(5.20)

The above definitions of the smoothing factor are grid size dependent because they depend on \( n_x \) and \( n_y \) (the number of grid points in the x and y directions respectively). The definitions can be changed to be grid-independent if we change the definition of the discrete set \( \Theta \) to be

\[
\Theta = \{ \theta : \theta_x \in [-\pi, \pi], \theta_y \in [-\pi, \pi] \}.
\]  

(5.21)

This grid-independent definition is much harder to compute numerically, and when the boundary conditions have a big influence on the solution, the results are not very realistic. The grid dependent definitions given above are best when the choice of \( n_x \) and \( n_y \) are in the same range as those expected when using the multigrid method. We would always like to have \( \mu < 1 \) uniformly in \( n_x \) and \( n_y \), at least when the boundary conditions do not have a strong influence. If that is not the case, then the coarse grid correction part of the multigrid method must be very good to overcome the smoother’s
divergent influence. We can also numerically investigate the behavior of $\mu$ as $n_x$ and $n_y \to 0$ to see what its asymptotic behavior is like.

Up to this point we have not addressed a very important class of relaxation methods for the analysis, and those which use multi-coloring schemes. For these methods we must modify the above definitions.

For the case of multi-color relaxation, the $\Phi_{i,j}(\theta)$ are not eigenfunctions any more, but certain subspaces spanned by their linear combinations are invariant; see [78] and [85]. For the basic iterative methods that we are considering, we have the following four invariant subspaces

$$\begin{align*}
\theta^1 &= \Theta \cap [-\frac{\pi}{2}, \frac{\pi}{2})^2 \\
\theta^2 &= \left(\theta^1_x - \text{sign}(\theta^1_x)\pi, \theta^1_y - \text{sign}(\theta^1_y)\pi\right) \\
\theta^3 &= \theta^1 - (0, \text{sign}(\theta^1_y)\pi) \\
\theta^4 &= \theta^1 - (\text{sign}(\theta^1_x)\pi, 0),
\end{align*}$$

and $\Phi(\theta)$ can now be written as

$$\Phi(\theta) = \left(\Phi(\theta^1), \Phi(\theta^2), \Phi(\theta^3), \Phi(\theta^4)\right).$$

The Fourier representation of an arbitrary periodic function is

$$u_{i,j} = \sum_{\theta \in \Theta} c_\theta^T \Phi_{i,j}(\theta)$$

where $c_\theta$ is a four dimensional vector.

We now have that the error before smoothing is

$$e^0(\theta) = c_\theta^T \Phi(\theta),$$

and then after smoothing it is

$$e^1(\theta) = (S(\theta)c_\theta)^T \Phi(\theta),$$
where $S(\theta)$ is a $4 \times 4$ matrix, which is called the amplification matrix, and $c_0$ is a vector of dimension 4. If under-relaxation or over-relaxation is used, then

$$S(\theta) \equiv \omega S(\theta) + (1 - \omega)I$$  \hspace{1cm} (5.27)

where $\omega$ is the relaxation damping parameter and $I$ is the identity matrix.

The amplification matrix $S(\theta)$ can be found by:

1. Write the equations for the error for one step (color) of one complete iteration of the smoother.

2. Combine the error equations for that step into one expression.

3. Evaluate the combined expression for each of the invariant sub-spaces.

4. Write the equation that expresses the $n^{th}$-step Fourier coefficient $c_n^0$ in terms of the initial Fourier coefficient $c_0^0$, which are related by the step amplification matrix; $c_n^0 = S_n(\theta)c_0^0$.

5. Do the above for each step of one complete smoothing iteration.

6. Multiply all the step amplification matrices together to get the amplification matrix for the smoother, which will express the Fourier coefficients $c_n^0$ in terms of $c_0^0$; $c_1^0 = S(\theta)c_0^0$.

This algorithm will be illustrated for the smoothing analysis of the point Gauss–Seidel method in section 5.5.

For multi-color relaxations, the definition of the Fourier smoothing factor, $\mu$, has to be modified in the following ways. The rough Fourier modes are now given by

$$\Phi(\theta^2), \quad \Phi(\theta^3), \quad \Phi(\theta^4)$$  \hspace{1cm} (5.28)

and the smooth Fourier modes are now represented by

$$\Phi(\theta^1), \quad \theta^1_x \neq -\frac{\pi}{2}, \quad \text{or} \quad \theta^1_y \neq -\frac{\pi}{2}.$$  \hspace{1cm} (5.29)
All of these values must be added to \( \Theta_r \), in order for the Fourier mode analysis to have any meaning for several cases that can arise. We can now define a projection operator, \( Q(\theta) \), for \( \Phi(\theta) \) onto the Fourier modes, and it is represented by a diagonal \( 4 \times 4 \) matrix

\[
Q(\theta) = \begin{pmatrix}
\delta(\theta) & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]  
(5.30)

where

\[
\delta(\theta) = \begin{cases}
1, & \theta_x = -\frac{\pi}{2} \text{ or } \theta_y = -\frac{\pi}{2} \\
0, & \text{otherwise}
\end{cases}
\]  
(5.31)

Define \( \Theta_1 = \theta^1 \); then the new definition for the Fourier smoothing factor is given by

\[
\mu = \max_{\theta \in \Theta_1} \{ \rho [Q(\theta)S(\theta)] \},
\]  
(5.32)

where \( \rho \) denotes the spectral radius.

For problems with Dirichlet boundary conditions, we can do as before, but we must now exclude the wave numbers of the form \((\theta^k_x, 0)\) and \((0, \theta^k_y)\) for \(k = 1, 3, 4\). We will never have the case that \( \theta^2_x = 0 \) or \( \theta^2_y = 0 \), implying that the corresponding terms for \( c_{\theta} \) are now zero. We can represent this situation by the addition of another projection matrix, \( P(\theta) \), to \( Q(\theta)S(\theta) \), to give \( P(\theta)Q(\theta)S(\theta) \), and define it to be

\[
P(\theta) = \begin{pmatrix}
p_1(\theta) & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]  
(5.33)

\[p_1(\theta) = \begin{cases}
1, & \theta_x = -\frac{\pi}{2} \text{ or } \theta_y = -\frac{\pi}{2} \\
0, & \text{otherwise}
\end{cases}
\]
where

\[ p_1(\theta) = \begin{cases} 
0 & \theta_x = 0 \text{ and/or } \theta_y = 0 \\
1 & \text{otherwise}
\end{cases} \]

\[ p_3(\theta) = \begin{cases} 
0 & \theta_x = 0 \\
1 & \text{otherwise}
\end{cases} \]  \hspace{1cm} (5.34)

\[ p_4(\theta) = \begin{cases} 
0 & \theta_y = 0 \\
1 & \text{otherwise}
\end{cases} \]

It can also be seen that \( p_3(\theta) = 0 \) implies that \( \theta_x^3 = 0 \), and that \( p_4(\theta) = 0 \) implies that \( \theta_y^4 = 0 \). The definition of the smoothing factor now becomes

\[ \mu^D = \max_{\theta \in \Theta_r} \{ \rho \left[ P(\theta)Q(\theta)S(\theta) \right] \}, \]

\hspace{1cm} (5.35)

where \( \rho \) denotes the spectral radius.

The smoothing factors for Dirichlet boundary conditions are better than those for other boundary conditions because they exclude points in \( \Theta_r \) and \( \Theta_z \). This fact means that if the maximum occurs on these excluded points, then the smoothing factor for Dirichlet boundary conditions will be smaller.

### 5.4 2D Model Problems

In the subsequent sections and chapters, various model problems are examined. The model problems are used for comparing the performance of the black box multigrid components (smoothers and grid transfer operators) on a finite set of model problems that represent various characteristics of more realistic problems.

The domain \( \Omega \) is the unit square for the two dimensional model problems:

1. \(-\Delta u = f\)
2. \(-u_{xx} - \varepsilon u_{yy} = f\)
3. \(-\varepsilon u_{xx} - u_{yy} = f\)
4. \(-\varepsilon \Delta u - u_x = f\)
5. \(-\varepsilon \Delta u + u_x = f\)
6. \(-\varepsilon \Delta u - u_y = f\)
7. \(-\varepsilon \Delta u + u_y = f\)
8. \(-\varepsilon \Delta u - u_x - u_y = f\)
9. \(-\varepsilon \Delta u + u_x + u_y = f\)
10. \(-\varepsilon \Delta u - u_x + u_y = f\)
11. \(-\varepsilon \Delta u + u_x - u_y = f\)

where \(\Delta u = u_{xx} + u_{yy}\), \(\varepsilon = 10^{-p}\) for \(p = 0, 1, \ldots, 5\).

The model problems will be discretized using central differences for the second order terms and upstream differencing for the first order terms.

### 5.5 Local Mode Analysis for Point Gauss–Seidel Relaxation

Local mode analysis results are presented for lexicographical and red/black ordering for point Gauss–Seidel relaxations.

Point Gauss–Seidel relaxation with lexicographic ordering gives the splitting

\[
[M] = \begin{bmatrix}
0 \\
W & C & 0 \\
S
\end{bmatrix}, \quad [N] = \begin{bmatrix}
-N \\
0 & 0 & -E \\
0
\end{bmatrix}.
\]

The amplification factor \(\lambda(\theta)\) is given by

\[
\lambda(\theta) = \frac{|E e^{i\theta_x} + N e^{i\theta_y}|}{|W e^{-i\theta_x} + C + S e^{-i\theta_y}|}.
\]

The red/black point Gauss–Seidel relaxation local mode amplification matrix
is computed below. The stencil is assumed to be 5-point because a four color scheme would be needed for a 9-point stencil. The details for the computation of the amplification matrix will only be given for the 5-point red/black point Gauss-Seidel relaxation. The amplification matrices for all other multi-color Gauss-Seidel type smoothers can be computed in a similar manner.

One iteration of the red/black point Gauss-Seidel relaxation is performed in two half steps. The first half step is computed for the red points, and the second half step is computed on the black points. The red points can be identified by those points \((i, j)\) where \(i + j\) is even and the black points when \(i + j\) is odd.

Let the error before smoothing be \(e^0_{i,j}\); then the error after the first half step is

\[
\begin{align*}
\frac{1}{2} e^1_{i,j} &= \begin{cases} 
- \frac{S e^0_{i,j-1} + W e^0_{i-1,j} + E e^0_{i+1,j} + N e^0_{i,j+1}}{C}, & i+j \text{ even} \\
\frac{e^0_{i,j}}, & i+j \text{ odd}
\end{cases} \tag{5.38}
\end{align*}
\]

and after the second half step it is

\[
\begin{align*}
e^1_{i,j} &= \begin{cases} 
\frac{e^1_{i,j}}, & i+j \text{ even} \\
- \frac{S e^1_{i,j-1} + W e^1_{i-1,j} + E e^1_{i+1,j} + N e^1_{i,j+1}}{C}, & i+j \text{ odd}
\end{cases} \tag{5.39}
\end{align*}
\]

The Fourier representation of \(e^n_{i,j}\), \(n = 0, \frac{1}{2}, 1\) is given by

\[
e^n_{i,j} = \sum_{\theta \in \Theta_2} (c^n_\theta)^T \Phi_{i,j}(\theta) \tag{5.40}
\]

where \(c^n_\theta\) and \(\Phi_{i,j}(\theta)\) are vectors of dimension 4.

Examining the first half step in the relaxation process, let the initial error be

\[
e^0_{i,j}(\theta^k) = \Phi_{i,j}(\theta^k), \quad k = 1, 2, 3, 4 . \tag{5.41}
\]

Substitution into equation (5.38) gives

\[
\begin{align*}
\frac{1}{2} e^\frac{1}{2}_{i,j}(\theta^k) &= \begin{cases} 
\frac{- S \Phi_{i-1,j}(\theta^k) + W \Phi_{i,j-1}(\theta^k) + E \Phi_{i+1,j}(\theta^k) + N \Phi_{i,j+1}(\theta^k)}{C}, & i+j \text{ even} \\
\Phi_{i,j}(\theta^k), & i+j \text{ odd}
\end{cases}
\end{align*}
\]

103
Recall that $\Phi_{i,j}(\theta^k) = e^{(i\theta^k + j\theta^k)}$. After substitution for the red points $(i + j \text{ even})$ we get

$$
\frac{1}{e_{i,j}^k(\theta^k)} = - \frac{S \Phi_{i,j-1}(\theta^k) + W \Phi_{i-1,j}(\theta^k) + E \Phi_{i+1,j}(\theta^k) + N \Phi_{i,j+1}(\theta^k)}{C}
= - \frac{S e^{-\theta^k} + W e^{-\theta^k} + E e^{\theta^k} + N e^{\theta^k}}{C} \Phi_{i,j}(\theta^k)
= \hat{\alpha}(\theta^k) \Phi_{i,j}(\theta^k)
$$

The error for the first half step of the relaxation is now written as

$$
\frac{1}{e_{i,j}^k(\theta^k)} = \begin{cases} 
\hat{\alpha}(\theta^k) \Phi_{i,j}(\theta^k), & i + j \text{ even} \\
\Phi_{i,j}(\theta^k), & i + j \text{ odd}
\end{cases}
$$

where

$$
\hat{\alpha}(\theta^k) = - \frac{S e^{-\theta^k} + W e^{-\theta^k} + E e^{\theta^k} + N e^{\theta^k}}{C}. \quad (5.43)
$$

Before proceeding any further, we will need the following theorem.

**Theorem 5.5.1** For any $\theta \in [-\pi, \pi)$ and $k \in \mathbb{Z}^+$,

$$
e^{i k(\theta - \pi)} = \begin{cases} 
e^{i k \theta}, & k \text{ even} \\
-ne^{i k \theta}, & k \text{ odd}
\end{cases} \quad (5.44)
$$

**Proof 5.5.1**

$$
e^{i k(\theta - \pi)} = \cos[k(\theta - \pi)] + i \sin[k(\theta - \pi)] \quad (5.45)
$$

If $k$ is even, then

$$
e^{i k(\theta - \pi)} = [\cos(k\theta) \cdot 1 + \sin(k\theta) \cdot 0] + i [\sin(k\theta) \cdot 1 - \cos(k\theta) \cdot 0]
= \cos(k\theta) + i \sin(k\theta)
= e^{i k \theta}
$$
If $k$ is odd, then

\[
e^{ik(\theta-\pi)} = [\cos(k\theta) \cdot (-1) + \sin(k\theta) \cdot 0] + \iota \left[ \sin(k\theta) \cdot (-1) - \cos(k\theta) \cdot 0 \right]
\]

\[
e^{ik\theta} = -\cos(k\theta) - \iota \sin(k\theta)
\]

\[
e^{ik\theta} = -e^{k\theta}
\]

We want to combine the expressions for the red and black points of equation (5.42) into a single expression. We already have $\Phi_{i,j}(\theta^1)$, but we need to add one or more other sub-spaces to to create a linear combination that will yield a single expression. If we take $\theta - \pi$ for all the angles whose indices are involved in the coloring pattern designation, $i$ and $j$ in this case, we get the additional sub-space that we need for the linear combination. The single expression linear combination is

\[
e_{i,j}^k(\theta^k) = A e^{i[\theta_i^k + j(\theta_j^k)]} + B e^{i[i(\theta_i^k-\pi) + j(\theta_j^k-\pi)]}.
\]

(5.46)

Using theorem 5.5.1 we can find the values of $A$ and $B$.

\[
A + B = \hat{\alpha} \quad \text{for } i + j \text{ even}
\]

\[
A - B = 1 \quad \text{for } i + j \text{ odd},
\]

which gives

\[
A = \frac{1}{2}(1 + \hat{\alpha})
\]

\[
B = \frac{1}{2}(\hat{\alpha} - 1),
\]

and therefore

\[
e_{i,j}^k(\theta^k) = \frac{1}{2} \left(1 + \hat{\alpha} \right) e^{i[\theta_i^k + j(\theta_j^k)]} + \frac{1}{2} \left(\hat{\alpha} - 1 \right) e^{i[i(\theta_i^k-\pi) + j(\theta_j^k-\pi)]}.
\]

(5.47)

Let $\alpha$ be defined by

\[
\alpha(\theta) = \frac{S e^{-\theta v} + W e^{-i\theta z} + E e^{i\theta z} + N e^{\theta v}}{C},
\]

(5.48)
and define $\beta$ to be

$$
\beta(\theta) = \frac{S e^{-i\theta_0} - W e^{-i\theta_z} - E e^{i\theta_z} + N e^{i\theta_0}}{C}.
$$

(5.49)

We now evaluate the first half step error for each of the four invariant subspaces.

$$
e_{i,j}^{\frac{1}{2}}(\theta^1) = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^1)\right) e^{i[\theta_0^1 + j\theta_0^1]} + \frac{1}{2} \left(\hat{\alpha}(\theta^1) - 1\right) e^{i[(\theta_0^1 - \pi) + j(\theta_0^1 - \pi)]}
= \frac{1}{2} \left(1 + \alpha\right) \Phi_{i,j}(\theta^1) + \frac{1}{2} \left(\alpha - 1\right) \Phi_{i,j}(\theta^2),
$$

(5.50)

$$
e_{i,j}^{\frac{1}{2}}(\theta^2) = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^2)\right) e^{i[\theta_0^2 + j\theta_0^2]} + \frac{1}{2} \left(\hat{\alpha}(\theta^2) - 1\right) e^{i[(\theta_0^2 - \pi) + j(\theta_0^2 - \pi)]}
= \frac{1}{2} \left(1 - \alpha\right) \Phi_{i,j}(\theta^2) - \frac{1}{2} \left(\alpha + 1\right) \Phi_{i,j}(\theta^1),
$$

(5.51)

$$
e_{i,j}^{\frac{1}{2}}(\theta^3) = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^3)\right) e^{i[\theta_0^3 + j\theta_0^3]} + \frac{1}{2} \left(\hat{\alpha}(\theta^3) - 1\right) e^{i[(\theta_0^3 - \pi) + j(\theta_0^3 - \pi)]}
= \frac{1}{2} \left(1 + \beta\right) \Phi_{i,j}(\theta^3) + \frac{1}{2} \left(\beta - 1\right) \Phi_{i,j}(\theta^4),
$$

(5.52)

$$
e_{i,j}^{\frac{1}{2}}(\theta^4) = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^4)\right) e^{i[\theta_0^4 + j\theta_0^4]} + \frac{1}{2} \left(\hat{\alpha}(\theta^4) - 1\right) e^{i[(\theta_0^4 - \pi) + j(\theta_0^4 - \pi)]}
= \frac{1}{2} \left(1 - \beta\right) \Phi_{i,j}(\theta^4) - \frac{1}{2} \left(\beta + 1\right) \Phi_{i,j}(\theta^3).
$$

(5.53)

We can now express the Fourier coefficient, $e_{\theta}^{\frac{1}{2}}$, for the first half step in terms of the initial Fourier coefficient $e_0$. This task is done by matching the corresponding coefficients for each of the four invariant subspaces to get

$$
e_{i,j}^{\frac{1}{2}} = \frac{1}{2} \begin{bmatrix}
\alpha + 1 & -\alpha & -1 & 0 & 0 \\
\alpha - 1 & 1 - \alpha & 0 & 0 \\
0 & 0 & \beta + 1 & -\beta & -1 \\
0 & 0 & \beta - 1 & 1 - \beta
\end{bmatrix} e_0, \quad \theta \in \Theta_z.
$$

(5.54)
We now proceed in a similar way for the second half step (black points) of the relaxation process. The error after the second half step was given in equation (5.39), and it can be written, as in the first half step, as

\[
\epsilon_{i,j}^1 = \begin{cases} 
\Phi_{i,j}(\theta^k), & i + j \text{ even} \\
\hat{\alpha}(\theta^k) \Phi_{i,j}(\theta^k), & i + j \text{ odd}
\end{cases}
\] (5.55)

where \( \hat{\alpha} \) is defined in equation (5.43).

The equation can be written as a linear combination, as was the first half step, to get

\[
\epsilon_{i,j}^1(\theta^k) = A e^{i[\theta^k_x + j\theta^k_y]} + B e^{i[\theta^k_x - \pi + j(\theta^k_y - \pi)]},
\] (5.56)

Using theorem 5.5.1 we can find the values of \( A \) and \( B \),

\[
A + B = 1, \quad \text{for } i + j \text{ even}
\]
\[
A - B = \hat{\alpha}, \quad \text{for } i + j \text{ odd},
\]

which are solved to yield

\[
A = \frac{1}{2} \left( 1 + \hat{\alpha} \right) \]
\[
B = \frac{1}{2} \left( 1 - \hat{\alpha} \right),
\]

hence,

\[
\epsilon_{i,j}^1(\theta^k) = \frac{1}{2} \left( 1 + \hat{\alpha} \right) e^{i[\theta^k_x + j\theta^k_y]} + \frac{1}{2} \left( 1 - \hat{\alpha} \right) e^{i[\theta^k_x - \pi + j(\theta^k_y - \pi)]}.
\] (5.57)

Since the \( \hat{\alpha}(\theta^k) \) for \( k = 1, 2, 3, 4 \) are the same as for the first half step, we have the same \( \alpha \) and \( \beta \). We proceed to the evaluation of the error for the four invariant sub-spaces to get

\[
\epsilon_{i,j}^1(\theta^1) = \frac{1}{2} \left( 1 + \hat{\alpha}(\theta^1) \right) e^{i[\theta^1_x + j\theta^1_y]} + \frac{1}{2} \left( 1 - \hat{\alpha}(\theta^1) \right) e^{i[\theta^1_x - \pi + j(\theta^1_y - \pi)]}
\]
\[
= \frac{1}{2} (1 + \alpha) \Phi_{i,j}(\theta^1) + \frac{1}{2} (1 - \alpha) \Phi_{i,j}(\theta^2),
\] (5.58)
\[
e^{\frac{i}{2}(\theta^2)} = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^2)\right) e^{i[\theta^2_1 + j\theta^2_2]} + \frac{1}{2} \left(1 - \hat{\alpha}(\theta^2)\right) e^{i[i(\theta^2_1 - \pi) + j(\theta^2_2 - \pi)]}
\]
\[
= \frac{1}{2} (1 - \alpha) \Phi_{i,j}(\theta^2) + \frac{1}{2} (1 + \alpha) \Phi_{i,j}(\theta^2),
\] (5.59)

\[
e^{\frac{i}{7}(\theta^3)} = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^3)\right) e^{i[\theta^3_1 + j\theta^3_2]} + \frac{1}{2} \left(1 - \hat{\alpha}(\theta^3)\right) e^{i[i(\theta^3_1 - \pi) + j(\theta^3_2 - \pi)]}
\]
\[
= \frac{1}{2} (1 + \beta) \Phi_{i,j}(\theta^3) + \frac{1}{2} (1 - \beta) \Phi_{i,j}(\theta^4),
\] (5.60)

\[
e^{\frac{i}{4}(\theta^4)} = \frac{1}{2} \left(1 + \hat{\alpha}(\theta^4)\right) e^{i[\theta^4_1 + j\theta^4_2]} + \frac{1}{2} \left(1 - \hat{\alpha}(\theta^4)\right) e^{i[i(\theta^4_1 - \pi) + j(\theta^4_2 - \pi)]}
\]
\[
= \frac{1}{2} (1 - \beta) \Phi_{i,j}(\theta^4) - \frac{1}{2} (1 + \beta) \Phi_{i,j}(\theta^3).
\] (5.61)

The Fourier coefficient \(c^1_\theta\) in terms of \(c^\frac{1}{2}_\theta\) is

\[
c^1_\theta = \frac{1}{2}
\begin{bmatrix}
1 + \alpha & 1 + \alpha & 0 & 0 \\
1 - \alpha & 1 - \alpha & 0 & 0 \\
0 & 0 & 1 + \beta & 1 + \beta \\
0 & 0 & 1 - \beta & 1 - \beta
\end{bmatrix}
\begin{bmatrix}
c^\frac{1}{2}_\theta\
\end{bmatrix}, \quad \theta \in \Theta_z.
\] (5.62)

Finally, we can express \(c^1_\theta\) in terms of \(c^0_\theta\) by substitution of \(c^\frac{1}{2}_\theta\) into equation (5.62) to get the red/black point Gauss-Seidel amplification matrix \(S(\theta)\) that gives the relation

\[
c^1_\theta = \frac{1}{2}
\begin{bmatrix}
\alpha(1 + \alpha) & -\alpha(1 + \alpha) & 0 & 0 \\
\alpha(1 - \alpha) & \alpha(\alpha - 1) & 0 & 0 \\
0 & 0 & \beta(1 + \beta) & -\beta(1 + \beta) \\
0 & 0 & \beta(1 - \beta) & \beta(\beta - 1)
\end{bmatrix}
\begin{bmatrix}
c^0_\theta\
\end{bmatrix}, \quad \theta \in \Theta_z.
\] (5.63)

The eigenvalues of \(Q(\theta)S(\theta)\) are

\[
\lambda_1(\theta) = 0
\]
\[
\lambda_2(\theta) = 0
\]
Table 5.1. Smoothing factor $\mu$ for point Gauss–Seidel relaxation in lexicographical (pGS-lex) and red/black (r/b-pGS) ordering for the indicated anisotropic diffusion problems (see section 5.4); where $\varepsilon = 10^{-p}$ and $(D)$ indicates Dirichlet boundary conditions.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p$</th>
<th>pGS-lex</th>
<th>r/b-pGS</th>
<th>r/b-pGS (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.83220</td>
<td>.82645</td>
<td>.82619</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99797</td>
<td>.99800</td>
<td>.99770</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99999</td>
<td>.99975</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>.83220</td>
<td>.82645</td>
<td>.82619</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99797</td>
<td>.99800</td>
<td>.99770</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99999</td>
<td>.99975</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>.83220</td>
<td>.82645</td>
<td>.82619</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99797</td>
<td>.99800</td>
<td>.99770</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99999</td>
<td>.99975</td>
</tr>
</tbody>
</table>

\[
\lambda_3(\theta) = \frac{1}{2} (\delta(\theta) a - b)
\]
\[
\lambda_4(\theta) = \beta,
\]

and for Dirichlet boundary condition case, the eigenvalues of $P(\theta)Q(\theta)S(\theta)$ are

\[
\lambda_1(\theta) = 0
\]
\[
\lambda_2(\theta) = 0
\]
\[
\lambda_3(\theta) = \frac{1}{2} (p_1(\theta) \delta(\theta) a - b)
\]
\[
\lambda_4(\theta) = \frac{1}{2} \beta (p_3(\theta) - p_4(\theta) + \beta (p_3(\theta) + p_4(\theta)) )
\]

The results of local mode analysis for the model problems from section 5.4 are shown in table 5.1 and table 5.2. The smoothing factors were computed numerically with the grid spacing $h_x = h_y = 1$ and the angles $\theta_x$ and $\theta_y$ were sampled at one degree increments.

Table 5.1 shows the results of the smoothing analysis for pure diffusion type problems. The point Gauss–Seidel relaxations are good smoothers for Poisson’s equation, but not for anisotropic problems. The table also shows that red/black ordering is better than lexicographic ordering.
Table 5.2. Smoothing factor $\mu$ for point Gauss–Seidel relaxation in lexicographical (pGS-lex) and red/black (r/b-pGS) ordering for the indicated convection–diffusion problems (see section 5.4); where $\varepsilon = 10^{-p}$ and (D) indicates Dirichlet boundary conditions.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>pGS-lex</th>
<th>r/b-pGS</th>
<th>r/b-pGS (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>.60176</td>
<td>.36000</td>
<td>.35990</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.87313</td>
<td>.73469</td>
<td>.73463</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99839</td>
<td>.99602</td>
<td>.99602</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99996</td>
<td>.99996</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>.45834</td>
<td>.36000</td>
<td>.35990</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.44608</td>
<td>.73469</td>
<td>.73463</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44099</td>
<td>.99602</td>
<td>.99602</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44099</td>
<td>.99996</td>
<td>.99996</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>.60176</td>
<td>.36000</td>
<td>.35990</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.87313</td>
<td>.73469</td>
<td>.73463</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99839</td>
<td>.99602</td>
<td>.99602</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99996</td>
<td>.99996</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>.45834</td>
<td>.36000</td>
<td>.35990</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.44608</td>
<td>.73469</td>
<td>.73463</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44099</td>
<td>.99602</td>
<td>.99602</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44099</td>
<td>.99996</td>
<td>.99996</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>.66281</td>
<td>.28125</td>
<td>.28125</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.91533</td>
<td>.69441</td>
<td>.69441</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99898</td>
<td>.99594</td>
<td>.99594</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99999</td>
<td>.99988</td>
<td>.99988</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>.32950</td>
<td>.28125</td>
<td>.28125</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.08202</td>
<td>.69441</td>
<td>.69441</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.00098</td>
<td>.99594</td>
<td>.99594</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.8E-6</td>
<td>.99988</td>
<td>.99988</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>.56192</td>
<td>.28125</td>
<td>.28125</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.84486</td>
<td>.69441</td>
<td>.69441</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99797</td>
<td>.99594</td>
<td>.99594</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99988</td>
<td>.99988</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>.56192</td>
<td>.28125</td>
<td>.28125</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.84486</td>
<td>.69441</td>
<td>.69441</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99797</td>
<td>.99594</td>
<td>.99594</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99988</td>
<td>.99988</td>
</tr>
</tbody>
</table>
Table 5.2 shows the results of the smoothing analysis for convection–diffusion problems. The red/black ordering for point Gauss–Seidel has, in general, better smoothing properties than those for lexicographic ordering except for problems 5, 7, and 9. The reason that lexicographic ordering is better for those problems is because the order in which the unknowns are updated is in the same direction as the convection characteristics. The smoothing factors approach one as the convection terms become more dominant, which implies that point Gauss–Seidel is not a robust smoother for these types of problems.

5.6 Local Mode Analysis for Line Gauss–Seidel Relaxation

Local mode analysis results are presented for lexicographic and zebra (reb/black) ordering for x- and y-line Gauss–Seidel relaxations.

X-line Gauss–Seidel relaxation with lexicographic ordering gives the splitting

\[
[M] = \begin{bmatrix}
  0 \\
  W & C & E \\
  S
\end{bmatrix}, \quad [N] = \begin{bmatrix}
  -N \\
  0 & 0 & 0 \\
  0
\end{bmatrix}.
\] (5.64)

The amplification factor \( \lambda(\theta) \) is given by

\[
\lambda(\theta) = \frac{|N e^{i\theta_v}|}{|W e^{-i\theta_x} + E e^{i\theta_x} + C + S e^{-i\theta_y}|}.
\] (5.65)

Zebra x-line Gauss–Seidel relaxation has the amplification matrix

\[
S(\theta) = \frac{1}{2} \begin{pmatrix}
  a & 0 & -a & 0 \\
  0 & c & 0 & -c \\
  b & 0 & -b & 0 \\
  0 & d & 0 & -d
\end{pmatrix},
\] (5.66)
where \( a = \alpha(1 + \alpha) \), \( b = \alpha(1 - \alpha) \), \( c = \beta(1 + \beta) \), and \( d = \beta(1 - \beta) \) and

\[
\begin{align*}
\alpha &= -\frac{S \, e^{-i\theta_y} + N \, e^{i\theta_y}}{W \, e^{-i\theta_x} + C + E \, e^{i\theta_x}}, \\
\beta &= -\frac{-S \, e^{-i\theta_y} + N \, e^{i\theta_y}}{-W \, e^{-i\theta_x} + C + E \, e^{i\theta_x}}.
\end{align*}
\]

The eigenvalues of \( S(\theta) \) are

\[
\begin{align*}
\lambda_1(\theta) &= 0 \\
\lambda_2(\theta) &= 0 \\
\lambda_3(\theta) &= \frac{1}{2} (\delta(\theta) a - b) \\
\lambda_4(\theta) &= \frac{1}{2} (c - d),
\end{align*}
\]

and for Dirichlet boundary conditions we have

\[
\begin{align*}
\lambda_1(\theta) &= 0 \\
\lambda_2(\theta) &= 0 \\
\lambda_3(\theta) &= \frac{1}{2} (p_1(\theta) \delta(\theta) a - p_3(\theta) b) \\
\lambda_4(\theta) &= \frac{1}{2} (c - p_4(\theta) d).
\end{align*}
\]

Y-line Gauss–Seidel relaxation with lexicographic ordering gives the splitting

\[
[M] = \begin{bmatrix}
N \\
W & C & 0 \\
S
\end{bmatrix}, \quad [N] = \begin{bmatrix}
0 & 0 & -E \\
0 & 0 & 0
\end{bmatrix}.
\]

The amplification factor \( \lambda(\theta) \) is given by

\[
\lambda(\theta) = \frac{|E \, e^{i\theta_x}|}{|W \, e^{-i\theta_x} + C + S \, e^{-i\theta_y} + N \, e^{i\theta_y}|}.
\]
Zebra y–line Gauss–Seidel relaxation has the amplification matrix

\[
S(\theta) = \frac{1}{2} \begin{pmatrix}
  a & 0 & 0 & a \\
  0 & c & c & 0 \\
  0 & d & d & 0 \\
  b & 0 & 0 & b \\
\end{pmatrix}
\]

(5.71)

where \(a = \alpha(\alpha + 1), \ b = \alpha(\alpha - 1), \ c = \beta(\beta + 1), \ and \ d = \beta(\beta - 1)\) and

\[
\alpha = -\frac{W e^{-\theta_x} + E e^{\theta_x}}{S e^{-\theta_y} + C + N e^{\theta_y}}
\]

(5.72)

\[
\beta = -\frac{-W e^{-\theta_x} + E e^{\theta_x}}{-S e^{-\theta_y} + C + N e^{\theta_y}}
\]

(5.73)

The eigenvalues of \(S(\theta)\) are

\[
\lambda_1(\theta) = 0
\]

\[
\lambda_2(\theta) = 0
\]

\[
\lambda_3(\theta) = \frac{1}{2} (\delta(\theta) a + b)
\]

\[
\lambda_4(\theta) = \frac{1}{2} (c + d),
\]

and for Dirichlet boundary conditions we have

\[
\lambda_1(\theta) = 0
\]

\[
\lambda_2(\theta) = 0
\]

\[
\lambda_3(\theta) = \frac{1}{2} (p_1(\theta)\delta(\theta) + p_1(\theta)b)
\]

\[
\lambda_4(\theta) = \frac{1}{2} (c + p_3(\theta)d).
\]

The results of local mode analysis for the model problems from section 5.4 are shown in tables 5.3 and 5.4. The smoothing factors were computed numerically with the grid spacing \(h_x = h_y = 1\) and the angles \(\theta_x\) and \(\theta_y\) were sampled at 1 degree increments.
Table 5.3. Smoothing factor $\mu$ for $x$– and $y$–line Gauss-Seidel relaxation in lexicographical ($\text{xlIGS}$ and $\text{ylIGS}$ respectively) and zebra ($\text{ZxlIGS}$ and $\text{ZylIGS}$ respectively) ordering for the indicated anisotropic diffusion problems (see section 5.4); where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>$\text{xlIGS}$</th>
<th>$\text{ZxlIGS}$</th>
<th>$\text{ZxlIGS} (D)$</th>
<th>$\text{ylIGS}$</th>
<th>$\text{ZylIGS}$</th>
<th>$\text{ZylIGS} (D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.44412</td>
<td>.25000</td>
<td>.24992</td>
<td>.44412</td>
<td>.25000</td>
<td>.24992</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>.44412</td>
<td>.12500</td>
<td>.1250</td>
<td>.82644</td>
<td>.82645</td>
<td>.82619</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44412</td>
<td>.12500</td>
<td>.1250</td>
<td>.99800</td>
<td>.99800</td>
<td>.99770</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44412</td>
<td>.12500</td>
<td>.02891</td>
<td>.99994</td>
<td>.99998</td>
<td>.99962</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>.83092</td>
<td>.82645</td>
<td>.82619</td>
<td>.44412</td>
<td>.12500</td>
<td>.12500</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99797</td>
<td>.99800</td>
<td>.99770</td>
<td>.44412</td>
<td>.12500</td>
<td>.12500</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99998</td>
<td>.99968</td>
<td>.44412</td>
<td>.12500</td>
<td>.02891</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>.99797</td>
<td>.99800</td>
<td>.99770</td>
<td>.44412</td>
<td>.12500</td>
<td>.12500</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99998</td>
<td>.99998</td>
<td>.99968</td>
<td>.44412</td>
<td>.12500</td>
<td>.02891</td>
</tr>
</tbody>
</table>
Table 5.3 shows the smoothing factors for line Gauss–Seidel relaxation for anisotropic diffusion model problems. It is seen that line relaxation is only a good smoother if the lines are taken in the direction of the strong coupling of the diffusion coefficients. Again, it is seen that the zebra ordering of the lines gives a better smoothing factor than lexicographic ordering.

Table 5.4 shows the smoothing factors for the convection–diffusion model problems for line Gauss–Seidel relaxation. The smoothing factors for line relaxation are good when the convection term characteristics are in the same direction as the lines. The smoothing factor becomes better (smaller) the more the convection terms dominate if the characteristics are in the direction of the lines. If the characteristics are not in the direction of the lines, then the smoothing factor degenerates quickly approaching one, the more the convection terms dominate the diffusion term. We see again that for lexicographic ordering the smoothing factor is better when the characteristics have at least one component in the direction of the lexicographic ordering of the lines.

5.7 Local Mode Analysis for Alternating Line Gauss–Seidel and ILLU Iteration

Local mode analysis results are presented for lexicographic and zebra ordering for alternating line Gauss–Seidel relaxation (x–line Gauss–Seidel followed by y–line Gauss–Seidel) and incomplete line LU by lines in x.

The alternating line Gauss–Seidel relaxation with lexicographic ordering amplification factor $\lambda(\theta)$ is given by

$$
\lambda(\theta) = \lambda_{x\text{gs}}(\theta) \lambda_{y\text{gs}}(\theta)
$$

(5.74)

where $\lambda_{x\text{gs}}(\theta)$ and $\lambda_{y\text{gs}}(\theta)$ are the x– and y–line Gauss–Seidel amplification factors.
Table 5.4. Smoothing factor $\mu$ for $x$– and $y$–line Gauss-Seidel relaxation in lexicographical ($x_lGS$ and $y_lGS$ respectively) and zebra ($Zx_lGS$ and $Zy_lGS$ respectively) ordering for the indicated convection–diffusion problems (see section 5.4); where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p$</th>
<th>$x_lGS$</th>
<th>$Zx_lGS$</th>
<th>$Zx_lGS$ (D)</th>
<th>$y_lGS$</th>
<th>$Zy_lGS$</th>
<th>$Zy_lGS$ (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>.45040</td>
<td>.15385</td>
<td>.15380</td>
<td>.62917</td>
<td>.36000</td>
<td>.35990</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.48377</td>
<td>.22449</td>
<td>.22449</td>
<td>.91218</td>
<td>.73469</td>
<td>.73463</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44412</td>
<td>.12500</td>
<td>.05644</td>
<td>.99898</td>
<td>.99602</td>
<td>.99602</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44412</td>
<td>.12500</td>
<td>.00057</td>
<td>.99999</td>
<td>.99996</td>
<td>.99996</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>.62917</td>
<td>.36000</td>
<td>.35990</td>
<td>.45040</td>
<td>.15385</td>
<td>.15380</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.91218</td>
<td>.73469</td>
<td>.73463</td>
<td>.48377</td>
<td>.22449</td>
<td>.22449</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99898</td>
<td>.99602</td>
<td>.99602</td>
<td>.44412</td>
<td>.12500</td>
<td>.05644</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99999</td>
<td>.99996</td>
<td>.99996</td>
<td>.44412</td>
<td>.12500</td>
<td>.00057</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>.32950</td>
<td>.36000</td>
<td>.35990</td>
<td>.45040</td>
<td>.15385</td>
<td>.15380</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.32950</td>
<td>.73469</td>
<td>.73463</td>
<td>.48377</td>
<td>.22449</td>
<td>.22449</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.32950</td>
<td>.99602</td>
<td>.99602</td>
<td>.44412</td>
<td>.12500</td>
<td>.05644</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.32950</td>
<td>.99996</td>
<td>.99996</td>
<td>.44412</td>
<td>.12500</td>
<td>.00057</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>.63226</td>
<td>.24324</td>
<td>.24318</td>
<td>.63226</td>
<td>.24324</td>
<td>.24318</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.91344</td>
<td>.69444</td>
<td>.69409</td>
<td>.91344</td>
<td>.69444</td>
<td>.69409</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99898</td>
<td>.99601</td>
<td>.99541</td>
<td>.99898</td>
<td>.99601</td>
<td>.99541</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99999</td>
<td>.99996</td>
<td>.99996</td>
<td>.99999</td>
<td>.99996</td>
<td>.99935</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>.27929</td>
<td>.24324</td>
<td>.24318</td>
<td>.27929</td>
<td>.24324</td>
<td>.24318</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.06696</td>
<td>.69444</td>
<td>.69409</td>
<td>.06696</td>
<td>.69444</td>
<td>.69409</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.00080</td>
<td>.99602</td>
<td>.99541</td>
<td>.00080</td>
<td>.99601</td>
<td>.99541</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8.0E-6</td>
<td>.99996</td>
<td>.99935</td>
<td>8.0E-6</td>
<td>.99996</td>
<td>.99935</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>.27929</td>
<td>.24324</td>
<td>.24318</td>
<td>.63226</td>
<td>.24324</td>
<td>.24318</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.06696</td>
<td>.69444</td>
<td>.69409</td>
<td>.91344</td>
<td>.69444</td>
<td>.69409</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.00080</td>
<td>.99601</td>
<td>.99541</td>
<td>.99898</td>
<td>.99601</td>
<td>.99541</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8.0E-6</td>
<td>.99996</td>
<td>.99935</td>
<td>.99999</td>
<td>.99996</td>
<td>.99935</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>.63226</td>
<td>.24324</td>
<td>.24318</td>
<td>.27929</td>
<td>.24324</td>
<td>.24318</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.91344</td>
<td>.69444</td>
<td>.69409</td>
<td>.06696</td>
<td>.69444</td>
<td>.69409</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.99898</td>
<td>.99601</td>
<td>.99541</td>
<td>.00080</td>
<td>.99601</td>
<td>.99541</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.99999</td>
<td>.99996</td>
<td>.99935</td>
<td>8.0E-6</td>
<td>.99996</td>
<td>.99935</td>
</tr>
</tbody>
</table>

116
respectively. Thus,

$$\lambda(\theta) = \frac{|N e^{i\theta y}|}{|We^{-i\theta x} + E e^{i\theta x} + C + S e^{-i\theta y} + N e^{i\theta y}|} \left| \frac{|E e^{i\theta x}|}{|We^{-i\theta x} + E e^{i\theta x} + C + S e^{-i\theta y} + N e^{i\theta y}|} \right| \quad (5.75)$$

The zebra alternating line Gauss–Seidel relaxation amplification matrix $S(\theta)$ is given by

$$S(\theta) = S_{x\ell g s}(\theta) S_{y\ell g s}(\theta) \quad (5.76)$$

where $S_{x\ell g s}(\theta)$ and $S_{y\ell g s}(\theta)$ are the x– and y–line Gauss–Seidel amplification matrices respectively. We represent the matrix $S(\theta)$ by

$$S(\theta) = \frac{1}{4} \begin{pmatrix} a & b & -a & -b \\ c & d & -c & -d \\ e & f & -e & -f \\ g & h & -g & -h \end{pmatrix} \quad (5.77)$$

where

$$a = p_1(\theta)\delta(\theta)a_xa_y \quad b = p_1(\theta)\delta(\theta)d_xa_y$$
$$c = b_xc_y \quad d = c_xc_y$$
$$e = p_3(\theta)b_xd_y \quad f = p_3(\theta)c_xd_y$$
$$g = p_4(\theta)a_xb_y \quad h = p_4(\theta)d_xb_y ,$$

and the subscripts $x$ and $y$ indicate that the coefficients came from $S_{x\ell g s}(\theta)$ and $S_{y\ell g s}(\theta)$ respectively; see equations (5.66) and (5.71). The eigenvalues of $S(\theta)$ are

$$\lambda_1(\theta) = 0, \quad \lambda_2(\theta) = 0,$$

$$\lambda_{3,4}(\theta) = \frac{D \pm \sqrt{D^2 - 4[(a - e)(d - h) - (b - f)(c - g)]}}{8}$$

where $D = a + d - e - h$ is the diagonal of $S(\theta)$. For non-Dirichlet boundary conditions we set $p_1(\theta) = p_3(\theta) = p_4(\theta) = 1.$
The incomplete x-line LU iteration (ILLU) amplification factor is not hard to compute, but it is a little more complicated than the other relaxation methods. We need to compute $M$ and $N$ for the ILLU splitting; see section 4.3. Incomplete factorization methods have the property that the stencils for $M$ and $N$ are dependent upon their location on the grid, even when the stencil of $L$ is not. However, the stencils of $M$ and $N$ usually tend rapidly to constant stencils away from the boundaries. It is these constant stencils for $M$ and $N$ that will be used for the local mode analysis. It can be seen that the smoothing factor increases towards one as the block (x-line) size increases. For this reason, we will assume the worst case, $n_x = \infty$, for the computation of the local smoothing factors.

The component $D_j$ from equation (4.54) is computed without the $j$ subscript until the stencil for $D$ becomes stable. By stable, we mean that the values do not change, or that the change is taking place only in the digits after a specified decimal place. We used six decimal places in our computations.

When a stable $D$ has been computed, then the stencils for $M$ and $N$ can be constructed and the smoothing factor computed using equation (5.11). Due to the nature of these computations, it is not possible to write down a general formula for the amplification factor as was done for the other relaxation methods.

The results of local mode analysis for the model problems from section 5.4 are shown in table 5.5 and table 5.6. The smoothing factors were computed numerically with the grid spacing $h_x = h_y = 1$, and the angles $\theta_x$ and $\theta_y$ were sampled at 1 degree increments.

Table 5.5 shows the smoothing factors for alternating line Gauss–Seidel relaxation and incomplete line LU iteration for the anisotropic diffusion model problems. Lexicographic ordering for alternating line relaxation provides a fair smoothing factor, but zebra ordering provides much better smoothing factors. The smoothing factors for
Table 5.5. Smoothing factor $\mu$ for alternating line Gauss-Seidel relaxation in lexicographical (ALGS) and zebra (AZLGS) ordering, and incomplete line LU iteration by lines in $x$ (ILLU) for the indicated anisotropic diffusion problems (see section 5.4); where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p$</th>
<th>ALGS</th>
<th>AZLGS</th>
<th>AZLGS (D)</th>
<th>ILLU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>.14634</td>
<td>.02547</td>
<td>.02546</td>
<td>.05788</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>.36903</td>
<td>.10107</td>
<td>.10104</td>
<td>.13272</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44322</td>
<td>.12472</td>
<td>.12467</td>
<td>.19209</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44411</td>
<td>.12500</td>
<td>.02890</td>
<td>.19920</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>.36903</td>
<td>.10107</td>
<td>.10104</td>
<td>.10769</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44322</td>
<td>.12472</td>
<td>.12467</td>
<td>.16422</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44411</td>
<td>.12500</td>
<td>.02890</td>
<td>.14136</td>
</tr>
</tbody>
</table>
incomplete line LU iteration are good as well, but alternating zebra line relaxation is slightly better.

Table 5.6 shows the smoothing factors for the convection–diffusion model problems for alternating line Gauss–Seidel relaxation and incomplete line LU iteration. The smoothing factors for lexicographic ordering for the alternating line relaxation are good but get even better when the characteristics are in the same direction (lexicographic) in which the lines are solved. The alternating zebra line relaxation gives good smoothing factors when the characteristics are in the same direction as the lines; indeed these smoothing factors are better than those for lexicographic ordering. The zebra ordering gives fair smoothing factors when the characteristics are not aligned with the lines. The incomplete line LU iteration is done by lines in the x–direction and is nearly a direct solver when the convection terms are dominant. The smoothing factors are good for all of the model problems, and they are about equal to or much better than those for alternating line relaxation. They are especially superior when the convection term characteristics are not aligned with the grid lines.

5.8 Local Mode Analysis Conclusions

We have looked at local mode analysis for several iterative methods for use as a smoother in our black box multigrid method. The test problems can be classified in many ways, but we will break them down into four types and refer to them via their model problem number from section 5.4. The first type is the isotropic diffusion equation represented by model problem (1). The second type is the anisotropic diffusion equations represented by model problems (2) and (3). The third type is the convection–diffusion equations represented by model problems (4)–(11) with $\varepsilon = 1$. The fourth type is the convection dominated equations represented by model problems (4)–(11) with $\varepsilon \ll 1$. 
Table 5.6. Smoothing factor $\mu$ for alternating line Gauss–Seidel relaxation in lexicographical (ALGS) and zebra (AZLGS) ordering, and incomplete line LU iteration by lines in x (ILLU) for the indicated convection–diffusion problems (see section 5.4); where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ALGS</th>
<th>AZLGS</th>
<th>AZLGS (D)</th>
<th>ILLU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>.22269</td>
<td>.04706</td>
<td>.04704</td>
<td>.07977</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.40812</td>
<td>.14253</td>
<td>.14251</td>
<td>.16150</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44322</td>
<td>.12444</td>
<td>.05611</td>
<td>.19952</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44411</td>
<td>.12499</td>
<td>.00057</td>
<td>.20000</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>.14750</td>
<td>.04706</td>
<td>.04704</td>
<td>.07977</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.15423</td>
<td>.14253</td>
<td>.14251</td>
<td>.16150</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.14634</td>
<td>.12444</td>
<td>.05611</td>
<td>.19952</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.14634</td>
<td>.12499</td>
<td>.00057</td>
<td>.20000</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>.22269</td>
<td>.04706</td>
<td>.04704</td>
<td>.03759</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.40812</td>
<td>.14253</td>
<td>.14251</td>
<td>.00567</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44322</td>
<td>.12444</td>
<td>.05611</td>
<td>.00009</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44411</td>
<td>.12499</td>
<td>.00057</td>
<td>4.4E-9</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>.14750</td>
<td>.04706</td>
<td>.04704</td>
<td>.03759</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.15423</td>
<td>.14253</td>
<td>.14251</td>
<td>.00567</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.14634</td>
<td>.12444</td>
<td>.05611</td>
<td>.00009</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.14634</td>
<td>.12499</td>
<td>.00057</td>
<td>4.4E-9</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>.24619</td>
<td>.06349</td>
<td>.06346</td>
<td>.04940</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.39787</td>
<td>.31498</td>
<td>.31483</td>
<td>.01489</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.44358</td>
<td>.40579</td>
<td>.40559</td>
<td>.00019</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>.44412</td>
<td>.40684</td>
<td>.40665</td>
<td>1.9E-6</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>.06754</td>
<td>.06349</td>
<td>.06346</td>
<td>.04940</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.00441</td>
<td>.31498</td>
<td>.31483</td>
<td>.01489</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.3E-7</td>
<td>.40579</td>
<td>.40559</td>
<td>.00019</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.4E-11</td>
<td>.40684</td>
<td>.40665</td>
<td>1.9E-6</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>.15074</td>
<td>.06349</td>
<td>.06346</td>
<td>.04940</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.05636</td>
<td>.31498</td>
<td>.31483</td>
<td>.01489</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.00073</td>
<td>.40579</td>
<td>.40559</td>
<td>.00019</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.3E-6</td>
<td>.40684</td>
<td>.40665</td>
<td>1.9E-6</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>.15074</td>
<td>.06349</td>
<td>.06346</td>
<td>.04940</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.05636</td>
<td>.31498</td>
<td>.31483</td>
<td>.01489</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>.00073</td>
<td>.40579</td>
<td>.40559</td>
<td>.00019</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.3E-6</td>
<td>.40684</td>
<td>.40665</td>
<td>1.9E-6</td>
</tr>
</tbody>
</table>
We see that red/black point Gauss–Seidel relaxation is a good smoother for only the isotropic diffusion and convection–diffusion ($\varepsilon = 1$) equations, however, for variable coefficients it is a good smoother for only isotropic diffusion equations.

Zebra line Gauss–Seidel relaxation is a good smoother all four types of problems, provided that the anisotropies and convection characteristics are aligned with the proper grid directions.

The only two robust choices for smoothers are the alternating zebra line Gauss–Seidel relaxation and the ILLU methods. They both exhibit good smoothing factors for all the types of problems. However, ILLU is better for all types of convection–diffusion equations and just slightly worse for the two types of diffusion equations. The suitability of either choice for the smoother will depend on the efficiency of the implementation, and under these circumstances it would appear that alternating zebra line Gauss–Seidel relaxation has the advantage.

5.9 Other Iterative Methods Considered for Smoothers

If one looks at the local mode analysis, it will be noticed that lexicographic point Gauss–Seidel relaxation has a good smoothing property for convection problems when the sweep direction is aligned with that of the convection. This suggests another smoother, namely, 4-direction point Gauss–Seidel relaxation. The 4-direction point Gauss–Seidel method performs four sweeps over the grid starting each time from a different corner of the grid. The first two sweeps are the same as symmetric point Gauss–Seidel, with the dominant sweep direction being in $x$, and the third and fourth sweeps are again a symmetric point Gauss–Seidel relaxation, but with the dominant sweep direction in $y$ this time.

The sweeping strategy for symmetric point Gauss–Seidel relaxation is the
same as performing one iteration of lexicographic point Gauss-Seidel relaxation followed by another sweep, but the second sweep starts at the \((N_x, N_y)\) point of the grid with the index in the x-direction decreasing the quickest. To form the 4-direction point Gauss-Seidel method we perform one iteration of symmetric point Gauss-Seidel followed by another iteration of symmetric point Gauss-Seidel, but this time rotated 90 degrees so that the rolls of x and y are reversed.

The 4-direction point Gauss-Seidel method exhibits good smoothing properties for all the model problem except the anisotropic ones. The 4-direction method is partially vectorizable, but not parallelizable. Let us take a look at one of the four sweeping direction, namely lexicographic, to illustrate how one can obtain some vectorization. The non-vectorizing lexicographic sweep is computed as follows.

\[
\begin{align*}
\text{DO } j &= 1, N_y \\
\text{DO } i &= 1, N_x \\
    u_{i,j} &= (N_{i,j}u_{i,j+1} + W_{i,j}u_{i-1,j} + E_{i,j}u_{i+1,j} + S_{i,j}u_{i,j-1} + f_{i,j})/C_{i,j}
    \\
\text{END DO} \\
\text{END DO}
\end{align*}
\]

Vectorization is prevented by the reference to \(u_{i-1,j}\). We can minimize the impact of the vector dependency by re-organizing the calculation and creating a new temporary array of length \(N_x\). The new code with vectorization is computed in the following way.

\[
\begin{align*}
\text{DO } j &= 1, N_y \\
\text{DO } i &= 1, N_x \\
    \text{tmp}(i) &= N_{i,j}u_{i,j+1} + E_{i,j}u_{i+1,j} + S_{i,j}u_{i,j-1} + f_{i,j} \\
\text{END DO} \\
\text{DO } i &= 1, N_x
\end{align*}
\]
\[ u_{i,j} = \left( tmp(i) + W_{i,j}u_{i-1,j} \right)/C_{i,j} \]

END DO

END DO

The first loop over \( i \) performs vector operations and the second loop over \( i \) performs scalar operations. The algorithm is presented for a 5-point stencil, the only difference for a 9-point stencil is the addition of the other computational elements to the calculation in the vector (first) loop. On the Cray Y-MP, the second algorithm is roughly equivalent, timewise, to alternating zebra line Gauss–Seidel for small grids (\(< 32 \times 32\)) and is faster for larger grids. The reason that the 4-direction method outperforms the alternating line method, for larger grids, is because the line solves are sequential in nature.

Several experiments were run on the Cray Y-MP using the 4-direction point Gauss–Seidel relaxation for the smoother. The results were mixed, but generally favorable.

The performance for isotropic diffusion or linear convection characteristic problems was good, as expected from the Fourier mode analysis. Anisotropic problems also performed quite poorly, as expected from the analysis. For convection–diffusion problems with variable convection characteristics the results were dependent on the form of the characteristics and the choice of grid transfer operators. The 4-direction point Gauss–Seidel smoother worked best with the nonsymmetric collapsing method \((\sigma L/\sigma L)\) from section 3.5.1 for the grid transfer operators. However, for re-entrant flows we were still unable to obtain a method which would give any kind of reasonable convergence rate. The results from one of the numerical experiments are given in table 7.34 in section 7.6.
CHAPTER 6

VECTOR ALGORITHM CONSIDERATIONS IN TWO DIMENSIONS

The computers that we used fall into three categories: sequential, vector, and parallel. Each of these types of computers has its own characteristics that can affect the development and implementation of algorithms that are to execute on them.

Sequential computers come in a lot of varieties, but they all execute basically “one” instruction at a time. By “one” instruction we mean to lump all of the pipelining architectures with the classic sequential computer architecture. This lumping may not be entirely fair, but we believe that it is nearly impossible to find a computer today that does not use some form of pipelining. If one looks at the scalar processors on vector computers, one sees a great deal of pipelining. As far as the choice of algorithms is concerned, they can indeed be lumped together. However, it does pay to remember that pipelining of instructions is taking place, and therefore the implementation of the algorithm should take advantage of it when at all possible. For the most part, compilers today are very good at keeping the pipeline full, but they can still benefit from a careful implementation that aligns the instructions to avoid possible bubbles (null operations) in the pipeline, e.g., excessive branching. We have used a SUN Sparc workstation to represent the class of sequential computers, but a standard PC would have done just as well.

The vector computers are those with vector units that can process a vector
(array) of data with one instruction with only the indices varying in a fixed relationship to each other. The Cray Y-MP, which we have used, is a prime example of such a computer. The CM-5, which we have also used, has vector units, but the vector units are not very fast when compared to the Y-MP’s, and they have a vector length of 16 (changed from 8 to 16 in late 1995), which is very short when compared to the Y-MP’s vector length of 64. Vector computers also have scalar processors for handling the non-vectorizable instructions, and these processors can be considered to be the same as those of the sequential computers.

The Cray Y-MP can have several processors that can be run in a parallel mode configuration. However, we have chosen to use only one processor on the Cray Y-MP so that we can concentrate on the vectorization issues.

There are several types of parallel computers and parallel computing models. The type of parallel computers that we considered are single instruction multiple data (SIMD) computers; the CM-5 is such a computer, but it might be more meaningful to classify it as a single program multiple data (SPMD) computer for our purposes. The SPMD programming model is one of the most widely used parallel programming models for almost all parallel computers. The CM-5 can be run under two different execution models: the data parallel model, which we used, and the message passing model, which we choose not to address in this thesis.

Probably the one issue that has the greatest effect on algorithm performance, regardless of the type of computer, is that of memory references. This issue can manifest itself in many ways at both the hardware and software level of the computer architecture. For sequential and vector computers it usually revolves around a memory cache, but memory bank architecture can also play a role. It should be noted that the Cray Y-MP does not have a memory cache. On parallel computers the memory cache and banks are usually subordinate to the data communications network.
Each of these three types of computers has its own influences on the choice and implementation of the various components of multigrid algorithms. However, we will restrict our choices for the vector computers in such a way as to avoid degrading the code’s execution on a sequential computer in any meaningful way. If, however, a particular choice would cause only minor degradation on the sequential computer but greatly improve its performance on vector computers, then it should be allowed.

For the above reasons, and because it is not too interesting, we will not examine our multigrid codes on any sequential computers. The performance of closely related black box multigrid codes for various problems has already appeared in the literature [27] [8]. However, for timing comparisons only we will include some data for a Sparc5 workstation.

6.1 Cray Hardware Overview

The Cray Y-mp is our baseline vector computer for the design of the black box multigrid vector algorithm. The hardware model that we will present for the software design is equally valid for the Cray Y-mp, X-mp, M90, and C90 computers because we are concerned only with the single processor vector algorithm.

The Cray Y-mp computers can have a number of CPUs (central processing units), typically 4, 8, or 16. The CPUs are each connected to shared memory and an I/O system; see figure 6.1. Each CPU has four ports, a memory path selector, instruction buffers, registers, and several functional units; see figure 6.2.

We will start by describing the CPU registers. The Cray computer’s word length is 64 bits. The vector registers are set up as 8 vectors of 64 elements each (the Cray C90 has 128 elements), where each element is a 64 bit word. There are 8 scalar registers and 64 intermediate result scalar registers, each with 64 bits.

The address registers can also have an impact of the software design. There
Figure 6.1: Cray Y-MP hardware diagram for n CPUs.

Figure 6.2: Cray CPU configuration.
are 8 address registers consisting of 32 bits each and 64 additional 32 bit intermediate address registers. The intermediate address registers are primarily used for processing the address register data.

In addition to the above mentioned registers there are a variety of others that will not be discussed here because they vary somewhat between Cray’s different computer models and because they do not really have an affect on the design of the vector algorithms. For completeness we will mention the major categories; they are the vector mask, vector length, hardware performance monitor, programmable clock, control, mode, status, memory error information, exchange address, and exchange package information registers along with a number of flag bits. There are also some additional registers for parallel processing on the Cray computers which fall into either the shared resources registers or the cluster registers.

The main memory consists of either 2 or 4 sections each containing 8 subsections of 64 banks. The memory bank size depends on the model of Cray computer and the memory size configuration chosen for that model. The two Cray computers that we used were a Y-MP and M90, which have memory banks of 256K words and 8M words respectively; see Appendix B for more details. The memory is interleaved throughout all the banks, subsections, and sections. Consecutively stored data items are placed into different memory banks, and no bank is reused until all the banks have been used.

The 8 vector and 8 scalar registers coincide with the 8 subsections of a memory section. The 64 (128, C90) vector elements per vector register coincide with the 64 memory banks of each subsection.

Each CPU has its own path to each section of memory. A single CPU cannot make simultaneous accesses to the same section of memory. Each CPU has four ports: two for reading, one for writing, and one for either an instruction buffer or an I/O request; see figure 6.2. In order for a CPU to access memory, it must have both an
available port and memory path.

A CPU memory reference makes an entire memory subsection unavailable to all ports of the CPU until the reference has completed (five clock cycles). In addition, the memory reference also makes that bank unavailable to all other ports of all the other CPUs in the system until it has completed.

There are 5 basic types of functional units in each CPU. The first one is vector bit-oriented, and it consists of integer add, logical, shift, pop/parity, and secondary logical operations. The second functional unit is the floating point vector operations unit, which is also used to perform scalar floating point operations. The third functional unit is the scalar bit-oriented unit and includes integer add, logical, shift, and pop/parity operations. The fourth functional unit is the address computational unit. The fifth functional unit is the instruction decode and fetch unit.

The CPU’s functional units all operate independently of each other. In addition, the functional units are fully segmented (pipelined). That is, the intermediate steps required to complete an operation are broken down into one clock period segments. Once the pipeline is full, results are produced at one per clock period. The number of segments that a functional unit has depends on the complexity of the functions it must perform; hence, the functional units are mostly of different segment lengths.

The functional units can also be chained together, and because they operate independently, it is possible to perform several operations concurrently. For instance, let \(a, b, c\) be vectors and \(d\) a scalar; then the vector–scalar operation \(a(i) = b(i) \times c(i) + d\) can be performed with one result \(i\) per clock period for all \(i\). The concurrent operations taking place are two vector loads (\(b\) and \(c\)), a vector multiply, a scalar add, and a vector store (\(a\)).

The Cray Y-MP can be forced to perform, essentially, as a sequential computer by compiling the code with the NOVEC compiler directive and by setting the compiler
optimization flag for no vectorization, e.g. `cf77 -O vector0` or `cf77 -o nonvector`. This can be very useful for determining the actual speedup associated with vectorizing the code.

**Definition 6.1.1** The speedup factor for a code or code fragment is defined as

$$S_T = \frac{\tau_{\text{old}}}{\tau_{\text{new}}}$$

(6.1)

where $\tau_{\text{old}}$ and $\tau_{\text{new}}$ are the execution times for the old and new codes respectively.

The speedup factor as we have defined it is sometimes called the relative speedup. The speedup factor can be used to measure the vectorization speedup by setting $\tau_{\text{old}}$ to the non-vectorization execution time and $\tau_{\text{new}}$ to the vectorization execution time.

To get the best performance for a given algorithm and still use only the higher level language (FORTRAN in our case), there are several things that can be done. First recall that the code must also be able to execute on a sequential computer and remain as portable as possible. This consideration means that we can not use any machine specific routines or instructions that will make the code non-portable. This decision limits the options available but does not impose too many difficulties or reduce the performance gains (by very much) that can be achieved.

The things that we can do are to control the data structure designs, implementation and choice of the algorithm, and the use of compiler options and directives.

### 6.2 Memory Mapping and Data Structures

Leading array indices should always be declared to be of odd length, e.g. 129. This is because most vector (and non-vector) computers have their memory set up in banks, and the data are distributed across the memory banks in various ways depending on the particular computer. The number of banks is almost always set up to be a power
of two. So, if an array is declared to be of even length, then there is a strong possibility for memory bank conflicts when fetching and storing consecutive array elements. The memory bank conflicts can significantly slow down the performance of a code. Typically on many vector computers, the speedup obtained when using an odd length for an array declaration is a speedup factor of 2 to 4 over using an even length.

Vectorization takes place only for inner loops, but in special circumstances nested loops may be combined into a single loop that is vectorized by the compiler. The most frequently used inner looping index should have the longest length possible. For example, if a double DO loop, with no vector dependencies, is indexed over \( i = 1, \ldots, N \) and \( j = 1, \ldots, M \) where \( N > M \), then the loop over \( i \) should be the inner one.

The data structures of the arrays should also be set up in such a way as to allow the most frequently used looping index to be placed as near the beginning (leftmost) index position as is possible.

By doing these simple restructurings of the arrays, it has been found that a speedup factor of anywhere from 2 to 8 can be obtained for the various components of the black box multigrid codes.

### 6.3 Scalar Temporaries

A scalar temporary is a scalar that is set equal to a vector expression. Scalar temporaries are most often used to express common subexpressions in a loop. The use of scalar temporaries is a very delicate issue, and the extent of their use varies with the complexity of the computation and the compiler that is used. It can often be the case that a speedup factor of 1.6 to 3 can be observed in code with the proper use of scalar temporaries over code that has either over-used or under-used scalar temporaries.

For the black box multigrid codes, the generation of the grid transfer operator coefficients and the formation of the coarse grid operators are highly susceptible to the
use of scalar temporaries due to the size and complexity of these computations. It is not possible to obtain an optimum implementation using scalar temporaries and have the code remain optimum and portable at the same time. However, we did manage to find a reasonable compromise which should be fairly efficient over a wide range of computers and compilers.

### 6.4 In–Code Compiler Directives

The use of compiler directives in the code can also greatly enhance the compiler’s ability to optimize the code. When compiler directives are not used, the compiler may do several things that can slow down the performance. The compiler may fail to vectorize a loop because it suspects that there may be data dependencies present. The compiler may add run–time code to determine the vector length, amount of loop unrolling that can be performed, or whether or not a loop can be vectorized or not. Using compiler directives can eliminate many of these problems and in addition can speed up both the execution and compilation time by eliminating the need for run–time checking. Compiler directives vary from computer to computer, but almost all have the advantage that they are interpreted as comment lines by other compilers. This means, however, that one will have to change the compiler directives when one moves the code from one type of computer to another, or that one will have to add all the compiler directives to the code for as many different computers as are likely to be used. We have chosen to place only the compiler directives for the Cray Y–MP in the vector versions of the black box multigrid codes.

The most commonly used compiler directives are:

1. Ignore Vector Dependencies. (CDIR$ IVDEP)
2. Scalar Operations. (CDIR$ SCA)
3. Inlining (CDIR$ INLINE)

### 6.5 Inlining

The use of subroutines to make a code modular has been in fashion for a couple of decades now. It has been found that codes are more readable and that the software can often be reused when the codes are split up modularly depending on their functionality. However, this modularity has the unfortunate effect of slowing down the codes’ performance on many computers. This is due mostly to the overhead involved with executing a subroutine. Many compilers have a directive for inlining. Inlining consists of the compiler taking the subroutine and rewriting it at the same code level as the calling program. This eliminates the overhead of the subroutine call from the code’s performance. One may ask why all compilers do not do inlining automatically. There are several answers to this question. Not all compilers can perform inlining. It takes much longer to compile a code that requests inlining. The executable is usually larger when inlining is requested because a subroutine may be called many times from different places in the code, and for each instance the code is copied into the calling routine, creating many copies of the same piece of code for that subroutine.

The black box multigrid codes use subroutines mostly to separate out the different multigrid components by functionality. Thus, the use of inlining does not cause the executable to be really huge, even though there are several instances of subroutines being duplicated. The duplications are mostly for the smoother and computing the residual and its $l_2$ norm.
6.6 Loop Swapping

Many algorithms, when implemented, contain nested DO loops. For the best performance, the inner loop should have the longest vector length possible while maintaining the loop’s vectorization. The speedup gain is dependent on the length of the loops and the amount of computation in the inner loop, but speedup factors from 1 to 6 are not uncommon.

The black box multigrid codes have been implemented with the longest vector length in the inner loop. This decision has meant that compiler directives were needed to inform the compiler that the inner loops contained no vector dependencies.

6.7 Loop Unrolling

Short loops can often generate more overhead instructions than computational instructions. Because of this fact, many compilers will automatically unroll loops that have a small fixed length. However, loops with a parameter as their indexing limit and very little computational work may sometimes be unrolled partially to leave more computational work per loop iteration, but not all compilers are capable of performing this kind of loop unrolling.

There are several short loops in the smoothing subroutines of the black box multigrid codes that can benefit from loop unrolling. The performance speedup factors for these loops range from 1.3 to 3.

6.8 Loops and Conditionals

Loops with a conditional statement in them will not vectorize. By conditional statements we mean IF statements and computed GOTO statements.

In the subroutine that computes the grid transfer coefficients in the black box multigrid codes, a test needs to be performed inside several of the loops to determine
which form the computation is to take; see the chapter 3 on grid transfer operators. The IF statement for the test is converted into a computation involving the maximum intrinsic function, which is used to combine both computational branches of the test’s outcome. This device makes the loop vectorizable, giving a speedup factor of nearly 100 over the non-vectorizable version involving the IF statement.

6.9 Scalar Operations

Loops and blocks of computation that can not be vectorized should be written in a form that the compiler can recognize as common scalar operations. Some of the common forms are simple vector dependencies and recursions. If the compiler can recognize these common scalar operations, it is possible for it to use specialized scalar libraries that make the best use of the hardware to obtain the best performance. The performance gain is usually modest and ranges up to a speedup factor of 1.5.

6.10 Compiler Options

The performance of a code is greatly dependent on the ability of the compiler to recognize code fragments and optimize them. In order for the compiler to generate fast and efficient code, it is not only important to write efficient code and to use in-code compiler directives, but also to use the appropriate compiler options in the compile commands. The choice of these options can dramatically speed up or slow down the performance.

The compiler options that were used on the Cray Y-MP for all the black box multigrid codes were,

```bash
cf77 -Zv -Wf"-dz -o inline3,aggress -A fast" files.f
```

where -Zv means to vectorize for a single processor and use the dependence analyzer
fpp; - WF indicates that the options in the quotes are for the FORTRAN compiler cft77; -dz says to disable debugging and other utilities; -o inline3, agress is two commands, where inline3 means to inline subroutines up to three levels deep, and agress means to raise the limits on stacks, internal tables, and searches to allow some loops to be optimized that might not otherwise vectorize; -A fast means to use the fast addressing mode and not the longer addresses that use indirect addressing under the default full addressing mode.

The use of these compiler options can easily double the performance of the black box multigrid codes by allowing the compiler to vectorize more fully the codes and by cutting out some of the overhead that is generated for software analysis tools.

6.11 Some Algorithmic Considerations for Smoothers

We are interested in the multicolor point and line Gauss–Seidel methods and in the ILLU x-line method for smoothers. All the Gauss–Seidel methods vectorize quite easily, while the ILLU method does so only marginally.

6.11.1 Point Gauss–Seidel Relaxation

While we have said that the vector and sequential algorithms are the same, when it comes to the point Gauss–Seidel smoother’s implementation there is a difference. The red/black point Gauss–Seidel method, performed in the normal fashion – first all red points are updated followed by updating all the black points – is very inefficient on cache based systems, requiring all the data to pass through the cache twice. An easy modification can be made that allows the data to only pass through the cache once can be found in [35]; the algorithm is

Algorithm 6.11.1 Let the grid be of size $N_x \times N_y$, then the cache based red/black Gauss–Seidel algorithm is given by:
1. update all red points in row $j = 1$

2. Do $j = 2, N_y$

3. update all red point in row $j$

4. update all black points in row $j - 1$

5. End Do

6. update all black points in row $j = N_y$

This cache based algorithm can give a speedup factor of two orders of magnitude on many cache based RISC processors. The 4-color Gauss-Seidel method can also be modified in a similar way to achieve the same type of speedup.

The cache algorithm is not as useful on vector computers because the vector length is now only over a single line rather than over the entire grid. The loss in vector performance is not much provided that the rows are at least as long as the computer’s vector length. The shorter the vectors become, compared to the machine’s vector length, the faster the performance drops off. The Cray Y-MP vector computers do not have a cache and should not use the cache based Gauss-Seidel algorithm, but the RISC processor computers should use this algorithm because they all employ caches.

6.11.2 Line Gauss–Seidel Relaxation  Zebra line Gauss–Seidel relaxation requires the solution of tridiagonal systems for the line equations. The tridiagonal systems are solved by Gaussian elimination. There are two approaches to solving the equations – either we solve lines every time from scratch or we factor the tridiagonal systems, saving the LU decompositions, and solve the factored systems. The first approach will use less memory by not having to save the LU decompositions, but the total solution time will take longer. The second approach is more favorable, if enough memory is available, because the LU decompositions account for 40% of the operation
count for the solution of the tridiagonal system of equations.

The LU factorization and solution phases for a tridiagonal system are both inherently sequential with essentially no vectorization. However, we are not solving just a single line, but a zebra ordering of all the lines on the grid. Vectorization can be obtained by performing each step of the decomposition and solution on all of the lines of the same color simultaneously. The benefits of vectorization diminish as the grids become coarser because there are fewer lines to vectorize across, but for standard coarsening the lines are also becoming shorter, reducing the amount of sequential work that must be performed. In this respect the standard coarsening algorithm is more efficient than the semi-coarsening algorithm.

On a RISC processor cache based system it is best to perform the LU solution on each tridiagonal x-line separately because it requires the data to pass through the cache only once. However, the zebra y-line Gauss-Seidel relaxation should use the vector algorithm, looping over all the lines in the x direction, which only requires the data to pass through the cache once.

We could have used the Linpack (Lapack) tridiagonal factorization and LU solvers, SGTTRF and SGTTTRS respectively, but they can not perform the factorization or back substitution for more than one system at a time and they do not use the cache in the same way. The routines are fine for a stand alone system, but they can not take advantage of the vectorization or cache potential that can be obtained by writing our own routines that have knowledge of the entire smoothing process. For this reason we have implemented our own solvers.

6.12 Coarsest Grid Direct Solver

The coarsest grid problem is either a single grid line (in the case of a rectangular fine grid with standard coarsening or in the case of the semi-coarsening algorithm)
or a small system of equations. The single grid line equation is a tridiagonal system which is easily solved, and we have chosen to save the LU factorization.

The small system of equations is a banded system. We chose to use the Linpack general banded routines, SGBFA and SGBSL, because they existed on all the machines we used. Lapack would be an even better choice, but not all the machines to which we had access had it installed. In addition, the implementation of the Linpack routines were optimized at our institution for our computers.

6.13 \(l_2\)-Norm of the Residual

The \(l_2\)-norm of the residual is used in the black box multigrid solver as one of the determinations for when to stop the multigrid iterations. It has also been used in the test codes to determine the reduction factors for an iteration and for various components. The computation of the norm is straightforward, but since the computation involves the sum of a large number of floating point values, it might be wise to ask if the result has any meaning. The question is quite valid because it is well known that summing floating point numbers can lead to errors in the resulting summation depending on how the actual sum is computed.

Originally the norm was computed by adding the squares of the residuals to the running sum total. We will call this type of summation the naive summation algorithm, and define it to be the following.

**Algorithm 6.13.1** Let \( r \) be a floating point array of \( N \) values, \( r_i, i = 1, \ldots, N \). The naive summation algorithm for \( \sum_{i=1}^{N} r_i \) is given by the following:

\[
\text{sum} = 0.
\]

Do \( i = 1, N \)

\[
\text{sum} = \text{sum} + r(i)
\]
The computed sum is equal to \( \sum_{i=1}^{N} r_i (1 + \delta_i) \) where \( |\delta_i| < (n-i)\epsilon \), \( \epsilon \) is the machine epsilon (which is the increment between representable floating point numbers), and \( n \) is the number of units (digits) in the last place that are in error.

The naive summation algorithm vectorizes, and will be unrolled several iterations by good optimizing compilers. It also performs very well on cache based RISC computers because the data is usually stored contiguously.

At first glance this may all seem to be of little importance, since the norm of the residual is only being used as a stopping criterion. However, after looking at the experimental and testing data there were many cases in which the convergence criterion was just barely missed and an additional multigrid iteration was performed. After a more extensive look at the norms, it was determined that several of the cases did not actually need to perform the additional iteration. To make matters worse, a couple of cases were found that had stopped prematurely because the norm of the residual was incorrectly computed.

It should be noted that these cases showed up quite often on the workstations and only rarely on the Cray computers. The trouble has to do with floating point arithmetic and the loss of accuracy. The Cray computers, which have a longer word length, perform arithmetic at a much higher precision than the workstations and hence very rarely encounter such trouble.

One approach to fixing this problem is to use higher precision for the summation, which is usually accomplished by doubling the current precision. The doubling algorithm uses the naive summation algorithm, but doubles the precision of all arithmetic operations, which gives the sum equal to \( \sum_{i=1}^{N} r_i (1 + \sum_{j=i}^{N} \delta_j) \) where \( |\delta_j| \leq \epsilon \). This would appear to be the answer that we are looking for except that it can execute
very slowly. Doubling the precision on 32 bit RISC workstations and Cray Y-MP means that the higher precision arithmetic is handled in software and not in the hardware. We will discuss this point in more detail later.

The loss of accuracy in the summation process can easily be fixed with very little extra cost on sequential machines by using the Kahan summation algorithm [40], [56]. The Kahan summation algorithm can be described as follows.

**Algorithm 6.13.2** Let \( r \) be a floating point array of \( N \) values, \( r_i, i = 1, \ldots, N \). The Kahan summation algorithm for \( \sum_{i=1}^{N} r_i \) is given by the following:

\[
\begin{align*}
\text{sum} &= r(1) \\
\text{correction} &= 0.
\end{align*}
\]

**Do** \( i = 2, N \)

\[
\begin{align*}
\text{next\_correction} &= r(i) - \text{correction} \\
\text{new\_sum} &= \text{sum} + \text{next\_correction} \\
\text{correction} &= (\text{new\_sum} - \text{sum}) - \text{next\_correction} \\
\text{sum} &= \text{new\_sum}
\end{align*}
\]

**EndDo**

The computed sum is equal to \( \sum_{i=1}^{N} r_i(1 + \delta_i) + O(N\epsilon^2)\sum_{i=1}^{N} r_i \) where \( |\delta_i| \leq 2\epsilon \) and \( \epsilon \) is the machine epsilon.

The difference between the two algorithms is now much clearer, since each summand in Kahan summation is perturbed by only \( 2\epsilon \) instead of the perturbations as large as \( n\epsilon \) in the naive algorithm.

The Kahan summation algorithm is not vectorizable, and even though the loop can be unrolled by the compiler, it is still miserably slow on the Cray Y-MP. However, on the sequential workstations it is only about twice as slow as the naive

142
Table 6.1. Cray Y-MP timings for the naive, Kahan, and doubling summation algorithms in seconds. The numbers in parenthesis are the timing ratios relative to the naive algorithms times.

<table>
<thead>
<tr>
<th>$N$ elements</th>
<th>Naive time</th>
<th>Kahan time (tk/tn)</th>
<th>Double time (td/tn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>6.702E-6</td>
<td>2.161E-5 (3.22)</td>
<td>6.185E-5 (9.23)</td>
</tr>
<tr>
<td>$10^3$</td>
<td>1.208E-5</td>
<td>1.788E-4 (14.8)</td>
<td>3.026E-4 (25.1)</td>
</tr>
<tr>
<td>$10^4$</td>
<td>6.964E-5</td>
<td>1.751E-3 (25.1)</td>
<td>2.708E-3 (38.9)</td>
</tr>
<tr>
<td>$10^5$</td>
<td>6.433E-4</td>
<td>1.748E-2 (27.2)</td>
<td>2.677E-2 (41.6)</td>
</tr>
<tr>
<td>$10^6$</td>
<td>6.382E-3</td>
<td>1.746E-1 (27.4)</td>
<td>2.675E-1 (41.9)</td>
</tr>
</tbody>
</table>

We can now compare the three summation algorithms on both the vector and RISC (Sparc-5) computers. The summation timings on the Cray Y-MP are given in table 6.1. On the Cray Y-MP double precision arithmetic is performed using software, and hence it is quite slow. Recall that the Cray Y-MP does not need to use double precision because it already uses 64 bits for single precision, which was found to be very adequate for the summation process. As we have already said, the Kahan summation algorithm does not vectorize and is quite slow. It is obvious that the only practical implementation is to use the naive summation algorithm on the Cray Y-MP.

The Sparc-5 timings, in table 6.2, show that the Kahan algorithm is about twice as slow as the naive algorithm, and that the double precision algorithm is only about 10% slower than the naive algorithm. Computing in double precision is very adequate for our needs.

We are now faced with three summation algorithms. Our choices now appear to be either to settle for three versions or just to use the naive summation algorithm, as before, and not worry about missing or adding an additional multigrid iteration. For the timing data presented in the later sections of this thesis we have chosen just to use the naive summation algorithm, but we believe that it is better to make two
Table 6.2. Sparc5 timings for the naive (tn), Kahan (tk), and doubling (td) summation algorithms in seconds. The numbers in parenthesis are the timing ratios relative to the naive algorithms times.

<table>
<thead>
<tr>
<th>N elements</th>
<th>Naive tn</th>
<th>Kahan tk (tk/tn)</th>
<th>Double td (td/tn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>2.70E-5</td>
<td>6.00E-5 (2.22)</td>
<td>3.20E-5 (1.19)</td>
</tr>
<tr>
<td>$10^3$</td>
<td>2.67E-4</td>
<td>6.04E-4 (2.27)</td>
<td>3.27E-4 (1.23)</td>
</tr>
<tr>
<td>$10^4$</td>
<td>2.75E-3</td>
<td>6.39E-3 (2.32)</td>
<td>3.31E-3 (1.21)</td>
</tr>
<tr>
<td>$10^5$</td>
<td>2.86E-2</td>
<td>6.59E-2 (2.30)</td>
<td>3.44E-2 (1.20)</td>
</tr>
<tr>
<td>$10^6$</td>
<td>2.88E-1</td>
<td>6.61E-1 (2.29)</td>
<td>3.56E-1 (1.24)</td>
</tr>
</tbody>
</table>

or even three versions of the code. For vector computers either the naive or double summation algorithm should be used depending on whether double precision arithmetic is implemented in hardware at either 64 or 32 bits respectively. On sequential machines either the Kahan or the double summation algorithm should be used depending on the implementation of double precision arithmetic.

### 6.14 2D Standard Coarsening Vector Algorithm

To this point we have discussed several issues concerning the black box multigrid components, vectorization, and programming on the Cray Y-MP, but we have not explicitly mentioned what our choices were for the code. We will do so now.

We have implemented the code being aware of all the vectorization issues and using the most efficient choices that have been discussed above.

#### 6.14.1 Coarsening

We used standard coarsening, taking every other fine grid point in both coordinate directions to form the coarse grid.

#### 6.14.2 Data Structures

The data structures for the grid equations are grid point stencil oriented. The mesh of unknowns has been augmented with a border of fictitious zero equations. The border is used to avoid having to write special code.
\begin{align*}
L &= (N_x, N_y, 9), \quad u = (N_x, N_y), \quad f = (N_x, N_y), \quad \text{Work\_Space} = (N_x, N_y, 3) \\
\text{Grid m} & \quad \text{Grid m-1} \quad \ldots \quad \text{Grid 1} \\
\text{Prolongation} &= (N_x, N_y, 8), \quad \text{Restriction} = (N_x, N_y, 8) \\
\text{Grid m-1} & \quad \ldots \quad \text{Grid 1} 
\end{align*}

Figure 6.3. Data structure layout for m grid levels; grid equations, work space, and grid transfer operators.

to handle the boundary of the grid. This arrangement makes the code easier to write and more efficient for vector operations.

There are several arrays to hold the grid equations: the discrete coefficient array, the array of unknowns, and the right hand side array. There are also a few extra auxiliary arrays to hold the grid transfer operator coefficients, the residual, and the LU decompositions of the line solves and of the coarsest grid problem.

Each grid level has its own data structure of the appropriate size that has been allocated, via pointers, as part of a larger linear array for each data type structure; see figure 6.3. This arrangement makes memory management for the number of grid levels easier.

6.14.3 Smoothers We have implemented the multicolor ordering point, line, and alternating line Gauss–Seidel methods and the ILLU x-line method. The cached based Gauss–Seidel algorithm is not used.
6.14.4 Coarsest Grid Solver  The coarsest grid solver is a direct solver using LU factorization.

6.14.5 Grid Transfer Operators  There are three choices for the grid transfer operators, discussed in chapter 3, that were implemented. They are the ones discussed in sections 3.5.1, 3.5.3, and 3.6.1.

6.14.6 Coarse Grid Operators  The coarse grid operators are formed using the Galerkin coarse grid approximation, which uses the grid transfer operators and the fine grid operator.

6.15 2D Semi–Coarsening Vector Algorithm

The semi–coarsening code was originally implemented by Joel E. Dendy, Jr. We have re-implemented it in a more efficient form to gain a speedup of about 5 over the previous vectorized version while maintaining and improving the portability of the code. The new implementation has kept all the functionality of the previous version.

6.15.1 Data Structures  The data structures for the grid equations are the same as those for the standard coarsening code including the fictitious border equations. However, now the work space array is reduced to \((N_x, N_y, 2)\) and the prolongation and restriction coefficient arrays are of length 2 instead of 8.

6.15.2 Coarsening  Semi–coarsening in the y-direction was used, taking every other fine grid point in the y-direction to form the coarse grid.

6.15.3 Smoothers  Red/black x–line Gauss–Seidel relaxation is used for the smoother. As an experiment the x–line ILLU method was also implemented.
6.15.4 **Coarsest Grid Solver**  The coarsest grid solver is either the direct LU factorization solver or a tridiagonal solver in the case that coarsening is continued until only one x-line remains.

6.15.5 **Grid Transfer Operators**  The grid transfer operator is the one used in section 3.6.1 applied in only the y-direction.

6.15.6 **Coarse Grid Operators**  The coarse grid operators are formed using the Galerkin coarse grid approximation, using the grid transfer and fine grid operators.
CHAPTER 7

2D NUMERICAL RESULTS

The numerical results in this section are for the two dimensional domain versions of the black box multigrid solvers.

7.1 Storage Requirements

The black box multigrid solvers present some tradeoff issues for speed versus storage. The algorithms require that we perform a number of tasks involving grid operators and grid transfer operators. We can choose to save storage and sacrifice speed by computing these when we need them. However, since we are using a geometric multigrid method these computations are not cheap. The most expensive is the formation of the coarse grid operators.

For the 2D grid levels we need storage for the grid equations (unknowns, coefficients, and right hand side), the grid transfer operators, and temporary work space. Let \( N_x \) and \( N_y \) be the number of grid points in the \( x \)- and \( y \)-directions respectively. We can compute how much storage will be needed by adding up the amount for grid point. We need 9 locations for the coefficient matrix and 1 each for the unknowns and right hand side. For the standard coarsening method we need 16 locations for both the grid transfer operator coefficients, and another 3 for temporary work. For the semi-coarsening method we need 4 locations for both the grid transfer operator coefficients, and another 2 for temporary work. We can ignore the amount of storage for the coarsest grid direct solver for now because it will remain constant and small when compared
Table 7.1. Storage requirements for various grid sizes for the grid unknowns and grid operators on the Cray Y-MP.

<table>
<thead>
<tr>
<th></th>
<th>Unknowns</th>
<th>coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N \times N$</td>
<td>$N^2$</td>
<td>$\sum N_i^2$</td>
</tr>
<tr>
<td>9 × 9</td>
<td>81</td>
<td>115</td>
</tr>
<tr>
<td>17 × 17</td>
<td>289</td>
<td>404</td>
</tr>
<tr>
<td>33 × 33</td>
<td>1089</td>
<td>1493</td>
</tr>
<tr>
<td>65 × 65</td>
<td>4225</td>
<td>5718</td>
</tr>
<tr>
<td>129 × 129</td>
<td>16641</td>
<td>22359</td>
</tr>
<tr>
<td>257 × 257</td>
<td>66049</td>
<td>88408</td>
</tr>
<tr>
<td>513 × 513</td>
<td>263169</td>
<td>351577</td>
</tr>
<tr>
<td>1025 × 1025</td>
<td>1050625</td>
<td>1402202</td>
</tr>
</tbody>
</table>

to the rest. This means that we need 30 locations for the standard coarsening and 17 for the semi-coarsening. However, we do not have grid transfer coefficients stored on the finest grid so we can subtract 16 and 4 locations from the total for the standard and semi-coarsening methods respectively. The amount of storage required for the 2D data structures, excluding storage for the line solves, is

$$
\begin{align*}
(30 \left( 1 + \frac{1}{4} + \cdots \right) - 16) N_x N_y &= 24 N_x N_y, \\
(17 \left( 1 + \frac{1}{2} + \cdots \right) - 4) N_x N_y &= 30 N_x N_y
\end{align*}
$$

for the standard and semi-coarsening methods respectively. If we only have a 5-point operator on the finest grid we do not need to store the other 4 coefficients and then the storage requirements become $20N_xN_y$ and $26N_xN_y$ for the standard and semi-coarsening methods respectively.

The growth of the storage requirements versus the grid size can be seen in Table 7.1. The first column is the number of grid points and is also the amount of storage needed for the grid unknowns or right hand sides for that grid level. The second column is the number of grid points for the indicated fine grid down to the coarsest grid level ($3 \times 3$). The last two columns are the amount of storage needed for
Table 7.2. The actual storage requirements for various grid sizes for the codes BMGNS, SCBMG, and MGD9V given in terms of number of real variables on the Cray Y-MP.

<table>
<thead>
<tr>
<th>$N \times N$</th>
<th>SCBMG</th>
<th>BMGNS</th>
<th>MGD9V</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 × 9</td>
<td>4721</td>
<td>4359</td>
<td>2063</td>
</tr>
<tr>
<td>17 × 17</td>
<td>12938</td>
<td>10309</td>
<td>7072</td>
</tr>
<tr>
<td>33 × 33</td>
<td>39635</td>
<td>29235</td>
<td>25777</td>
</tr>
<tr>
<td>65 × 65</td>
<td>132908</td>
<td>96001</td>
<td>97986</td>
</tr>
<tr>
<td>129 × 129</td>
<td>476165</td>
<td>345999</td>
<td>381651</td>
</tr>
<tr>
<td>257 × 257</td>
<td>1783454</td>
<td>1312669</td>
<td>1506020</td>
</tr>
<tr>
<td>513 × 513</td>
<td>6868535</td>
<td>5113515</td>
<td>5982965</td>
</tr>
<tr>
<td>1025 × 1025</td>
<td>26895056</td>
<td>20860411</td>
<td>23849734</td>
</tr>
</tbody>
</table>

The price of memory has been falling steadily for years, and as a consequence computers are being built with more and more memory. Due to this phenomenon and in the interest of speed, we have chosen to compute these operators once and store them. We could have chosen just to store the grid operators and compute the grid transfer operators when they are needed, and this procedure might even be practical for the three dimensional problem codes when memory is used up at an alarming rate.

The actual storage requirements for the various codes versus the grid size is given in table 7.2. The BMGNS code, which uses zebra alternating line Gauss–Seidel, requires the least amount of storage for grids larger than 65 × 65. The MGD9V codes uses less storage for small grid levels because its grid transfer operators require only a fourth the storage of BMGNS, due to symmetries. However, MGD9V must store the ILLU smoother decomposition for each grid level, and this additional storage becomes more significant for larger grids. The SCBMG code always requires more storage because it uses semi-coarsening.
Table 7.3. Speedup of the new vectorized version, \textit{vbmg}, and Dendy’s vectorized version, \textit{bmg}, over the scalar version of \textit{bmg}.

<table>
<thead>
<tr>
<th>grid size $N$ ($N \times N$)</th>
<th>BMGNS</th>
<th>VBMGNS</th>
<th>Speedup ($\text{bmgns/vbmgns}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2.3</td>
<td>13.8</td>
<td>5.8</td>
</tr>
<tr>
<td>17</td>
<td>3.2</td>
<td>21.5</td>
<td>6.7</td>
</tr>
<tr>
<td>33</td>
<td>5.5</td>
<td>33.6</td>
<td>6.1</td>
</tr>
<tr>
<td>65</td>
<td>7.2</td>
<td>43.3</td>
<td>6.0</td>
</tr>
<tr>
<td>129</td>
<td>9.5</td>
<td>54.2</td>
<td>5.7</td>
</tr>
<tr>
<td>257</td>
<td>11.4</td>
<td>68.4</td>
<td>6.0</td>
</tr>
<tr>
<td>513</td>
<td>14.1</td>
<td>81.8</td>
<td>5.8</td>
</tr>
<tr>
<td>1025</td>
<td>16.2</td>
<td>92.4</td>
<td>5.7</td>
</tr>
</tbody>
</table>

7.2 Vectorization Speedup

The original functionality of the Black Box Multigrid codes [29] was kept and used as a baseline. There were several restrictions that we imposed on what could be done with the codes. The desire behind developing and optimizing the black box multigrid codes was to maintain as much of the original functionality of the black box multigrid method. The original method was implemented as a research prototype. Even though care was taken to keep it somewhat portable and to adher to most of the SLATEC guidelines [38], at least for the documentation, it still had a long way to go before being ready for release to the scientific and engineering community. We wanted to maintain all of the cycling strategies, except for those controlled by the MCYCL parameter, which is only valid when performing the initial stages of an F-cycle. We also decided to remove the truncation error estimates, ITAU, because they were no longer meaningful.

The speedup of the new vectorized version of the standard coarsening code is given in table 7.3. The table shows that while Dendy’s original code had some vectorization in it, the new version has much more. The speedup is due to a variety of
The speedup is not consistent over the range of grid sizes because of the effect of vector length and memory cache misses. There may also be some effects due to the fact that the timings were done in a time sharing environment, but they should be quite small when compared to the other issues.

While table 7.3 is only for the standard coarsening code, it also reflects the speedup seen in the semi-coarsening code. The data in table 7.4 compares the execution time in seconds on the Cray Y-MP and the CM-2 of Dendy’s semi-coarsening version to the new vectorized semi-coarsening version on the Cray Y-MP. The table shows that the new semi-coarsening version is roughly six times faster than Dendy’s on the Cray Y-MP. The poor performance of the CM-2 version of scbmg is due to the much slower computing units and the communications penalty on the CM-2. The CM-2 version was run on one quarter of 1024 nodes (256 processors) under the slicewise model.

The timing comparisons are perhaps clearer when presented in graphical form. We compare the three main types of algorithms represented by BMGNS, SCBMG, and

Table 7.4. Comparison and speedup of the Semi-coarsening versions. scbmg is Dendy’s version, and vscbmg is the new vectorized version. The — entries mean that there is no data.

<table>
<thead>
<tr>
<th>grid size $N$ ($N \times N$)</th>
<th>CM-2 SCBMG</th>
<th>Cray Y-MP SCBMG</th>
<th>VSCBMG</th>
<th>Speedup (scbmg/vscbmg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>—</td>
<td>.011</td>
<td>.0019</td>
<td>5.8</td>
</tr>
<tr>
<td>64</td>
<td>.65</td>
<td>.04</td>
<td>.0059</td>
<td>6.8</td>
</tr>
<tr>
<td>128</td>
<td>.99</td>
<td>.09</td>
<td>.015</td>
<td>6.0</td>
</tr>
<tr>
<td>256</td>
<td>1.84</td>
<td>.27</td>
<td>.045</td>
<td>6.0</td>
</tr>
<tr>
<td>512</td>
<td>4.55</td>
<td>.95</td>
<td>.16</td>
<td>5.9</td>
</tr>
<tr>
<td>1024</td>
<td>—</td>
<td>3.69</td>
<td>.64</td>
<td>5.8</td>
</tr>
</tbody>
</table>
MGD9V on the Cray Y–MP using a single processor. While the setup is done only once, the solution may require many V–cycle iterations. Hence, the time for the setup and the time for one V–cycle are presented separately.

The BMGNS code uses zebra alternating line Gauss–Seidel relaxation for the smoother (which is the most expensive option), and the non-symmetric grid operator collapsing method involving $\sigma L$ for the grid transfer operators (which is the most expensive collapsing method).

The setup time includes the computation of the coarse grid operators, the grid transfer operators (prolongation and restriction), the factorization for the coarsest grid solver and any setup required by the smoother. Figure (7.1) represents the setup time in seconds for the three codes.

It is not surprising that SCBMG is the fastest, since it only needs to compute grid transfer coefficients in the y–direction and factorization for lines in the x–direction, while BMGNS has to compute the six additional grid transfer coefficients for each prolongation and restriction operator and an additional line factorization for the smoother. The factorization for the coarsest grid direct solver must also be computed for BMGNS. The MGD9V algorithm saves time in computing the grid transfer coefficients over BMGNS, and like SCBMG it does not need to compute a factorization for the direct solver on the coarsest grid. However, the factorizations and setup for the ILLU smoother are very expensive and are not really vectorizable.

The times for one complete V–cycle are given in figure (7.2), excluding the setup time. The V–cycle is a $V(1,1)$-cycle for BMGNS and SCBMG, and a $V(0,1)$-cycle for MGD9V.

The V–cycle timing graph shows the same relationship as the setup timing graph, except that the time for one V–cycle is much less than the setup time.
Figure 7.1: Comparison of Setup time for BMGNS, SCBMG, and MGD9V
Figure 7.2: Comparison of one V-cycle time for BMGNS, SCBMG, and MGD9V
Table 7.5. Operation counts (actual) for standard coarsening black box multigrid Setup phase.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L^H$</td>
<td>$369N_xN_y$</td>
</tr>
<tr>
<td></td>
<td>$256N_xN_y$</td>
</tr>
<tr>
<td>$\sigma L/\sigma L$</td>
<td>$174N_xN_y - 135N_x - 135N_y + 96$</td>
</tr>
<tr>
<td></td>
<td>$122N_xN_y - 91N_x - 91N_y + 60$</td>
</tr>
<tr>
<td>$\sigma L/L$ Hybrid</td>
<td>$166N_xN_y - 131N_x - 131N_y + 96$</td>
</tr>
<tr>
<td></td>
<td>$114N_xN_y - 87N_x - 87N_y + 60$</td>
</tr>
<tr>
<td>Schaffer</td>
<td>$168N_xN_y - 112N_x - 112N_y + 56$</td>
</tr>
<tr>
<td></td>
<td>$120N_xN_y - 88N_x - 88N_y + 56$</td>
</tr>
<tr>
<td>ZxLGS decomposed</td>
<td>$4N_xN_y - 3N_y$</td>
</tr>
<tr>
<td>ZyLGS decomposed</td>
<td>$4N_xN_y - 3N_x$</td>
</tr>
<tr>
<td>ZALGS decomposed</td>
<td>$8N_xN_y - 3N_x - 3N_y$</td>
</tr>
<tr>
<td>xILLU decomposed</td>
<td>$58N_xN_y - 52N_x - 106N_y + 101$</td>
</tr>
</tbody>
</table>

7.3 2D Computational Work

The amount of work performed by the various multigrid components as implemented on the Cray Y-MP are given below. The operation counts lump multiplication, division, addition, and subtraction together.

The setup phase is broken down into three parts: grid transfer operators, coarse grid operator, and smoother decomposition. The grid parameters, $N_x$ and $N_y$, are the number of coarse grid points in the respective coordinate directions. The operation counts are given in table 7.5. The first line in each block with multiple lines refers to when the grid operator has a 9-point stencil, and the second line for when it has a 5-point stencil.

The operations counts say that the $\sigma L/L$-Hybrid method is the fastest for computing the grid transfer operator coefficients. We can also see that the zebra alternating line Gauss-Seidel smoother is very cheap to decompose when compared to the ILLU smoother.

The number of operations performed to compute the residual and perform the
Table 7.6. Operation counts (actual) for standard coarsening black box multigrid Residual and grid transfer components.

<table>
<thead>
<tr>
<th>Residual</th>
<th>$18N_xN_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$10N_xN_y$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prolongation</th>
<th>$20N_xN_y - 14N_x - 14N_y + 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>$16N_xN_y$</td>
</tr>
</tbody>
</table>

grid transfers are given in table 7.6.

The amount of work needed to perform the various smoothers is given in table 7.7. Again the first line in each block refers to when the grid operator has a 9-point stencil, and the second line for when it has a 5-point stencil.

It is important to remember that the operation counts give only a rough idea of how the actual components will perform for several reasons. The first is that we have lumped all of the different arithmetic operations together into one count. The amount of time to perform the various arithmetic operations can vary up to several clock cycles, with division usually taking the longest. In addition, the operation counts say nothing about how the code will perform on the hardware. Some of the issues that can affect the performance are whether or not the operations vectorize, fill the pipelines, and require cache paging. These issues can drastically change the amount of execution time needed to perform the operations. Even though the multigrid components listed here have all been optimized in FORTRAN they still all vary with regard to these hardware issues, and hence it is not easy to predict, without running the codes, which will actually execute the fastest.

### 7.4 Timing Results for Test Problems

In this section we present some timing results of the various codes for comparing the performance of the codes and their components.
Table 7.7. Operation counts (actual) for standard coarsening black box multigrid smoother component.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>4C-PGS</td>
<td>$17N_xN_y$</td>
</tr>
<tr>
<td>2C-PGS</td>
<td>$9N_xN_y$</td>
</tr>
<tr>
<td>ZxLGS</td>
<td>$18N_xN_y$</td>
</tr>
<tr>
<td></td>
<td>$10N_xN_y$</td>
</tr>
<tr>
<td>ZyLGS</td>
<td>$18N_xN_y$</td>
</tr>
<tr>
<td></td>
<td>$10N_xN_y$</td>
</tr>
<tr>
<td>ZALGS</td>
<td>$36N_xN_y$</td>
</tr>
<tr>
<td></td>
<td>$20N_xN_y$</td>
</tr>
<tr>
<td>xILLU</td>
<td>$40N_xN_y - 17N_x - 13N_y + 9$</td>
</tr>
<tr>
<td></td>
<td>$32N_xN_y - 17N_x - 13N_y + 9$</td>
</tr>
</tbody>
</table>

Table 7.8. Problem 8: Helmholtz Equation. Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss-Seidel (lines in x followed by lines in y) for the smoother.

<table>
<thead>
<tr>
<th>Grid Size n</th>
<th>it.</th>
<th>Total Setup</th>
<th>Total Smoothing</th>
<th>Direct Solver</th>
<th>Average per Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>9.001E-4</td>
<td>6.492E-4</td>
<td>2.022E-3</td>
<td>5.609E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.856E-3</td>
<td>2.541E-3</td>
<td>5.969E-3</td>
<td>1.028E-3</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>3.811E-3</td>
<td>5.219E-3</td>
<td>1.189E-2</td>
<td>2.019E-3</td>
</tr>
<tr>
<td>65</td>
<td>4</td>
<td>8.778E-3</td>
<td>1.261E-2</td>
<td>2.783E-2</td>
<td>4.762E-3</td>
</tr>
<tr>
<td>129</td>
<td>4</td>
<td>2.630E-2</td>
<td>3.899E-2</td>
<td>8.392E-2</td>
<td>1.441E-2</td>
</tr>
<tr>
<td>257</td>
<td>4</td>
<td>8.707E-2</td>
<td>1.290E-1</td>
<td>2.807E-1</td>
<td>4.841E-2</td>
</tr>
</tbody>
</table>
To illustrate how fast these codes perform in solving a problem, we examine the timing results for solving the discontinuous Helmholtz problem from section 7.5, see table 7.8. Table 7.8 gives the timing results, in seconds, for various stages of the program execution for various grid sizes. The grid is square, so in the first column where \( n = 9 \), we mean a grid of size \( 9 \times 9 \) and so forth for the rest of the column entries. The second column gives the number of multigrid \( V(1,1) \)-cycle iterations that were needed to reduce the \( l_2 \) norm of the initial residual by six orders of magnitude. The third column gives the total setup time, which involves the time it takes to form all of the grid transfer operators, generate all the coarse grid operators, and perform any decompositions needed for the smoother. The fourth column gives the total time for smoothing. The fifth column gives the total time for the direct solver. The last column contains the average time it took to complete one \( V(1,1) \)-cycle.

We observe that the code runs fairly quickly, and that it appears to scale very well with respect to the grid size. We also note that the total setup time is about 1.7 times that of the average cycle time and that in addition this time is about 2.7 times the total smoothing time for one iteration. A more detailed examination of these relationships between the various multigrid components is given below.

The rest of the tables in this section are the results for one multigrid \( V(1,1) \)-cycle. The results are separated by multigrid components for easier comparison between the types of multigrid algorithms. All times are given in seconds of CPU time on the Cray Y-MP in single processor mode.

The time to perform the LU decomposition of the coarsest grid \( (3 \times 3) \) problem for the direct solver is \( 7.176 \times 10^{-5} \) seconds. The direct solver on the coarsest grid level \( (3 \times 3, \text{ standard coarsening}) \) takes \( 2.609 \times 10^{-5} \) seconds. These times are constant for all of the standard coarsening algorithms that use the direct solver. It should be noted that these times are based on a coarsest grid size of \( 3 \times 3 \) and that if another coarsest
grid size is chosen, then the times will also change.

The amount of work to perform the grid transfers depends on the grid size and on the type of coarsening used. A comparison between standard and semi-coarsening is given in Table 7.9. As one would expect, semi-coarsening grid transfers are faster than standard coarsening grid transfers. The standard coarsening restriction requires 4 times the work that semi-coarsening does, and prolongation requires 10 times more work. However, due to the number of grid points that they operate on and the way in which they are computed by the hardware, there is only about a factor of 3.2 to 4.0 for prolongation and 2.0 to 3.6 for restriction, depending on the grid size.

Table 7.10 gives the timings results for four standard coarsening smoothers and the semi-coarsening smoother. Note that for red/black point Gauss-Seidel relaxation, the ratios are not uniform or monotone in nature. This situation seems to be due to several factors involving the vector length, memory stride, and cache issues. The point relaxation methods seem to be much more sensitive to memory access delays. The zebra line Gauss-Seidel relaxation also shows some of these types of variation, but after taking into account multiple runs of both x- and y-line relaxation, they can be averaged out. The time variations for the line relaxations again point to memory access delays as being the main cause. The physical layout of the data in memory means that

Table 7.9. Timings in seconds for multigrid grid transfer components for one \( V(1, 1) \)-cycle for various grid sizes; comparing standard and semi-coarsening methods.

<table>
<thead>
<tr>
<th>Grid Size ( n )</th>
<th>Standard Coarsening</th>
<th>Semi - Coarsening</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prolongation</td>
<td>Restriction</td>
</tr>
<tr>
<td>9</td>
<td>5.151E-5</td>
<td>4.525E-5</td>
</tr>
<tr>
<td>17</td>
<td>1.029E-4</td>
<td>8.675E-5</td>
</tr>
<tr>
<td>33</td>
<td>2.042E-4</td>
<td>1.693E-4</td>
</tr>
<tr>
<td>65</td>
<td>4.430E-4</td>
<td>4.063E-4</td>
</tr>
<tr>
<td>129</td>
<td>1.119E-3</td>
<td>1.264E-3</td>
</tr>
<tr>
<td>257</td>
<td>3.675E-3</td>
<td>4.306E-3</td>
</tr>
</tbody>
</table>
Table 7.10. Timings for the total smoothing time in seconds for one multigrid $V(1,1)$-cycle for various grid sizes and smoothers.

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Total Smoothing Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>R/BPGS</td>
</tr>
<tr>
<td>9</td>
<td>1.57E-4</td>
</tr>
<tr>
<td>17</td>
<td>3.31E-4</td>
</tr>
<tr>
<td>33</td>
<td>6.76E-4</td>
</tr>
<tr>
<td>65</td>
<td>1.47E-3</td>
</tr>
<tr>
<td>129</td>
<td>3.91E-3</td>
</tr>
<tr>
<td>257</td>
<td>1.24E-2</td>
</tr>
</tbody>
</table>

x- and y-line relaxation require different access strides for the grid operator coefficients. A closer look at the issue shows that in almost all cases that were measured, y-line relaxation is slightly faster than x-line relaxation on the Cray Y-MP.

While table 7.10 shows that point and line Gauss-Seidel relaxation are both quite fast, we have seen from local mode analysis that they are not robust. The alternating line, ILLU, and semi-coarsening methods are robust. We observe that standard coarsening with zebra alternating line relaxation has approximately the same performance time as the semi-coarsening method, not surprising since the semi-coarsening method is performing half the number of line relaxations, only x-lines, and since the lines are at least twice as long as in the standard coarsening method. The differences between these two can then be attributed to their performance on the hardware. Both of these smoothers are much faster than the ILLU smoother, ranging from about 2.3 to 4 times faster. However, do not forget that we saw in the local mode analysis that the ILLU method was a much better smoother.

The ratio of time spent smoothing versus the time spent doing grid transfers is given in table 7.11. See the comments above under smoothers concerning the behavior of the point relaxation. The ratio of smoothing to grid transfers shows that the smoother is the dominant computation in the multigrid cycling algorithm. It also
Table 7.11. Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Timing ratios (smoothing/grid transfer) for one $V(1,1)$–cycle for various grid sizes.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>(Smoothing)/(Grid Transfers)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/BPGS</td>
</tr>
<tr>
<td>9</td>
<td>1.68</td>
</tr>
<tr>
<td>17</td>
<td>1.75</td>
</tr>
<tr>
<td>33</td>
<td>1.81</td>
</tr>
<tr>
<td>65</td>
<td>1.73</td>
</tr>
<tr>
<td>129</td>
<td>1.66</td>
</tr>
<tr>
<td>257</td>
<td>1.56</td>
</tr>
</tbody>
</table>
Table 7.12. Timings for the multigrid setup (generate all grid transfer and grid operators and perform decompositions for smoother) for one V(1,1)–cycle for various grid sizes.

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>ZALGS</th>
<th>Schaffer’s Idea</th>
<th>SCBMG</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>sL/sL</td>
<td>sL/L–Hybrid</td>
<td>ZALGS</td>
</tr>
<tr>
<td>9</td>
<td>8.174E-4</td>
<td>7.611E-4</td>
<td>9.001E-4</td>
</tr>
<tr>
<td>17</td>
<td>1.780E-3</td>
<td>1.623E-3</td>
<td>1.856E-3</td>
</tr>
<tr>
<td>33</td>
<td>3.914E-3</td>
<td>3.583E-3</td>
<td>3.811E-3</td>
</tr>
<tr>
<td>65</td>
<td>9.209E-3</td>
<td>8.518E-3</td>
<td>8.778E-3</td>
</tr>
<tr>
<td>129</td>
<td>2.746E-2</td>
<td>2.560E-2</td>
<td>2.630E-2</td>
</tr>
</tbody>
</table>

shows that for ILLU or semi–coarsening the smoothing is even more dominant for large grids.

The times to perform the setup for various algorithms are given in table 7.12. The column headings are $\sigma L/\sigma L$ for the grid transfer operator from section 3.5.1; $\sigma L/L$–Hybrid for the grid transfer operator from section 3.5.3; and Schaffer’s idea for the grid transfer operator from section 3.6. We present the setup timing results for the codes using the zebra alternating line Gauss–Seidel relaxation, since it requires the decompositions of the both the x– and y–line solves.

The number of operations that it takes to form the coarse grid operators, $369n_xn_y$, per grid level, dominates the number of operations that it takes to form the grid transfer operators. The number of operations it takes for the decompositions for the ILLU method is even greater, as seen in the fifth column of table 7.12.

The collapsing methods for the grid transfer operators ($\sigma L/\sigma L$ and $\sigma L/L$–Hybrid) are similar, but the hybrid version requires fewer computations. It is also a little surprising that the extension of Schaffer’s ideas is also about as fast as the collapsing methods, since it has to perform lines solves to get the grid transfer coefficients. The actual number of operations, for a 9–point fine grid stencil, for $\sigma L/\sigma L$
Table 7.13. Timings for one multigrid $V(1,1)$-cycle for various grid sizes, excluding setup time.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Cycle Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/BPGS</td>
</tr>
<tr>
<td>9</td>
<td>3.856E-4</td>
</tr>
<tr>
<td>17</td>
<td>7.236E-4</td>
</tr>
<tr>
<td>33</td>
<td>1.395E-3</td>
</tr>
</tbody>
</table>

is $174n_xn_y - 135n_x - 135n_y + 96$, for hybrid $\sigma L/L$ is $166n_xn_y - 131n_x - 131n_y + 96$, and, for the extension of Schaffer’s ideas is $168n_xn_y - 112n_x - 112n_y + 56$ per grid level of size $n_x \times n_y$. This comparison shows that the collapsing methods require more computations (35%) per grid level but are not significantly slower (7%) because they vectorize, while the tri-diagonal solves for Schaffer’s ideas do not. This example shows again that operation counts don’t tell the complete story of how well a method will perform.

The times for one complete $V(1,1)$-cycle, excluding the setup and overhead time, are given for various smoothers in table 7.13. As expected, the point and line Gauss–Seidel methods are the fastest, even though we again see the strange behavior of point relaxation. It is interesting to note that the cycle time for the standard coarsening code using alternating line relaxation is virtually identical to that of the semi-coarsening code. This fact is primarily due to the fact that the smoothers, which are essentially equivalent in computation time, dominate the cycle time. The ILLU version is again the slowest method.
7.5 Numerical Results for Test Problem 8

Problem 8 is a discontinuous diffusion four-corner junction problem. This problem has appeared many times in the literature; see [1], [26], and [24]. It is defined by

\[-\nabla \cdot D \nabla u = f \quad \text{on } \Omega = (0, N_x) \times (0, N_y)\]

\[D \frac{\partial u}{\partial n} + \frac{1}{2} u = 0 \quad \text{on } \partial \Omega\]

where \(x = x^*\) and \(y = y^*\) are the interface lines for the discontinuities; see figure 7.3.

We compare the five different choices of prolongation operators in the standard coarsening black box multigrid method using zebra alternating line Gauss-Seidel relaxation or incomplete x-line LU iteration for the smoother. The comparison is done for a variety of grid sizes ranging from \(9 \times 9\) to \(257 \times 257\).

The data in the tables list the number of \(V(1, 1)\) cycles needed for the \(l_2\) norm of the residual to be reduced by 6 orders of magnitude with an initial guess of zero. The next three entries are the first \(V\)-cycle, last \(V\)-cycle, and average convergence factors. If convergence was not reached in 50 \(V\)-cycles then an * appears, and the convergence factor is given based on only 50 \(V\)-cycles.

Results for the method using an extension of Schaffer's idea for the grid transfer coefficients using alternating zebra Gauss-Seidel relaxation are given in table 7.14. The method exhibits good convergence factors for all grid sizes, and in addition the convergence factors grow very slowly as the grid size increases.

Results for the method using the grid transfer operators based on the form \(\sigma L/L\), from section 3.5.2, using alternating zebra Gauss-Seidel relaxation are given in
Figure 7.3. Domain $\Omega$ for problem 8.1; N and M stand for Neumann and Mixed boundary conditions respectively.

Table 7.14. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $ (n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>6.93E-5</td>
<td>5.82E-4</td>
<td>2.01E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>7.89E-3</td>
<td>1.42E-2</td>
<td>1.23E-2</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>3.03E-2</td>
<td>2.65E-2</td>
<td>2.74E-2</td>
</tr>
<tr>
<td>65</td>
<td>4</td>
<td>4.01E-2</td>
<td>2.43E-2</td>
<td>2.76E-2</td>
</tr>
<tr>
<td>129</td>
<td>4</td>
<td>4.34E-2</td>
<td>2.12E-2</td>
<td>2.49E-2</td>
</tr>
<tr>
<td>257</td>
<td>4</td>
<td>5.36E-2</td>
<td>2.94E-2</td>
<td>2.98E-2</td>
</tr>
</tbody>
</table>
Table 7.15. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators of the form $\sigma L/L$. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>2.50E-4</td>
<td>1.78E-3</td>
<td>6.67E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.17E-2</td>
<td>2.03E-2</td>
<td>1.77E-2</td>
</tr>
<tr>
<td>33</td>
<td>7</td>
<td>1.03E-1</td>
<td>1.29E-1</td>
<td>1.21E-2</td>
</tr>
<tr>
<td>65</td>
<td>16</td>
<td>3.85E-1</td>
<td>4.03E-1</td>
<td>3.99E-1</td>
</tr>
<tr>
<td>129</td>
<td>33</td>
<td>7.54E-1</td>
<td>6.52E-1</td>
<td>6.53E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>8.22E-1</td>
<td>8.15E-1</td>
<td>8.15E-1</td>
</tr>
</tbody>
</table>
Table 7.16. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.5.3. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>2.56E-4</td>
<td>1.82E-3</td>
<td>6.84E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.43E-2</td>
<td>2.30E-2</td>
<td>2.04E-2</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>4.96E-2</td>
<td>4.31E-2</td>
<td>4.43E-2</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>7.85E-2</td>
<td>5.69E-2</td>
<td>6.07E-2</td>
</tr>
<tr>
<td>257</td>
<td>6</td>
<td>1.01E-1</td>
<td>6.96E-2</td>
<td>7.41E-2</td>
</tr>
</tbody>
</table>

table 7.15. This method does not seem to be very attractive for this type of problem. The convergence factors grow quickly and approach one as the grid size increases.

The method using the grid transfer operators based on the form $\sigma L/L$–hybrid, from section 3.5.3, using alternating zebra Gauss–Seidel relaxation are given in table 7.16. The method seems to be attractive for this type of problem. The convergence factor, for the first $V$-cycle, grows as a function of problem size, but the convergence factor for subsequent $V$-cycles settles down to about 0.07.

The method using the grid transfer operators based on the form $\sigma L/\sigma L$, from section 3.5.1, using alternating zebra Gauss–Seidel relaxation are given in table 7.17. This method is almost identical to the last method, $\sigma L/L$–hybrid, as it should be for diffusion problems. The two methods differ only slightly in the grid transfer operators when the switch in the denominator is used.

The method using the grid transfer operator, $L/L$ form, from section 3.4 is given in table 7.18. This method does not perform very well at all. The method does not employ the use of the denominator switch in the grid transfer operators.

Tables 7.19 through 7.23 are the same as the previous tables except that now
Table 7.17. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.5.1. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>2.56E-4</td>
<td>1.83E-3</td>
<td>6.84E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.44E-2</td>
<td>2.30E-2</td>
<td>2.04E-2</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>4.69E-2</td>
<td>4.31E-2</td>
<td>4.43E-2</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>7.85E-2</td>
<td>5.69E-2</td>
<td>6.07E-2</td>
</tr>
<tr>
<td>257</td>
<td>6</td>
<td>1.01E-1</td>
<td>6.96E-2</td>
<td>7.41E-2</td>
</tr>
</tbody>
</table>

Table 7.18. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.4. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>2.57E-4</td>
<td>1.83E-3</td>
<td>6.85E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.43E-2</td>
<td>2.30E-2</td>
<td>2.04E-2</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>1.46E-2</td>
<td>1.72E-2</td>
<td>1.44E-2</td>
</tr>
<tr>
<td>65</td>
<td>14</td>
<td>4.90E-1</td>
<td>3.58E-1</td>
<td>3.66E-1</td>
</tr>
<tr>
<td>129</td>
<td>*</td>
<td>1.11E+0</td>
<td>7.75E-1</td>
<td>7.81E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>1.36E+0</td>
<td>9.38E-1</td>
<td>9.45E-1</td>
</tr>
</tbody>
</table>
Table 7.19. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of \( V(1,1) \) cycles using incomplete x–line LU iteration by lines in \( x \) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size ( n \times n )</th>
<th>Iterations ( V(1,1) )</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>3.40E-9</td>
<td>3.40E-9</td>
<td>3.40E-9</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>2.04E-5</td>
<td>3.05E-4</td>
<td>7.90E-5</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>2.12E-3</td>
<td>4.70E-3</td>
<td>3.57E-3</td>
</tr>
<tr>
<td>65</td>
<td>4</td>
<td>1.30E-2</td>
<td>9.70E-3</td>
<td>1.04E-2</td>
</tr>
<tr>
<td>129</td>
<td>4</td>
<td>2.36E-2</td>
<td>1.13E-2</td>
<td>1.34E-2</td>
</tr>
<tr>
<td>257</td>
<td>4</td>
<td>3.11E-2</td>
<td>2.01E-2</td>
<td>1.68E-2</td>
</tr>
</tbody>
</table>

Table 7.20. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.5.2. Various grid sizes versus the number of \( V(1,1) \) cycles using incomplete x–line LU iteration by lines in \( x \) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size ( n \times n )</th>
<th>Iterations ( V(1,1) )</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>5.52E-9</td>
<td>5.52E-9</td>
<td>5.52E-9</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>1.62E-5</td>
<td>2.07E-4</td>
<td>5.80E-5</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>3.45E-3</td>
<td>1.34E-2</td>
<td>8.22E-3</td>
</tr>
<tr>
<td>65</td>
<td>7</td>
<td>7.44E-2</td>
<td>1.54E-1</td>
<td>1.38E-1</td>
</tr>
<tr>
<td>129</td>
<td>17</td>
<td>2.75E-1</td>
<td>4.42E-1</td>
<td>4.29E-1</td>
</tr>
<tr>
<td>257</td>
<td>36</td>
<td>3.65E-1</td>
<td>6.91E-1</td>
<td>6.78E-1</td>
</tr>
</tbody>
</table>
Table 7.21. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.5.3. Various grid sizes versus the number of $V(1,1)$ cycles using incomplete $x$-line LU iteration by lines in $x$ for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>6.35E-9</td>
<td>6.35E-9</td>
<td>6.35E-9</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>1.90E-5</td>
<td>3.12E-4</td>
<td>7.70E-5</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>1.35E-3</td>
<td>3.65E-3</td>
<td>2.60E-3</td>
</tr>
<tr>
<td>65</td>
<td>3</td>
<td>7.34E-3</td>
<td>8.43E-3</td>
<td>8.00E-3</td>
</tr>
<tr>
<td>129</td>
<td>4</td>
<td>1.36E-2</td>
<td>1.26E-2</td>
<td>1.28E-2</td>
</tr>
<tr>
<td>257</td>
<td>4</td>
<td>1.69E-2</td>
<td>2.04E-2</td>
<td>1.69E-2</td>
</tr>
</tbody>
</table>

Table 7.22. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.5.1. Various grid sizes versus the number of $V(1,1)$ cycles using incomplete $x$-line LU iteration by lines in $x$ for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>6.33E-9</td>
<td>6.33E-9</td>
<td>6.33E-9</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>1.90E-5</td>
<td>3.12E-4</td>
<td>7.70E-5</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>1.35E-3</td>
<td>3.65E-3</td>
<td>2.60E-3</td>
</tr>
<tr>
<td>65</td>
<td>3</td>
<td>7.34E-3</td>
<td>8.43E-3</td>
<td>8.00E-3</td>
</tr>
<tr>
<td>129</td>
<td>4</td>
<td>1.36E-2</td>
<td>1.26E-2</td>
<td>1.28E-2</td>
</tr>
<tr>
<td>257</td>
<td>4</td>
<td>1.69E-2</td>
<td>2.04E-2</td>
<td>1.69E-2</td>
</tr>
</tbody>
</table>
Table 7.23. **Problem 8: Helmholtz Equation**, Standard coarsening with grid transfer operators from section 3.4. Various grid sizes versus the number of $V(1,1)$ cycles using incomplete x–line LU iteration by lines in x for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>6.33E-9</td>
<td>6.33E-9</td>
<td>6.33E-9</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>1.90E-5</td>
<td>3.13E-4</td>
<td>7.71E-5</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>1.21E-3</td>
<td>3.14E-3</td>
<td>2.25E-3</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>4.55E-2</td>
<td>6.36E-2</td>
<td>5.94E-2</td>
</tr>
<tr>
<td>129</td>
<td>18</td>
<td>4.24E-1</td>
<td>4.62E-1</td>
<td>4.59E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>8.02E-1</td>
<td>8.18E-1</td>
<td>8.18E-1</td>
</tr>
</tbody>
</table>
the smoother is an incomplete x-line LU iteration instead of the alternating zebra line Gauss–Seidel relaxation that was used before. The same observations hold as before, except that now the convergence factors are a bit smaller due to the fact that ILLU makes a better smoother than alternating line Gauss–Seidel.
7.6 Numerical Results for Test Problem 9

Problem 9 is a convection–diffusion problem, which can be found in [24], [66], [77]. The problem is defined as

\[-\varepsilon \Delta u + a(x, y) \frac{\partial u}{\partial x} + b(x, y) \frac{\partial u}{\partial y} = 0 \quad \text{on } \Omega = (0, 1) \times (0, 1) \quad (7.5)\]

\[u(x, y) = \sin(\pi x) + \sin(\pi y) + \sin(13\pi y) + \sin(13\pi y) \quad \text{on } \partial \Omega, \quad (7.6)\]

where

\[a(x, y) = (2x - 1)(1 - x^2), \]

\[b(x, y) = 2xy(y - 1)\]

and \(\varepsilon = 10^{-5}\); see figure 7.4.

Five choices of prolongation operators for the standard coarsening black box multigrid method using zebra alternating line Gauss–Seidel relaxation or incomplete x-line LU iteration for the smoother are presented. The comparison is done for a variety of grid sizes ranging from \(9 \times 9\) to \(257 \times 257\).

The results for the convection–diffusion equation using zebra alternating line Gauss–Seidel relaxation for the smoother are given in tables 7.24, 7.25, 7.26, 7.27, and 7.28.
Table 7.24. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>8.78E-3</td>
<td>8.58E-6</td>
<td>2.75E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.61E-2</td>
<td>1.57E-5</td>
<td>3.05E-3</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>2.78E-2</td>
<td>1.57E-2</td>
<td>3.04E-2</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>3.98E-2</td>
<td>6.03E-2</td>
<td>6.06E-2</td>
</tr>
<tr>
<td>129</td>
<td>6</td>
<td>5.67E-2</td>
<td>1.09E-1</td>
<td>9.08E-2</td>
</tr>
<tr>
<td>257</td>
<td>7</td>
<td>6.88E-2</td>
<td>1.19E-1</td>
<td>1.13E-1</td>
</tr>
</tbody>
</table>

Table 7.25. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.2. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>2.11E-2</td>
<td>5.63E-3</td>
<td>7.89E-3</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>3.18E-2</td>
<td>4.34E-3</td>
<td>2.48E-3</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>6.51E-2</td>
<td>2.38E-2</td>
<td>5.38E-2</td>
</tr>
<tr>
<td>65</td>
<td>9</td>
<td>1.49E+0</td>
<td>1.28E-1</td>
<td>1.88E-1</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>4.15E+1</td>
<td>1.13E+1</td>
<td>7.01E+0</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>2.91E+3</td>
<td>1.24E+3</td>
<td>1.24E+3</td>
</tr>
</tbody>
</table>
Table 7.26. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.3. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ ((n \times n))</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>5.74E-3</td>
<td>1.86E-5</td>
<td>3.27E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.54E-2</td>
<td>1.77E-5</td>
<td>3.24E-3</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>2.65E-2</td>
<td>1.42E-2</td>
<td>2.99E-2</td>
</tr>
<tr>
<td>257</td>
<td>8</td>
<td>1.29E+0</td>
<td>8.34E-2</td>
<td>1.35E-1</td>
</tr>
</tbody>
</table>

Table 7.27. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.1. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ ((n \times n))</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>2.24E-2</td>
<td>8.12E-3</td>
<td>8.96E-3</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>2.63E-2</td>
<td>2.84E-3</td>
<td>2.18E-2</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>4.90E-1</td>
<td>4.40E-2</td>
<td>6.68E-2</td>
</tr>
<tr>
<td>65</td>
<td>div</td>
<td>1.86E+1</td>
<td>1.53E+1</td>
<td>1.53E+1</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>1.67E+4</td>
<td>2.88E+3</td>
<td>2.88E+3</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>1.51E+9</td>
<td>9.02E+8</td>
<td>9.02E+8</td>
</tr>
</tbody>
</table>
Table 7.28. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.4. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>3.89E-2</td>
<td>2.39E-5</td>
<td>3.16E-3</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>5.35E-2</td>
<td>1.11E-2</td>
<td>5.59E-2</td>
</tr>
<tr>
<td>33</td>
<td>7</td>
<td>7.29E-2</td>
<td>6.07E-2</td>
<td>1.24E-1</td>
</tr>
<tr>
<td>65</td>
<td>10</td>
<td>1.65E-1</td>
<td>1.54E-1</td>
<td>2.10E-1</td>
</tr>
<tr>
<td>129</td>
<td>12</td>
<td>1.69E-1</td>
<td>1.92E-1</td>
<td>2.95E-1</td>
</tr>
<tr>
<td>257</td>
<td>18</td>
<td>1.90E-1</td>
<td>2.27E-1</td>
<td>4.31E-1</td>
</tr>
</tbody>
</table>

Table 7.29. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on extension of Schaffer's idea. Various grid sizes versus the number of $V(1, 1)$ cycles using incomplete line LU iteration by lines in x for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>8.41E-14</td>
<td>8.41E-14</td>
<td>8.41E-14</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>6.91E-12</td>
<td>6.91E-12</td>
<td>6.91E-12</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>6.29E-10</td>
<td>6.29E-10</td>
<td>6.29E-10</td>
</tr>
<tr>
<td>65</td>
<td>1</td>
<td>4.63E-8</td>
<td>4.63E-8</td>
<td>4.63E-8</td>
</tr>
<tr>
<td>129</td>
<td>2</td>
<td>1.95E-6</td>
<td>5.21E-5</td>
<td>1.01E-5</td>
</tr>
<tr>
<td>257</td>
<td>2</td>
<td>4.88E-5</td>
<td>1.09E-3</td>
<td>2.31E-4</td>
</tr>
</tbody>
</table>
Table 7.30. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.2. Various grid sizes versus the number of $V(1, 1)$ cycles using incomplete line LU iteration by lines in $x$ for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>1.19E-13</td>
<td>1.19E-13</td>
<td>1.19E-13</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>9.36E-12</td>
<td>9.36E-12</td>
<td>9.36E-12</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>8.28E-10</td>
<td>8.28E-10</td>
<td>8.28E-10</td>
</tr>
<tr>
<td>65</td>
<td>1</td>
<td>6.60E-8</td>
<td>6.60E-8</td>
<td>6.60E-8</td>
</tr>
<tr>
<td>129</td>
<td>2</td>
<td>3.33E-6</td>
<td>1.09E-4</td>
<td>1.91E-5</td>
</tr>
<tr>
<td>257</td>
<td>2</td>
<td>2.33E-3</td>
<td>1.96E-4</td>
<td>6.74E-4</td>
</tr>
</tbody>
</table>

Table 7.31. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.3. Various grid sizes versus the number of $V(1, 1)$ cycles using incomplete line LU iteration by lines in $x$ for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>8.45E-14</td>
<td>8.45E-14</td>
<td>8.45E-14</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>6.62E-12</td>
<td>6.62E-12</td>
<td>6.62E-12</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>5.74E-10</td>
<td>5.74E-10</td>
<td>5.74E-10</td>
</tr>
<tr>
<td>65</td>
<td>1</td>
<td>4.11E-8</td>
<td>4.11E-8</td>
<td>4.11E-8</td>
</tr>
<tr>
<td>129</td>
<td>2</td>
<td>1.71E-6</td>
<td>3.65E-5</td>
<td>7.90E-6</td>
</tr>
<tr>
<td>257</td>
<td>2</td>
<td>4.21E-5</td>
<td>8.20E-4</td>
<td>1.86E-4</td>
</tr>
</tbody>
</table>
Table 7.32. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.1. Various grid sizes versus the number of $V(1, 1)$ cycles using incomplete line LU iteration by lines in $x$ for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9$</td>
<td>$1$</td>
<td>$1.45E-13$</td>
<td>$1.45E-13$</td>
<td>$1.45E-13$</td>
</tr>
<tr>
<td>$17$</td>
<td>$1$</td>
<td>$8.56E-12$</td>
<td>$8.56E-12$</td>
<td>$8.56E-12$</td>
</tr>
<tr>
<td>$33$</td>
<td>$1$</td>
<td>$5.11E-10$</td>
<td>$5.11E-10$</td>
<td>$5.11E-10$</td>
</tr>
<tr>
<td>$65$</td>
<td>$1$</td>
<td>$2.54E-8$</td>
<td>$2.54E-8$</td>
<td>$2.54E-8$</td>
</tr>
<tr>
<td>$129$</td>
<td>$2$</td>
<td>$1.06E-6$</td>
<td>$1.81E-5$</td>
<td>$4.38E-6$</td>
</tr>
<tr>
<td>$257$</td>
<td>*</td>
<td>$6.84E+0$</td>
<td>$8.25E-1$</td>
<td>$8.60E-1$</td>
</tr>
</tbody>
</table>

Table 7.33. **Problem 9: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.4. Various grid sizes versus the number of $V(1, 1)$ cycles using incomplete line LU iteration by lines in $x$ for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9$</td>
<td>$1$</td>
<td>$2.08E-13$</td>
<td>$2.08E-13$</td>
<td>$2.08E-13$</td>
</tr>
<tr>
<td>$17$</td>
<td>$1$</td>
<td>$1.23E-11$</td>
<td>$1.23E-11$</td>
<td>$1.23E-11$</td>
</tr>
<tr>
<td>$33$</td>
<td>$1$</td>
<td>$5.90E-10$</td>
<td>$5.90E-10$</td>
<td>$5.90E-10$</td>
</tr>
<tr>
<td>$65$</td>
<td>$1$</td>
<td>$3.22E-8$</td>
<td>$3.22E-8$</td>
<td>$3.22E-8$</td>
</tr>
<tr>
<td>$129$</td>
<td>$2$</td>
<td>$1.75E-6$</td>
<td>$1.02E-4$</td>
<td>$1.34E-5$</td>
</tr>
<tr>
<td>$257$</td>
<td>$2$</td>
<td>$6.76E-5$</td>
<td>$1.14E-3$</td>
<td>$2.78E-4$</td>
</tr>
</tbody>
</table>
Table 7.34. **Problem 9: Convection–Diffusion Equation.** Standard coarsening with grid transfer operators based on section 3.5.1. Various grid sizes versus the number of $V(1,1)$ cycles using 4-direction point Gauss–Seidel relaxation for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>3.09E-6</td>
<td>1.91E-6</td>
<td>2.43E-6</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>2.34E-6</td>
<td>4.04E-6</td>
<td>3.08E-6</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td>1.74E-6</td>
<td>5.02E-6</td>
<td>2.95E-6</td>
</tr>
<tr>
<td>65</td>
<td>2</td>
<td>5.05E-6</td>
<td>1.38E-5</td>
<td>8.35E-6</td>
</tr>
<tr>
<td>129</td>
<td>2</td>
<td>6.15E-5</td>
<td>3.05E-4</td>
<td>1.37E-4</td>
</tr>
<tr>
<td>257</td>
<td>4</td>
<td>5.35E+1</td>
<td>7.19E-5</td>
<td>1.00E-2</td>
</tr>
</tbody>
</table>

From the tables we see that the methods that use extended Schaffer’s ideas perform the best closely followed by the hybrid method from section 3.5.3. Notice that changing the smoother from the alternating line Gauss–Seidel to the incomplete line LU iteration greatly improves the convergence factors. When alternating line Gauss–Seidel is used, we see that both the grid transfer operator methods from sections 3.5.2 and 3.5.1, respectively, diverge for large grid sizes, but that they are both convergent when the incomplete line LU iteration is used for the smoother.

We also see that using 4-direction point Gauss–Seidel relaxation for the smoother with the grid transfer operator method from section 3.5.1 ($\sigma L/\sigma L$) gives results that are comparable to those using the ILLU smoother. However, as can be seen by the initial convergence factor for the $257 \times 257$ grid, that problems have crept in for large grid sizes. The convergence factor is only greater than one for the initial iteration and then for subsequent iterations the convergence rate drops off very quickly. However, for larger fine grids the convergence factor oscillates back and fourth from around 50 to around 0.37, and the methods are divergent.
Figure 7.5: Domain $\Omega$ for problem 10; D stands for Dirichlet boundary condition.

### 7.7 Numerical Results for Test Problem 10

Problem 10 is a convection–diffusion problem, which can be found in the literature; see [24], [66], [77]. The problem is defined by

$$-\varepsilon \Delta u + a(x, y) \frac{\partial u}{\partial x} + b(x, y) \frac{\partial u}{\partial y} = 0 \quad \text{on} \quad \Omega = (0, 1) \times (0, 1) \tag{7.7}$$

$$u(x, y) = \sin(\pi x) + \sin(\pi y) + \sin(13\pi y) + \sin(13\pi y) \quad \text{on} \partial\Omega, \tag{7.8}$$

where

$$a(x, y) = 4x(x - 1)(1 - 2y),$$

$$b(x, y) = -4y(y - 1)(1 - 2x),$$

and $\varepsilon = 10^{-5}$; see figure 7.5.

This problem is a re-entrant flow problem; such problems are among the most difficult convection-diffusion problems. None of our methods are adequate for solving these types of problems and several are not even convergent except for small grid sizes. However, using ILLU for the smoother does help many of the methods become convergent, even if the convergence factor is rather poor.
There are several reasons why our methods do not work properly on these types of problems. One is that the smoothers are just not adequate. Another is that all of our grid transfer operators, that we have considered, are close to violating the order of interpolation rule \[15\] \[41\] \[45\] \[85\]. The rule states that

\[ m_r + m_p \geq m_I \]  

(7.9)

where \( m_r \) and \( m_p \) are the orders of interpolation for the restriction and prolongation operators, respectively, and \( m_I \) is the order of the grid equation operator. In our case we have \( m_r = 1 \), \( m_p = 2 \), and \( m_I = 3 \) for equality. Numerically the rule is violated for some of the grid equations due to the effects of computer arithmetic. Another way to look at the trouble is that the grid transfer operators fail to map all the high frequency errors into the range of the smoother.

De Zeeuw’s MGD9V \[24\] code was designed for these types of convection-diffusion problems, and his interpolation operator does map the error into the range of the ILLU smoother; see table 7.43. Although MGD9V does become divergent for large grids (> 160 x 160), it does perform much better than any of the other methods for the smaller grid sizes.
Table 7.35. **Problem 10: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in $x$ followed by lines in $y$) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>14</td>
<td>1.06E-1</td>
<td>3.96E-1</td>
<td>3.58E-1</td>
</tr>
<tr>
<td>17</td>
<td>25</td>
<td>1.38E-1</td>
<td>6.06E-1</td>
<td>5.68E-1</td>
</tr>
<tr>
<td>33</td>
<td>*</td>
<td>1.56E-1</td>
<td>7.96E-1</td>
<td>7.65E-1</td>
</tr>
<tr>
<td>65</td>
<td>*</td>
<td>1.67E-1</td>
<td>9.11E-1</td>
<td>8.70E-1</td>
</tr>
<tr>
<td>129</td>
<td>*</td>
<td>2.05E-1</td>
<td>9.63E-1</td>
<td>9.14E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>2.66E-1</td>
<td>9.83E-1</td>
<td>9.31E-1</td>
</tr>
</tbody>
</table>

Table 7.36. **Problem 10: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.2. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in $x$ followed by lines in $y$) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>15</td>
<td>1.20E-1</td>
<td>4.22E-1</td>
<td>3.89E-1</td>
</tr>
<tr>
<td>17</td>
<td>36</td>
<td>2.19E-1</td>
<td>7.03E-1</td>
<td>6.79E-1</td>
</tr>
<tr>
<td>33</td>
<td>*</td>
<td>2.36E-1</td>
<td>8.42E-1</td>
<td>8.36E-1</td>
</tr>
<tr>
<td>65</td>
<td>div</td>
<td>4.35E-1</td>
<td>1.12E+0</td>
<td>1.05E+0</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>1.30E+1</td>
<td>4.60E+1</td>
<td>4.60E+1</td>
</tr>
</tbody>
</table>
Table 7.37. **Problem 10: Convection-Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.3. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss-Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>5.79E-2</td>
<td>2.29E-1</td>
<td>1.95E-1</td>
</tr>
<tr>
<td>17</td>
<td>22</td>
<td>1.30E-1</td>
<td>5.65E-1</td>
<td>5.25E-1</td>
</tr>
<tr>
<td>33</td>
<td>*</td>
<td>1.71E-1</td>
<td>8.20E-1</td>
<td>7.88E-1</td>
</tr>
<tr>
<td>65</td>
<td>*</td>
<td>1.89E-1</td>
<td>9.34E-1</td>
<td>8.92E-1</td>
</tr>
<tr>
<td>129</td>
<td>*</td>
<td>2.57E-1</td>
<td>9.76E-1</td>
<td>9.27E-1</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>4.00E-1</td>
<td>1.98E+0</td>
<td>1.98E+0</td>
</tr>
</tbody>
</table>

Table 7.38. **Problem 10: Convection-Diffusion Equation**, Standard coarsening with grid transfer operators based section 3.5.1. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss-Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>5</td>
<td>4.64E-2</td>
<td>6.76E-2</td>
<td>6.19E-2</td>
</tr>
<tr>
<td>17</td>
<td>7</td>
<td>2.70E-1</td>
<td>8.72E-2</td>
<td>1.20E-1</td>
</tr>
<tr>
<td>33</td>
<td>div</td>
<td>5.66E+0</td>
<td>9.93E+1</td>
<td>9.93E+1</td>
</tr>
<tr>
<td>65</td>
<td>div</td>
<td>2.66E+2</td>
<td>2.15E+3</td>
<td>2.15E+3</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>1.04E+5</td>
<td>1.77E+8</td>
<td>1.77E+8</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>2.04E+6</td>
<td>1.08E+12</td>
<td>1.08E+12</td>
</tr>
</tbody>
</table>
Table 7.39. **Problem 10: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based section 3.4. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>8.88E-2</td>
<td>2.07E-1</td>
<td>1.89E-1</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>1.32E-1</td>
<td>5.31E-1</td>
<td>4.90E-1</td>
</tr>
<tr>
<td>33</td>
<td>*</td>
<td>1.76E-1</td>
<td>8.67E-1</td>
<td>8.31E-1</td>
</tr>
<tr>
<td>65</td>
<td>*</td>
<td>2.46E-1</td>
<td>9.73E-1</td>
<td>9.25E-1</td>
</tr>
<tr>
<td>129</td>
<td>*</td>
<td>3.58E-1</td>
<td>9.92E-1</td>
<td>9.36E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>6.20E-1</td>
<td>9.90E-1</td>
<td>9.41E-1</td>
</tr>
</tbody>
</table>

Table 7.40. **Problem 10: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1, 1)$ cycles using ILLU for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>5.58E-2</td>
<td>2.26E-1</td>
<td>1.93E-1</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>9.22E-2</td>
<td>4.22E-1</td>
<td>3.81E-1</td>
</tr>
<tr>
<td>33</td>
<td>26</td>
<td>1.10E-1</td>
<td>6.28E-1</td>
<td>5.86E-1</td>
</tr>
<tr>
<td>65</td>
<td>49</td>
<td>1.13E-1</td>
<td>7.85E-1</td>
<td>7.53E-1</td>
</tr>
<tr>
<td>129</td>
<td>*</td>
<td>1.09E-1</td>
<td>8.77E-1</td>
<td>8.38E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>8.80E-2</td>
<td>9.27E-1</td>
<td>8.75E-1</td>
</tr>
</tbody>
</table>

Table 7.41. **Problem 10: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.3. Various grid sizes versus the number of $V(1, 1)$ cycles using ILLU for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>7</td>
<td>3.98E-2</td>
<td>1.56E-1</td>
<td>1.28E-1</td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>8.19E-2</td>
<td>3.80E-1</td>
<td>3.38E-1</td>
</tr>
<tr>
<td>33</td>
<td>24</td>
<td>1.04E-1</td>
<td>6.07E-1</td>
<td>5.62E-1</td>
</tr>
<tr>
<td>65</td>
<td>47</td>
<td>1.10E-1</td>
<td>7.76E-1</td>
<td>7.42E-1</td>
</tr>
<tr>
<td>129</td>
<td>*</td>
<td>1.09E-1</td>
<td>8.81E-1</td>
<td>8.40E-1</td>
</tr>
<tr>
<td>257</td>
<td>*</td>
<td>8.53E-1</td>
<td>9.46E-1</td>
<td>8.92E-1</td>
</tr>
</tbody>
</table>
Table 7.42. **Problem 10: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based section 3.5.1. Various grid sizes versus the number of \(V(1, 1)\) cycles using ILLU for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size (n \times n)</th>
<th>Iterations (V(1, 1))</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>9.42E-3</td>
<td>3.87E-2</td>
<td>2.70E-2</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>1.31E-2</td>
<td>6.51E-2</td>
<td>4.69E-2</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>1.51E-2</td>
<td>7.80E-2</td>
<td>5.56E-2</td>
</tr>
<tr>
<td>65</td>
<td>9</td>
<td>3.74E-2</td>
<td>2.51E-1</td>
<td>2.01E-1</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>6.88E+4</td>
<td>2.04E+5</td>
<td>1.90E+5</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>1.05E+5</td>
<td>2.88E+5</td>
<td>2.70E+5</td>
</tr>
</tbody>
</table>

Table 7.43. **Problem 10: Convection–Diffusion Equation**, for De Zeeuw’s MGD9V. Various grid sizes versus the number of \(V(0, 1)\) cycles using ILLU for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size (n \times n)</th>
<th>Iterations (V(1, 1))</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>5</td>
<td>7.53E-2</td>
<td>1.05E-1</td>
<td>8.21E-2</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>1.54E-2</td>
<td>1.94E-1</td>
<td>9.31E-2</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>9.45E-3</td>
<td>2.84E-1</td>
<td>1.52E-1</td>
</tr>
<tr>
<td>65</td>
<td>8</td>
<td>1.36E-2</td>
<td>4.42E-1</td>
<td>2.83E-1</td>
</tr>
<tr>
<td>129</td>
<td>11</td>
<td>1.75E-2</td>
<td>5.84E-1</td>
<td>4.27E-1</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>1.27E+0</td>
<td>2.90E+0</td>
<td>2.49E+0</td>
</tr>
</tbody>
</table>
Numerical Results for Test Problem 11

Problem 11 is a convection–diffusion problem, which can be found in the literature; see [24], [66], [77]. The problem is defined by

\[-\varepsilon \Delta u + a(x, y) \frac{\partial u}{\partial x} + b(x, y) \frac{\partial u}{\partial y} = 0 \quad \text{on} \quad \Omega = (0, 1) \times (0, 1) \quad (7.10)\]

\[u(x, y) = \sin(\pi x) + \sin(\pi y) + \sin(13\pi y) + \sin(13\pi y) \quad \text{on} \partial \Omega, \quad (7.11)\]

and

\[a(x, y) = \begin{cases} 
(2y - 1)(1 - \tilde{x}^2) & \text{if } \tilde{x} > 0 \\
(2y - 1) & \text{if } \tilde{x} \leq 0 
\end{cases}, \]

\[b(x, y) = \begin{cases} 
2\tilde{x}y(y - 1) & \text{if } \tilde{x} > 0 \\
0 & \text{if } \tilde{x} \leq 0 
\end{cases}, \]

where \(\tilde{x} = 1.2x - 0.2\) and \(\varepsilon = 10^{-5}\); see figure 7.6. The difference between problem 11 and problem 9 is that we now have a stagnation line rather than a stagnation point emanating from the boundary.

The results of the numerical experiments for this problem are similar to those for problem 9. The method using Schaffer’s ideas for the grid transfer operators is the
Table 7.44. **Problem 11: Convection–Diffusion Equation**. Standard coarsening with grid transfer operators based on extension of Schaffer's idea. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>8.55E-3</td>
<td>6.31E-6</td>
<td>2.32E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.46E-2</td>
<td>9.62E-6</td>
<td>2.34E-3</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>2.47E-2</td>
<td>1.29E-2</td>
<td>2.81E-2</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>3.70E-2</td>
<td>5.46E-2</td>
<td>5.64E-2</td>
</tr>
<tr>
<td>129</td>
<td>6</td>
<td>5.57E-2</td>
<td>9.67E-2</td>
<td>8.61E-2</td>
</tr>
<tr>
<td>257</td>
<td>7</td>
<td>7.27E-2</td>
<td>1.06E-1</td>
<td>1.08E-1</td>
</tr>
</tbody>
</table>

best, followed closely by the hybrid method from section 3.5.3. We also see that grid transfer operators generated using the method in section 3.4 are also quite adequate.
Table 7.45. **Problem 11: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.2. Various grid sizes versus the number of \( V(1, 1) \) cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size ( n \times n )</th>
<th>Iterations ( V(1, 1) )</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>1.95E-2</td>
<td>4.11E-3</td>
<td>6.62E-3</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>3.16E-2</td>
<td>2.38E-3</td>
<td>1.95E-2</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>4.44E-2</td>
<td>5.39E-2</td>
<td>5.62E-2</td>
</tr>
<tr>
<td>65</td>
<td>11</td>
<td>8.02E-1</td>
<td>2.20E-1</td>
<td>2.63E-1</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>8.56E+0</td>
<td>7.14E+0</td>
<td>7.14E+0</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>1.26E+3</td>
<td>3.41E+2</td>
<td>3.41E+2</td>
</tr>
</tbody>
</table>

Table 7.46. **Problem 11: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.3. Various grid sizes versus the number of \( V(1, 1) \) cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size ( n \times n )</th>
<th>Iterations ( V(1, 1) )</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>5.81E-3</td>
<td>9.65E-5</td>
<td>7.49E-4</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.49E-2</td>
<td>1.20E-5</td>
<td>2.50E-3</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>2.30E-2</td>
<td>1.12E-2</td>
<td>2.67E-2</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>4.61E-2</td>
<td>5.38E-2</td>
<td>5.65E-2</td>
</tr>
<tr>
<td>129</td>
<td>6</td>
<td>6.87E-2</td>
<td>8.64E-2</td>
<td>9.10E-2</td>
</tr>
<tr>
<td>257</td>
<td>10</td>
<td>3.56E-1</td>
<td>7.31E-2</td>
<td>2.18E-1</td>
</tr>
</tbody>
</table>
Table 7.47. **Problem 11: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.5.1. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>1.97E-2</td>
<td>6.09E-3</td>
<td>8.07E-3</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>2.47E-2</td>
<td>1.26E-3</td>
<td>1.49E-2</td>
</tr>
<tr>
<td>33</td>
<td>7</td>
<td>1.11E+0</td>
<td>6.38E-2</td>
<td>1.16E-1</td>
</tr>
<tr>
<td>65</td>
<td>div</td>
<td>3.57E+0</td>
<td>7.94E+0</td>
<td>7.94E+0</td>
</tr>
<tr>
<td>129</td>
<td>div</td>
<td>7.85E+5</td>
<td>9.99E+4</td>
<td>9.99E+4</td>
</tr>
<tr>
<td>257</td>
<td>div</td>
<td>9.96E+9</td>
<td>7.00E+9</td>
<td>7.00E+9</td>
</tr>
</tbody>
</table>

Table 7.48. **Problem 11: Convection–Diffusion Equation**, Standard coarsening with grid transfer operators based on section 3.4. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss–Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>3.73E-2</td>
<td>9.56E-5</td>
<td>3.78E-3</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>5.23E-2</td>
<td>6.26E-3</td>
<td>4.22E-2</td>
</tr>
<tr>
<td>33</td>
<td>7</td>
<td>6.07E-2</td>
<td>3.65E-2</td>
<td>1.05E-1</td>
</tr>
<tr>
<td>65</td>
<td>9</td>
<td>1.41E-1</td>
<td>1.69E-1</td>
<td>1.99E-1</td>
</tr>
<tr>
<td>129</td>
<td>11</td>
<td>1.53E-1</td>
<td>1.73E-1</td>
<td>2.75E-1</td>
</tr>
<tr>
<td>257</td>
<td>17</td>
<td>1.79E-1</td>
<td>2.13E-1</td>
<td>4.06E-1</td>
</tr>
</tbody>
</table>
7.9 Numerical Results for Test Problem 13

Problem 13 is an anisotropic and discontinuous problem defined as

\[- \nabla \cdot D \nabla u + cu = f \quad \text{on} \quad \Omega = (0, N_x) \times (0, N_y) \tag{7.12}\]

\[D \frac{\partial u}{\partial n} + \frac{1}{2} u = 0 \quad \text{on} \quad \partial \Omega \quad \text{at} \quad x = N_x \quad \text{or} \quad y = N_y \tag{7.13}\]

\[\frac{\partial u}{\partial n} = 0 \quad \text{on} \quad \partial \Omega \quad \text{at} \quad x = 0 \quad \text{or} \quad y = 0 \tag{7.14}\]

where

\[
\begin{align*}
D_1 &= .1 \\
D_2 &= 1 \\
c &= 1 \\
f &= 1 \\
D_1 &= 1000 \\
D_2 &= 1000
\end{align*}
\]

\[
\begin{align*}
\text{if} \quad x \in [0, 31], \quad y \in [0, 31]; \\
x \in (33, 64], \quad y \in (33, 64]
\end{align*}
\]

\[
\begin{align*}
\text{if} \quad x \in (31, 33], \quad y \in [0, 31]; \\
x \in [0, 31], \quad y \in [31, 33]; \\
x \in (33, 64], \quad y \in (31, 33]; \\
x \in (31, 33], \quad y \in (33, 64]
\end{align*}
\]
Table 7.49. Problem 13: Diffusion Equation, Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1, 1)$ cycles using zebra alternating line Gauss-Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>1.84E-2</td>
<td>2.32E-2</td>
<td>2.13E-2</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.67E-2</td>
<td>2.22E-2</td>
<td>1.95E-2</td>
</tr>
<tr>
<td>33</td>
<td>5</td>
<td>3.10E-2</td>
<td>6.57E-2</td>
<td>5.79E-2</td>
</tr>
<tr>
<td>65</td>
<td>6</td>
<td>1.29E-2</td>
<td>1.14E-1</td>
<td>7.52E-2</td>
</tr>
<tr>
<td>129</td>
<td>5</td>
<td>1.51E-2</td>
<td>1.19E-1</td>
<td>6.18E-2</td>
</tr>
<tr>
<td>257</td>
<td>6</td>
<td>2.47E-2</td>
<td>1.94E-1</td>
<td>8.72E-2</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
D_1 &= 1 \\
D_2 &= .1 \\
c &= 1 \\
f &= 1
\end{align*}
\]

\[
\begin{align*}
D_1 &= 100 \\
D_2 &= 100 \\
c &= 1 \\
f &= 0
\end{align*}
\]

and the domain is illustrated in figure 7.7.

For this problem we have chosen to report only the three most valuable grid transfer operators from sections 3.6, 3.5.3, and 3.5.1. All three give very good performance over the range of grid sizes tested.
Table 7.50. Problem 13: Diffusion Equation, Standard coarsening with grid transfer operators based section 3.5.3. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss-Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>2.17E-2</td>
<td>2.71E-2</td>
<td>2.56E-2</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.09E-2</td>
<td>2.33E-2</td>
<td>1.78E-2</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>7.35E-3</td>
<td>5.73E-2</td>
<td>2.38E-2</td>
</tr>
<tr>
<td>65</td>
<td>7</td>
<td>1.44E-2</td>
<td>1.80E-1</td>
<td>1.14E-1</td>
</tr>
<tr>
<td>129</td>
<td>8</td>
<td>2.63E-2</td>
<td>1.96E-1</td>
<td>1.47E-1</td>
</tr>
<tr>
<td>257</td>
<td>8</td>
<td>4.55E-2</td>
<td>2.40E-1</td>
<td>1.71E-1</td>
</tr>
</tbody>
</table>

Table 7.51. Problem 13: Diffusion Equation, Standard coarsening with grid transfer operators based section 3.5.1. Various grid sizes versus the number of $V(1,1)$ cycles using zebra alternating line Gauss-Seidel (lines in x followed by lines in y) for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>2.17E-2</td>
<td>2.71E-2</td>
<td>2.56E-2</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1.09E-2</td>
<td>2.33E-2</td>
<td>1.78E-2</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>7.35E-3</td>
<td>5.73E-2</td>
<td>2.38E-2</td>
</tr>
<tr>
<td>65</td>
<td>7</td>
<td>1.44E-2</td>
<td>1.80E-1</td>
<td>1.14E-1</td>
</tr>
<tr>
<td>129</td>
<td>8</td>
<td>2.63E-2</td>
<td>1.96E-1</td>
<td>1.47E-1</td>
</tr>
<tr>
<td>257</td>
<td>8</td>
<td>4.55E-2</td>
<td>2.40E-1</td>
<td>1.71E-1</td>
</tr>
</tbody>
</table>
Table 7.52. Problem 17: Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(1, 1)$ cycles using alternating zebra line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>18</td>
<td>6.35E+0</td>
<td>1.72E-1</td>
<td>4.38E-1</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>1.43E-1</td>
<td>3.09E-1</td>
<td>2.91E-1</td>
</tr>
<tr>
<td>33</td>
<td>12</td>
<td>1.18E-1</td>
<td>3.34E-1</td>
<td>3.07E-1</td>
</tr>
<tr>
<td>65</td>
<td>13</td>
<td>9.22E-2</td>
<td>3.54E-1</td>
<td>3.21E-1</td>
</tr>
<tr>
<td>129</td>
<td>13</td>
<td>7.81E-2</td>
<td>3.58E-1</td>
<td>3.22E-1</td>
</tr>
<tr>
<td>257</td>
<td>13</td>
<td>7.56E-2</td>
<td>3.59E-1</td>
<td>3.22E-1</td>
</tr>
</tbody>
</table>

7.10 Numerical Results for Test Problem 17

Problem 17 is a discontinuous staircase problem, which can be found in the literature; see [26], [77], [24]. The problem is defined by

\[
-\nabla \cdot D \nabla u = f \quad \text{on} \quad \Omega = (0, 16) \times (0, 16)
\]  

\[
D \frac{\partial u}{\partial n} + \frac{1}{2} u = 0 \quad \text{on} \quad \partial \Omega
\]

where

\[
D = 1, \quad f = 0 \quad (x, y) \text{ outside the shaded area}
\]

\[
D = 1000, \quad f = 1 \quad (x, y) \text{ inside the shaded area;}
\]

see figure 7.8.

In many real world applications boundaries are often curved, making the discretization hard to perform accurately on rectangular meshes. After discretization the curved boundary will look something like a staircase. Problems with staircase interfaces in the domain are not handled well by classical multigrid methods. In particular, multigrid methods which employ five point stencils on coarser grids are doomed to failure for staircase problems, since for sufficiently coarse grids, the five point stencil cannot resolve the staircase. The black box multigrid methods, however, can handle
Figure 7.8. Domain $\Omega$ for problem 17; N and M stand for Neumann and Mixed boundary conditions respectively.
Table 7.53. Problem 17: Standard coarsening with grid transfer operators based on original collapsing method. Various grid sizes versus the number of $V(1, 1)$ cycles using alternating zebra line Gauss–Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13</td>
<td>4.54E-1</td>
<td>2.97E-1</td>
<td>3.17E-1</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>6.34E-2</td>
<td>1.64E-1</td>
<td>1.44E-1</td>
</tr>
<tr>
<td>33</td>
<td>8</td>
<td>5.35E-2</td>
<td>1.62E-1</td>
<td>1.48E-1</td>
</tr>
<tr>
<td>65</td>
<td>8</td>
<td>4.92E-2</td>
<td>1.73E-1</td>
<td>1.54E-1</td>
</tr>
<tr>
<td>129</td>
<td>8</td>
<td>5.10E-2</td>
<td>1.85E-1</td>
<td>1.65E-1</td>
</tr>
<tr>
<td>257</td>
<td>8</td>
<td>5.64E-2</td>
<td>1.89E-1</td>
<td>1.70E-1</td>
</tr>
</tbody>
</table>

staircase interfaces rather well because they use operator induced grid transfer operators and the Galerkin coarse grid approximation to form the coarse grid operators; the nine point operators created in this way can resolve the staircase on coarser grids.
Table 7.54. Problem 17: Standard coarsening with grid transfer operators based on extension of Schaffer’s idea. Various grid sizes versus the number of $V(0,1)$ cycles using x-line ILLU for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>1.96E+0</td>
<td>1.08E-1</td>
<td>1.81E-1</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>1.85E-1</td>
<td>1.71E-1</td>
<td>1.73E-1</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>2.93E-2</td>
<td>8.96E-1</td>
<td>8.06E-2</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>2.03E-2</td>
<td>1.15E-1</td>
<td>6.13E-2</td>
</tr>
<tr>
<td>129</td>
<td>5</td>
<td>2.27E-2</td>
<td>5.07E-2</td>
<td>5.38E-2</td>
</tr>
<tr>
<td>257</td>
<td>7</td>
<td>2.89E-2</td>
<td>1.35E-1</td>
<td>1.19E-1</td>
</tr>
</tbody>
</table>

Table 7.55. Problem 17: Standard coarsening with grid transfer operators based on the hybrid collapsing method. Various grid sizes versus the number of $V(0,1)$ cycles using x-line ILLU for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>11</td>
<td>4.49E-1</td>
<td>2.68E-1</td>
<td>2.82E-1</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>2.91E-2</td>
<td>8.17E-2</td>
<td>6.85E-2</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>2.33E-2</td>
<td>8.21E-2</td>
<td>6.69E-2</td>
</tr>
<tr>
<td>65</td>
<td>6</td>
<td>1.90E-2</td>
<td>8.91E-2</td>
<td>6.96E-2</td>
</tr>
<tr>
<td>129</td>
<td>6</td>
<td>1.90E-2</td>
<td>1.09E-1</td>
<td>8.17E-2</td>
</tr>
<tr>
<td>257</td>
<td>6</td>
<td>1.98E-2</td>
<td>1.11E-1</td>
<td>8.44E-2</td>
</tr>
</tbody>
</table>

Table 7.56. Problem 17: Semi-coarsening code. Various grid sizes versus the number of $V(1,1)$ cycles using zebra x-line Gauss-Seidel for the smoother, first, last and average convergence factor.

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
<td>9.13E-1</td>
<td>2.66E-1</td>
<td>3.07E-1</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>2.14E-1</td>
<td>1.76E-1</td>
<td>1.83E-1</td>
</tr>
<tr>
<td>33</td>
<td>8</td>
<td>1.47E-1</td>
<td>1.61E-1</td>
<td>1.59E-1</td>
</tr>
<tr>
<td>65</td>
<td>8</td>
<td>1.13E-1</td>
<td>1.54E-1</td>
<td>1.48E-1</td>
</tr>
<tr>
<td>129</td>
<td>8</td>
<td>1.05E-1</td>
<td>1.51E-1</td>
<td>1.44E-1</td>
</tr>
<tr>
<td>257</td>
<td>8</td>
<td>1.14E-1</td>
<td>1.50E-1</td>
<td>1.44E-1</td>
</tr>
</tbody>
</table>

197
7.11 Comparison of 2D Black Box Multigrid Methods

We have looked at several example problems and how the various methods perform for those examples, but now we would like to determine which methods really are the best. The most obvious criterion for judging the best method would be to use the execution time to solve a given problem to a given tolerance. The trouble with this metric, although it is quite practical, is that it does not take into account implementation and algorithm variations, nor does it say anything about whether one method is more efficient than another method.

We need a criterion that will measure both the execution time and the final accuracy of the solution. We propose to use a normalized execution time and the average convergence factor for our metric. We define the metric to be

\[ P = T \times CF_{ave} \tag{7.17} \]

where \( P \) is the performance metric, \( T \) is the normalized execution time, and \( CF_{ave} \) is the average convergence factor. By normalized execution time we mean that we have taken the tie for five V-cycles and the setup time and divided by five to give the average time for a V-cycle plus one-fifth the setup time. This allows us to take into account the variation in setup time for the different methods. The execution cycle time and average convergence factor are measured for how long it takes a method reduce the initial L-2 norm of the residual by a given amount; the results shown here use a reduction of six orders of magnitude. Like the convergence factor the smaller the performance metric \( P \) the better the method.

The first comparison is for test problem 8, the four-corner junction problem (section 7.5); see table 7.57. The methods are listed in the far left hand column of the table where the characters represent the various methods. The method character string can be decoded into four fields; 1st character, 2nd character, 3rd character, and
Table 7.57. Comparison of various Black Box Multigrid methods on the Cray Y-MP for a 2D diffusion equation given in problem 8

<table>
<thead>
<tr>
<th>method</th>
<th>9x9</th>
<th>17x17</th>
<th>33x33</th>
<th>65x65</th>
<th>129x129</th>
<th>257x257</th>
</tr>
</thead>
<tbody>
<tr>
<td>sciv11</td>
<td>2.883E-7</td>
<td>8.950E-6</td>
<td>6.051E-5</td>
<td>5.377E-4</td>
<td>7.668E-3</td>
<td>1.315E-2</td>
</tr>
<tr>
<td>sciv01</td>
<td>1.183E-7</td>
<td>3.660E-5</td>
<td>8.650E-4</td>
<td>6.755E-3</td>
<td>3.715E-2</td>
<td>1.859E-1</td>
</tr>
<tr>
<td>sciv11</td>
<td>2.648E-11</td>
<td>1.504E-6</td>
<td>4.366E-4</td>
<td>8.317E-3</td>
<td>5.438E-2</td>
<td>2.679E-1</td>
</tr>
<tr>
<td>htpv11</td>
<td>2.083E-5</td>
<td>5.550E-5</td>
<td>2.144E-4</td>
<td>5.802E-4</td>
<td>1.714E-3</td>
<td>5.786E-3</td>
</tr>
<tr>
<td>Ziv01</td>
<td>5.295E-6</td>
<td>1.884E-5</td>
<td>7.367E-5</td>
<td>2.125E-4</td>
<td>6.020E-4</td>
<td>1.992E-1</td>
</tr>
</tbody>
</table>
4th through 6th characters. The first field represents the grid transfer operator used; s = schaffer, h = hybrid, o = original. The second field represents the coarsening; t = standard, c = semi-coarsening. The third field represents the type of smoother employed; a = alternating line, i = ILLU, l = line, p = point. The fourth field represents the type of multigrid cycling; v11 = V(1, 1)-cycling, V(0, 1)-cycling.

Table 7.57 shows that the standard coarsening method with alternating line relaxation using Schaffer’s idea for the grid transfer coefficients is the best method for larger grids, while the same method with ILLU for the smoother is the best for smaller grids. Most of the methods perform about the same, but semi-coarsening with an ILLU smoother is the worst. It is a little surprising to see that De Zeeuw’s MGD9V is beaten by the standard coarsening alternating line method when it was seen before that MGD9V had a faster execution time. Recall from the numerical examples that the ILLU smoother was essential for obtaining good convergence for convection–diffusion equations, except for the semi-coarsening method.

The examination of a convection–diffusion equation for our new performance metric should prove enlightening. We will choose to look at the convection–diffusion problem 9 [24].

Table 7.58 show the performance metric for the convection–diffusion equation given in problem 9 for the various methods. The two clear winners for this problem are the standard coarsening method using ILLU for the smoother with the grid transfer operators computed by either Schaffer’s idea or the hybrid collapsing method.

While stiv11 and htv11 are the best, De Zeeuw’s method, MGD9V, shows nice consistency in performance for the range of grid sizes, but it also has the advantage of being more robust for more complex convection–diffusion equations, especially those with re-entrant flows. While our methods are still convergent for most of the more
Table 7.58. Comparison of various Black Box Multigrid methods on the Cray Y-MP for a 2D convection–diffusion equation given in problem 9.

<table>
<thead>
<tr>
<th>method</th>
<th>9x9</th>
<th>17x17</th>
<th>33x33</th>
<th>65x65</th>
<th>129x129</th>
<th>257x257</th>
</tr>
</thead>
<tbody>
<tr>
<td>stav11</td>
<td>1.835E-6</td>
<td>6.188E-5</td>
<td>1.263E-3</td>
<td>5.060E-3</td>
<td>2.181E-2</td>
<td>9.300E-2</td>
</tr>
<tr>
<td>htau11</td>
<td>2.149E-6</td>
<td>6.473E-5</td>
<td>1.248E-3</td>
<td>5.064E-3</td>
<td>2.208E-2</td>
<td>1.285E-1</td>
</tr>
<tr>
<td>otav11</td>
<td>8.287E-5</td>
<td>4.422E-4</td>
<td>3.281E-3</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>stiv01</td>
<td>2.286E-12</td>
<td>1.069E-10</td>
<td>4.757E-9</td>
<td>5.709E-6</td>
<td>1.241E-3</td>
<td>5.667E-2</td>
</tr>
<tr>
<td>htv01</td>
<td>2.144E-12</td>
<td>8.683E-11</td>
<td>3.526E-9</td>
<td>2.996E-6</td>
<td>5.472E-5</td>
<td>2.389E-3</td>
</tr>
<tr>
<td>otv01</td>
<td>4.249E-12</td>
<td>9.064E-11</td>
<td>2.146E-9</td>
<td>1.506E-6</td>
<td>6.271E-5</td>
<td>*</td>
</tr>
<tr>
<td>stiv11</td>
<td>2.124E-16</td>
<td>4.079E-14</td>
<td>8.658E-12</td>
<td>1.682E-9</td>
<td>1.537E-6</td>
<td>1.208E-4</td>
</tr>
<tr>
<td>htv11</td>
<td>2.022E-16</td>
<td>3.651E-14</td>
<td>7.519E-12</td>
<td>1.432E-9</td>
<td>1.163E-6</td>
<td>9.424E-5</td>
</tr>
<tr>
<td>otv11</td>
<td>3.652E-16</td>
<td>5.015E-14</td>
<td>7.086E-12</td>
<td>9.420E-10</td>
<td>6.740E-7</td>
<td>***</td>
</tr>
<tr>
<td>scvl11</td>
<td>1.097E-6</td>
<td>4.410E-5</td>
<td>6.635E-4</td>
<td>4.112E-3</td>
<td>2.738E-2</td>
<td>1.708E-1</td>
</tr>
<tr>
<td>scv01</td>
<td>1.158E-11</td>
<td>5.618E-10</td>
<td>7.717E-7</td>
<td>2.182E-5</td>
<td>1.409E-3</td>
<td>5.587E-2</td>
</tr>
<tr>
<td>Ziv01</td>
<td>3.335E-8</td>
<td>1.20E-7</td>
<td>4.883E-7</td>
<td>2.630E-6</td>
<td>1.477E-5</td>
<td>1.293E-4</td>
</tr>
</tbody>
</table>
complex convection-diffusion equations, they are not useful as solvers because the convergence factor is often above 0.9. We are not going to present the performance metric for the more complex convection problems because it is difficult to get convergence for large grids even using MGD9V.
CHAPTER 8

PARALLEL ALGORITHM CONSIDERATIONS IN
TWO DIMENSIONS

The parallel algorithm has undergone a lot of changes as the hardware and software support have changed. Originally, there were several codes developed for the CM-2 and CM-200, but when the CM-5 came along, those codes were abandoned. We will present only the CM-5 code, after a brief outline of the previous work.

8.1 CM-2 and CM-200 Parallel Algorithms

We used the CM-Fortran SPMD model on the CM-2 and CM-200, which contained 65K integer processors and 4K floating point processors (Weitek). The CM-200 computer had two modes of operation called the paris (parallel instruction) model and the slicewise model. The slicewise model was the preferred one because it looked at the machine as if it were only constructed of the 4K floating point processors. The CM-2 and CM-200 have a front end (host) computer, a Sparc workstation, that is connected to the hypercube data network, which connects all the processors. The integer processors are bit serial, and the floating point processors are 32 bit based. On the CM-200 the data is stored bitwise and passed through a transpose device which packs it into 16 and 32 bit words, which can be passed to the floating point processors.

The only real differences between the CM-2 and CM-200 are that the transposer and slicewise models were made standard on the CM-200 and that the overall
hardware for both the processors and networks are faster on the CM-200.

Codes were developed for the standard coarsening algorithm using all three of the data structures shown in figure 8.5. The CMSSL (CM Scientific Software Library) was very limited at that time and we had to write our own tridiagonal line and direct solvers. The direct solver was so slow that we found it easier and faster to use the front end to perform and store the LU decomposition and then just pass the right hand side to the front end and the solution back to the CM-2. On the CM-200, which was faster than the CM-2, the direct solver ran about as fast or a little faster than using the front end and passing the two vector arrays.

All of the approaches taken to develop an efficient black box multigrid code ran into trouble with communications, and their performance was dismal when compared to the Cray Y-MP. The communications bottle neck appeared in both the transpose device and the processor-to-processor communication. There was no way around the transpose device, so efforts had to be concentrated on communications between processors. Much of this effort was hampered by the lack of control over the data’s automatic layout across the processors.

Dendy, Ida, and Rutledge [32] partially solved the layout problem by writing code to access the CM Run Time System (CMRTS). The code they wrote is essentially at the assembly level. While this approach met success, it also was not portable and when either the hardware or system software was updated (modified), the code could not be guaranteed to work properly. The layout problem was not really resolved until the codes were implemented on the CM-5 computer.

There were several attempts to reduce the communication overhead by using various communication packages. The two we tried were the CM FastGraph package and the CMSSL poly-shift routines.

The CM FastGraph (CMFG) package has two major components consisting
of the communication compiler and the mapping facility. The CMFG can be used to speed up general communications between multiple processors. The speedup is achieved by determining and storing the routing paths once at the beginning of the program’s execution. When the communications are performed, there is thus no need to determine the routing paths dynamically, thus reducing the time it takes to complete the communications. The CMFG components were used for the grid transfer operations between fine and coarse grid levels. First the communication map is defined for passing data between the different grid level data structures. The CMFG compiler then creates the maps to be used by the CMFG communication routines. While the communications were much quicker, the overhead involved in defining the communications was very prohibitive, often adding minutes to the execution time depending on the size and number of data communications to be performed. This approach, while interesting, just was not practical for our software library approach. However, the CMFG approach has proven to be useful for problems that run the same size problem many times (e.g. time-dependent problems), by storing all the routing paths for all the grid levels.

Throughout the multigrid algorithm there are many instances where communication is needed with neighboring grid points; in such a case, multiple calls are made to communication routines. These communications are either circular (CSHIFT) or end-off (EOSHIFT). The CSHIFT routine shifts the data in a circular fashion with wrap around occurring at the array boundaries. The EOSHIFT routine shifts data with data dropping off the end of an array with a predefined value being shifted into the array. The poly-shift communication routines allow the user to define and use communication stencils to combine multiple calls to other communication routines, CSHIFT or EOSHIFT, into only one communication call. The poly-shift stencil communication allows the various communications that make up the stencil to be overlapped when possible, and hence to reduce the amount of time spent on the communications. The
Table 8.1. Timing comparison per V-cycle for the semi-coarsening code on the Cray Y-MP, CM-2, and CM-200. Times are given in seconds, and the CM-2 and CM-200 times are the elapsed time for 256 and 512 Floating point processors. "*" means the problem was too big to run, and "-" means there was no data.

<table>
<thead>
<tr>
<th>Size n × n</th>
<th>Cray Y-MP (old)</th>
<th>Cray Y-MP (new)</th>
<th>CM-2 256</th>
<th>CM-2 512</th>
<th>CM-200 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.004</td>
<td>0.0005</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>0.006</td>
<td>0.0009</td>
<td>0.51</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>32</td>
<td>0.009</td>
<td>0.0019</td>
<td>0.59</td>
<td>0.75</td>
<td>-</td>
</tr>
<tr>
<td>64</td>
<td>0.041</td>
<td>0.0044</td>
<td>0.65</td>
<td>0.77</td>
<td>0.14</td>
</tr>
<tr>
<td>128</td>
<td>0.093</td>
<td>0.013</td>
<td>0.99</td>
<td>0.80</td>
<td>0.23</td>
</tr>
<tr>
<td>256</td>
<td>0.267</td>
<td>0.043</td>
<td>1.84</td>
<td>1.79</td>
<td>0.40</td>
</tr>
<tr>
<td>512</td>
<td>0.948</td>
<td>0.158</td>
<td>4.55</td>
<td>3.04</td>
<td>1.00</td>
</tr>
<tr>
<td>1024</td>
<td>3.69</td>
<td>0.656</td>
<td>*</td>
<td>8.11</td>
<td>3.01</td>
</tr>
<tr>
<td>2048</td>
<td>*</td>
<td>2.714</td>
<td>*</td>
<td>25.39</td>
<td>-</td>
</tr>
</tbody>
</table>

poly-shift communications consist of three routines. The allocation and stencil setup routine $\text{pshift\_setup}$, the communication routine $\text{pshift}$, and the deallocation routine $\text{deallocate\_pshift\_setup}$. The use of the poly-shift communications in the two-dimensional codes gave a very slight performance gain for the standard coarsening code and was actually slightly detrimental for the semi-coarsening code. Only the three-dimensional standard coarsening code showed any real benefits from using the poly-shift communication routines. The reason for the lack of performance gains was overhead. There was a set of poly-shift routines that were not as general as the CMSSL version, but performed nearly three times faster because of much less overhead. Use of the specialized poly-shift routines would have caused the multigrid codes to run considerably faster, but they still would not have been competitive with the Cray Y-MP versions.

8.1.1 Timing Comparisons For historical reasons it is interesting to compare the timing results for both the standard and semi-coarsening black box multigrid codes on both the Cray Y-MP and CM-2.
Table 8.1 gives a timing comparison for the semi-coarsening code on both the
Cray Y-MP and CM-200. The Cray Y-MP gives times for the "old" original vector
code by Dendy and the "new" optimized vector code that we developed. The CM-200
semi-coarsening code timings are the same as those in [32], page 1466. The CM times
are for the elapsed time, which is the sum of the busy and idle times. Busy time is
defined to be the parallel execution time, and idle time includes the sequential and
communication time. Note that the elapsed time is not the same as wallclock time,
which includes time-sharing and system overhead time. The CM-200 code is at least
an order of magnitude slower than the new Cray Y-MP code. However, if we examine
the trend, we can see that the CM-200 is slowly gaining on the Cray Y-MP code;
unfortunately we will run out of processors and memory long before it can catch up.

Table 8.2 gives comparison timings for the standard coarsening code using
alternating zebra line Gauss-Seidel smoothing. The Cray Y-MP timings also refer to
the “old” original standard coarsening code by Dendy [27], that was ported to the
Cray Y-MP, and the "new” code that we developed. The same observations can be
made about the standard coarsening codes as was made about the semi-coarsening.
However, the standard coarsening CM-200 code is more than twice as slow as the
semi-coarsening code. The reason for this fact can be understood by noting that the
standard coarsening version requires more communication and uses the inefficient data
layouts that the system provides. The standard coarsening parallel code did not make
use of poly-shift communications.

### 8.2 CM-5 Hardware Overview

The CM-5 computer, as we have already said, is an SPMD computer with
up to 1024 processors. We have considered only the data parallel model of execution
in our studies. A program, under this model, is copied into each processor’s memory,
Table 8.2. Timing comparison per V-cycle for the standard coarsening code on the Cray Y-MP, CM-2, and CM-200 using AZLGS smoothing. Times are given in seconds, and the CM-2 and CM-200 times are the elapsed time for 256 and 512 floating point processors. “*” means the problem was too big to run, and “-” means there was no data.

<table>
<thead>
<tr>
<th>Size n</th>
<th>Cray Y-MP (old)</th>
<th>Cray Y-MP (new)</th>
<th>CM-2 256</th>
<th>CM-2 512</th>
<th>CM-200 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.0031</td>
<td>0.0005</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>0.0068</td>
<td>0.001</td>
<td>1.49</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>32</td>
<td>0.0123</td>
<td>0.002</td>
<td>1.58</td>
<td>2.02</td>
<td>0.697</td>
</tr>
<tr>
<td>64</td>
<td>0.0275</td>
<td>0.0046</td>
<td>1.81</td>
<td>2.11</td>
<td>0.914</td>
</tr>
<tr>
<td>128</td>
<td>0.072</td>
<td>0.0128</td>
<td>2.71</td>
<td>2.21</td>
<td>1.52</td>
</tr>
<tr>
<td>256</td>
<td>0.266</td>
<td>0.0443</td>
<td>5.04</td>
<td>4.79</td>
<td>2.45</td>
</tr>
<tr>
<td>512</td>
<td>0.957</td>
<td>0.165</td>
<td>12.34</td>
<td>7.89</td>
<td>4.21</td>
</tr>
<tr>
<td>1024</td>
<td>3.84</td>
<td>0.673</td>
<td>*</td>
<td>20.41</td>
<td>12.55</td>
</tr>
<tr>
<td>2048</td>
<td>15.42</td>
<td>2.792</td>
<td>*</td>
<td>62.89</td>
<td>-</td>
</tr>
</tbody>
</table>
and then every processor executes the same instruction at the same time on its own
data. When data is needed from the other processors, it is passed through a data
communication network that connects all of the processors to each other.

To design parallel programs it is essential to understand the underlying hard-
ware, the high level parallel programming language, and the behavior of the parallel
run time system. While much effort is being made to hide these issues and make them
transparent to the average user of the computing community, there is much room for
improvement. All parallel computers and their high level languages, to date, have not
had any real success in making it possible to ignore these three issues. This situation
has mostly been due to a lack of software support both in the languages and software
libraries. Some have said that object oriented design and algorithms is the answer, but
first the underlying framework and code has to be developed, by no means a trivial
task, and the computing community is not even close to the general prototype stage at
this time.

The description of the CM-5 given so far is not sufficient to understand and ap-
preciate the issues and complexities of designing an efficient parallel algorithm. Hence,
we provide a more detailed description.

The CM-5 is an SPMD computer with 1024 processing nodes, 4096 vector
units, a partition manager, and I/O processor(s) that are all connected by a commu-
ication network; see figure 8.1. The partition manager is the user’s gateway to the
CM-5. It is essentially a Sparc workstation that manages the various user partitions
(time–sharing) of processing nodes, networking, and I/O communications with external
devices, such as remote terminals, disk and tape drives, and printers. It also executes
and stores all of a program’s scalar data and scalar instructions along with the CM
Run–Time-System (CMRTS).

There are two communication networks, one for control and the other for data.
Figure 8.1: CM-5 system diagram for n processing nodes (PN).
The control network is used by operations that require all the processing nodes to work together, such as partitioning of the processing nodes, global synchronization, and general broadcasting of messages. The control network is implemented as a complete binary tree with all the system components at the leaves.

The data communication network is set up in the form of a fat-tree connection network that allows data to flow in both directions simultaneously. A fat-tree is basically a tree in which the connections become denser as one progresses toward the root, allowing for wider communication channels to cope with the increased message traffic. The fat-tree is a 4-ary tree with either 2 or 4 parent connections. It is used for point to point communication between individual processing nodes. The fat-tree network allows a variety of data structures to be mapped onto it easily, such as vectors, matrices, meshes (1, 2, and 3 dimensional grids), hypercubes, and of course hierarchical tree-like structures, while maintaining a high bandwidth.

The organization of the individual processors of the CM-5 is as follows: each consists of four memory banks of 8 to 32 MBytes each, four vector processing units, a RISC (Sparc) processor, and a network interface controller, all connected together by an internal bus; see figure 8.2. Each memory bank is connected to a vector unit which in turn connects to the internal bus. The RISC processor manages the issuing of vector instructions to the vector units, address computations, loop control, and other housekeeping activities. The network interface controller transmits and receives messages from the fat-tree network, communicating with other processors and I/O units.

The vector units act both as a memory manager for the node’s RISC processor and as a vector arithmetic accelerator. The RISC processor can read or write to any of the four vector unit memory banks. The vector instructions are given by the RISC processor as memory address instructions with special bits set to indicate the type of
Figure 8.2: CM-5 processor node diagram with four vector units.
vector operation to be performed.

It is important to note that the vector units are not independent processors. They do not load or execute instructions, but merely perform memory management and vector arithmetic functions. However, because they do perform all the arithmetic functions from the user's program, it is convenient to think of the vector units as acting like processors.

The vector units consist of an internal bus interface, a vector instruction decoder, a memory controller, a bank of 64 64-bit registers, and a pipelined arithmetic logic unit (ALU); see figure 8.3. The bank of registers can also be addressed as 128 32-bit registers. In addition there are a couple of control registers. There is a 16-bit vector mask register and a 4-bit vector length register. The vector mask register controls certain conditional operations and receives single bit status results. The vector length register indicates the vector length that is being used and ranges from a vector length of 1 to 16; thus, a vector length of one is used for scalar operations.

It should be noted that the original vector length on the CM-5 was 4, in 1993 it was increased to 8, and finally it was increased to 16 in late 1995. The increase in the vector length has improved the performance of the black box multigrid codes over the years, and in addition, it has caused modifications in the implementation of several of the algorithmic components.

A special note concerning communication as it relates to the vector units, which are implemented two to a chip is that units 0 and 1 are on one chip and units 2 and 3 on the other. This arrangement means that communication between two vector units on the same chip are faster than communication between vector units of the same processing node but different chips. Communication between vector units of differing processing nodes will involve the data communication network and be much slower.

There are 1024 processors on the CM-5, but since each processor contains
Figure 8.3: Diagram of one CM-5 vector unit.
four vector units (processors) it is better to think of the CM-5 as having 4096 vector processors. This viewpoint is justified because the vector units perform virtually all the computations. When we refer to a processor in our discussion, we will in general be referring to the vector processing units.

8.3 CM-5 Memory Management

Memory management is usually less important than either parallel computation or communication in determining the speed in program execution, but it is still one of the major considerations in obtaining good performance in CM-Fortran. It is possible to run out of parallel memory even when it would appear that the data should fit. To understand why this can happen we need to examine the memory more closely.

Scalar memory is any memory region not dedicated to storing parallel data, including the scalar memory of the partition manager and any portion of the processing node’s memory not being used by the parallel stack or heap.

Parallel memory is any region of memory located on all four vector unit memory banks on all the processing nodes. Parallel memory is the same size and starts at the same address on all the memory banks. There are two types of parallel memory: stack and heap. Stack memory is temporary memory. Heap memory is relatively permanent and is allocated and deallocated arbitrarily and never gets compacted. Because heap memory does not get compacted it can become fragmented and leave areas of memory unusable.

The partition manager runs a complete UNIX operating system, but each processing node runs only a subset which is called the PN kernel. The PN kernel occupies about 0.5 MBytes of memory in each processing node’s memory.

It is enlightening to see how a processing node’s memory is partitioned to hold the PN kernel, parallel stack memory, parallel heap memory, scalar memory, and user
code; see figure 8.4. The CM operating system is responsible for assigning the memory pages. The parallel memory pages must be aligned across all four memory banks on a processing node, but the scalar memory pages can be assigned arbitrarily.

The node’s memory is organized into high and low memory with the PN kernel stored in vector unit 0’s low memory. The PN kernel takes up only half a megabyte of memory, but it effectively takes up 2 MBytes because its memory shadow on the other three memory banks are unusable for parallel data. To make things worse, the 1.5 MBytes of memory in the PN kernel shadow is not always assigned for scalar memory, so that other memory locations used for the scalar memory also cause a memory shadow that blocks parallel memory assignment.

The user code (variables and instructions) are stored in high memory in scalar memory pages. This arrangement leaves the parallel stack and heap stored in between the PN kernel and user code, with the stack always being stored towards the low end memory relative to the parallel heap. Both the parallel stack and heap grow from their starting locations towards high memory.

Parallel arrays come in many forms and are stored either on the stack or heap. There are four types of user defined arrays and two types of compiler generated arrays.

The user defined arrays types will be defined now; in the discussion "local" means declared in a routine and not defined outside of that routine. Ordinary local arrays are those declared in a routine without the SAVE or DATA attribute. These arrays are allocated on the stack on entry into the routine and deallocated upon exiting the routine. Permanent local arrays are declared with the SAVE or DATA attribute. They are allocated on the heap when entering the routine for the first time and are never deallocated. Dynamically allocated arrays are explicitly allocated and deallocated by function calls and are stored on the heap. Common block arrays are allocated on the heap when the array is first used and are never deallocated.
Figure 8.4. CM-5 processor node memory map for vector unit configuration. Area 1 is the scalar stack, area 2 is the scalar heap, and area 3 is the user’s code (scalar variables and instructions). White space is unclaimed memory and is neither scalar nor parallel.
The compiler generates three kinds of internal temporary arrays which are all stored on the heap. The first type are communication temporaries. They are arrays that temporarily hold results from communication operations which are the result of either explicit or implicit communication taking place in an expression. The second kind of temporaries are the computation temporaries. They are the result of either computations being performed inside a communication function (e.g. CSHIFT) or when a selection type statement is executed (e.g. FORALL, WHERE). Communication and computation temporaries are allocated at the time the expression is evaluated and are deallocated when the calculation of the expression has completed. The third kind are common subexpression temporaries. They are arrays that hold values of common subexpressions between the first time they are used and the last time that they are needed. A common subexpression temporary is generated to store every WHERE statement’s mask, and sometimes the mask for FORALL statements.

We will need to define what we mean by basic code block and PE code block in order to simplify our discussion. A basic code block is a segment of statements bounded by control flow statements. A PE code block is a region of pure parallel computation involving no control flow statements.

The compiler can collapse and reuse communication and computation temporaries in basic code blocks, instructions that do not involve control flow, if the temporaries have the same shape.

Common subexpression temporaries can be stored in registers with increased speed and efficiency if they occur within the same loop, PE code block, and basic code block.

When the amount of memory available on the processing nodes becomes an issue, there are several rules of thumb that should be followed.
1. Use more complex array expressions and less arrays.

2. Rewrite code fragments to reduce the number of temporary arrays generated by the compiler.

3. Try to reuse arrays or parts of arrays.

4. Split up program units so that fewer arrays and temporaries are allocated at one time.

5. Use the aliasing functions to use the same memory for arrays of different layouts.

6. Consider what the effects will be for garbage element array padding and vector-length padding, which can invisibly increase an array's storage.

7. Use the scratch space on the parallel I/O devices, e.g., Data Vault.

Temporary array compiler allocation can be reduced in several ways. Write expressions so that common subexpressions are easily recognized; if one cannot easily see them, then the compiler may not be able to see them either. Avoid writing complicated expressions that involve many array functions or changes of array layouts, which will cause the generation of communication and computation temporaries. Be aware that most communication functions assume that their source and destination arrays are distinct. If the source and destination are the same array then a communication temporary will be generated.

8.4 Dynamic Memory Management Utilities

The Dynamic Memory Management Utilities (DMMU) were developed by W. Spangenberg at Los Alamos National Laboratory [8]. The DMMU were designed to address the problem of control of the data structure layout on the CM-5 processors. The
CM–FORTRAN compiler and Run–Time–System (RTS) aggressively monitor the array layouts to assure that arrays are distributed uniformly across the processors. While the monitoring and re-distribution of array layouts is good for many applications, it can be disastrous for multigrid method performance. As we have indicated before, when the VP ratio is less than or equal to one, the most efficient communication was through the use of compatible arrays and masks. When the VP ratio is greater than one we used incompatible arrays, because compatible arrays led to inefficient computations and storage. When the aspect ratio of parallel grid axes become large or processors become unused, the CM RTS re-distributes the arrays to minimize the number of idle processors. The re-distribution occurs when the grid dimensions are not equal and also when coarsening takes place, especially for the semi-coarsening multigrid method. The re-distribution problems can persist even when using the PROCS and BLOCS attributes for the array layout.

The fine grid coefficient matrix (array), \( L \), defines the physical layout of the data on the processors. The layout is specified by the LAYOUT command which is part of CM FORTRAN and High Performance FORTRAN (HPF). The command specifies the serial and parallel dimensions of the array (SERIAL and NEWS attributes respectively), and in addition, it can also specify the the physical extents of the array onto the processors (PROCS attribute) and the subgrid size on the processors (BLOCS attribute). The PROCS attribute defines the physical extents of the array across the processors, and the BLOCS attribute defines the array axis subgrid length, which is given as a ratio of the axis extent to the physical processor extent.

Every array has a geometry descriptor that contains these attributes. The DMMU use the fine grid coefficient array’s geometry descriptor as a template to control the layout of the other arrays. For each grid dimension, all arrays have the same physical extents over the processors by using a common PROCS directive, from the geometry
descriptor template, for each parallel axis. All arrays on each grid level, in each grid dimension, have the same subgrid extents which are specified by a common BLOCS directive, from the geometry descriptor template, for each parallel axis. The DMMU uses the geometry descriptor template to dynamically allocate arrays with the same layout on each grid level. The compatible and incompatible arrays for different grid levels can also be aligned using the DMMU to obtain more efficient communications between grid levels. The intergrid communications require the use of temporary arrays for efficiency, and are also allocated dynamically by the DMMU. The use of the DMMU ensures that the FORTRAN 90, CM FORTRAN, and HPF array and array section operations perform correctly while also allowing for the most efficient inter-processor communication.

The black box multigrid codes use the DMMU to dynamically allocate all of the internal arrays. The process works in the following way. First we need to determine the number of grid levels and their size. Next we need to determine the number of processors and at which grid level the VP ratio will be less than or equal to one. We then obtain the geometry descriptor template for the fine grid coefficient array. We determine the physical extents across the processors of the congruent geometry template’s parallel axes, assign the appropriate axis’ physical extent, and check to see that we have a consistent physical extents across the processors. We then determine the appropriate PROCS and BLOCS directives for each grid level. We then create a congruent array alias for each array to conveniently reference the different grid levels via an index. Finally, we dynamically allocate all the coarse grid arrays, using the geometry descriptor template to enforce the desired data layout.
8.5 CM-5 Software Considerations

Due to the state of compilers today, when designing a parallel program, one should keep in mind the computer’s architecture to obtain reasonably fast performance. Dropping down out of a higher level language to the underlying support language, sometimes even to assembly language, will always yield the best performance, but will inevitably lead to non-portable, long, and confusing code. However, the underlying structure of a computer often changes with updates to both the hardware and system software, certainly the case with the CM-5. These changes can greatly affect the life of any computer program which uses specific hardware or low-level software features. The high level computer languages are much more stable and are usually not affected by these changes. Hence, we have chosen to keep the code as portable and readable as possible by exclusively using the higher level languages.

The CM-5 supports several higher level languages that support the data parallel model, the two most common being C and FORTRAN. We have chosen to use FORTRAN because it is more stable, and its behavior is better understood for performing numerical computations than is the behavior of C. The flavor of FORTRAN that the CM-5 employs, called CM-FORTRAN, is a subset of FORTRAN 90, but it also includes the entire F77 ANSI standard and a few CM extensions. The parallelism is achieved by the way that FORTRAN 90 expresses looping indices, data structures, and data dependencies in the program. This simplification in notation can lead to more compact and easier to read codes for most algorithms. However, care must be taken to note if a variable is actually an array or variable and whether it resides on the front-end (sequential) or on the CM (parallel) side of the computer.

It is important to note some relationships in computing and communication on the CM-5. Computations are very fast if they are performed entirely on one processor.
Communications are very expensive when compared to computations but can vary a lot among themselves. There are several types of communications that we are interested in: circular, end–off, and irregular. The circular and end–off shift communications are about the same speed, with circular getting the edge, but they are both much faster than the irregular pattern communications. Another aspect that relates to speed is the distance of the communication path between processors. If the distance is a power of 2, then the best performance results, but the fastest is nearest neighbor communication, which is only slightly faster.

8.6 Coarsening and Data Structures in 2D

It is important to understand how the data are laid out in the memory across the processing nodes. When a parallel array has more elements than there are processors, the array is decomposed into contiguous subsets and spread across the processing nodes. The subsets are called subgrids and they are uniform in size, shape, and starting memory address on all the memory banks on all the processing nodes. If the data will not fill all the subgrids, then some of the subgrids will be padded with null data until they are all full. Computations taking place on subgrid data are all done sequentially on a processing node by the vector units. The best efficiency is obtained when there is no padding, and thus full vector operations can be performed. When padding is present in the subgrids additional overhead is incurred; first no-op instructions are sent to the vector units associated with the padded elements, and then a mask is created and used to prevent storing the results.

We have considered two types of coarsening to generate the coarser grids: standard coarsening and semi-coarsening. The degree of parallelization is quite different for these two choices. For a given grid, the next coarser grid has one fourth the number of grid points for the standard coarsening method versus one half in the
semi-coarsening method. For the standard coarsening method, the number of points in both dimensions is reduced by half, taking every other grid point from the fine grid to form the coarser grid. The semi-coarsening method reduces the number of points in only one dimension by half. The rest of the discussion will concentrate on the standard coarsening method but will equally apply to the semi-coarsening method with the obvious difference that coarsening is only in one direction. Any comments that do not apply to both methods will be pointed out as they arise.

There are several ways in which we can set up the grid data structures. The fine grid, in two dimensions, is layed out and partitioned into subgrids across the processors, see figure 8.5(a). Ideally this layout is thought of as one grid point per processor. However, it is important to remember that for large grids, each processor contains a contiguous subgrid of grid points and that each grid point is treated as if it were on an individual processor; such imaginary processors are often referred to as virtual processors, but we will just refer to them as processors.

In order to discuss the data structures and their relationship to communications, we need to define the grid spacing relationship between the different grid levels. Let us assume a uniform fine grid for now, but our comments will equally apply to the data structures for non-uniform grids. Recall the notation used in Algorithm MGV($k$, $\nu_1$, $\nu_2$, $h$), from section 1.4, where $k$ referred to the grid level and ranged from 1 (coarsest) to $M$ (finest). The coarse grid spacing, for our solvers, is determined by doubling the fine grid spacing, $h^{k-1} = 2h^k$, this leads to the equation,

$$d = 2^{M-k}$$

where $d$ is the grid communication distance between neighboring grid points on grid level $k$ and $M$ is the total number of grid levels. The actual grid spacing, for a uniform grid, on grid level $k$ is $h^k = dh^M$. 

Figure 8.5. Standard coarsening grid data structure layout for the finest (left), coarse (center), and coarsest (right) grid levels, where • represents an active processor (grid point), and figures (a), (b), and (c) represent three different data structures.
Most of the computations in the multigrid algorithm are performed using neighboring grid points, which requires communications between the active grid points. The distance of the communications between nearest neighbor grid points is given by equation 8.1. This formula is valid in the direction of the coarsening for both standard and semi-coarsening in two or three dimensions.

A possible disadvantage of all the grid levels sharing the same data structure is that the distance of the communication increases with the coarseness of the grid level. The increase in communication distance may cause slower data transfer rates.

A further disadvantage for the standard coarsening method is that most of the CMSSL routines can not be used on coarser grids because the communication distance is greater than one. This disadvantage can be overcome if we introduce data structure transformation routines to convert data from our data structures to ones on which the CMSSL can operate. The CMSSL routines are needed for the LU decomposition and to perform tridiagonal line solves, which are used in the smoother and the direct solver respectively. The conversion of the data structures will of course increase the execution time. If we keep the same data structure and do not want to use conversion routines, we are then forced to abandon the CMSSL and write our own routines. Writing our own LU decomposition solver and tridiagonal solvers for the CM-5 is not a trivial task, and it will be impossible to obtain efficiency any where near that of the CMSSL, written in assembly language, by writing in CM-Fortran alone.

Another choice for the grid data structures is to keep one as the grid communication distance between nearest neighbor grid points. There are two data structures that can be used to accomplish this goal. The first way is to choose to use the same data structure for all grid levels but to use for computation only a compact subgrid of the fine grid level corresponding to the coarse grid. The other points in the data structure are unused, see figure 8.5(b). The second way is to have each coarse grid
have its own compact data structure of just the right size, see figure 8.5(c).

In each case, communication for the computations on a grid level are all nearest neighbor communications, which are the fastest possible communications. However, there are disadvantages with these two data structuring schemes. The grid transfer operations are now complicated and require the use of general communications routines, which are the slowest type of communication between processors.

Clearly the choice of data structures will have a different impact on each of the various multigrid components. For this reason, we examine the multigrid components separately and then discuss the choice of tradeoffs.

8.7 Coarse Grid Operators

The discussion above about data structures covers most of the options for which data structures are reasonable for the coarse grid operators. However, there are a few more pitfalls that should be considered.

The choice of data structures on the CM computers can have a large effect on performance, besides communications, when temporary variables are created to hold intermediate data from computations and when data is passed between routines.

Temporary variables can affect the performance in two ways: size and alignment. The storage for the temporary variable is the same as that of the largest data structure involved in the computation. Complex computations often require several temporary variables. If there are any communications in a computation, then it is almost always the case that a temporary variable is created to hold the data from the communication to be used in the computation. The alignment of a temporary variable is governed by the data structures involved in the computation. The choices that the runtime system makes can sometimes cause a slight misalignment of the data and slow down performance by introducing communication. It sounds much worse than it
really is because these communications have always been found to be between virtual processors or between the vector units on a single processor node, and the overhead is usually negligible if the subgrid size is relatively small.

When data are passed between routines it can cause the creation of temporary variables into which the data are copied. This creation happens when the called routine is only using a subset of the data structure from the calling routine. However, the creation of temporary variables has also occasionally been observed on the CM-2 and CM-200 when the entire data structure has been passed from the calling routine to the called routine; it has not yet been observed on the CM-5.

There is an overhead cost in time that is associated with the creation and use of temporary variables by the runtime system. These costs can sometimes be cut if the implementation already uses temporary variables, but it is best if the implementation can minimize the need for temporary variables altogether.

8.8 Grid Transfer Operators

The grid transfer operations involve mostly communication of data between two grid levels. It is therefore important to minimize the amount of data being transferred and to use the most efficient type of communication that we can. However, the time spent in one $V(1,1)$-cycle on the vector computers (Cray Y-MP) performing grid transfers was only about 25 percent of that spent on smoothing when alternating line relaxation was used. We expect to see the same kind of relationship between grid transfers and smoothing on the CM-5, and the percentage may even drop because the smoother will usually involve more communication than the grid transfers. However, the smoother and grid transfer routines may use different types of communication, affecting the percentage of time spent in each routine.
8.9 Smoothers

There are several relaxation methods available for smoothing; they are the point, x-line, y-line and alternating line Gauss–Seidel relaxation using multicolor ordering. By multicolor ordering we mean that either red/black or 4-color ordering is used.

The active grid points are a power of two distance apart when multi-coloring is used, which means that it is still possible to use efficient communication between processors. Regardless of which data structure layout is used, the multicolor ordering in the smoother will always use communications that are either a power of two apart or nearest neighbor.

Recall the comments from section 8.6 about the CMSSL. If we choose to use the grid data structure layout in figure 8.5(a) we can not use the CMSSL tridiagonal solvers for the line solves without using data structure conversion routines. However, the semi-coarsening method can use the CMSSL tridiagonal solver because the line solves are not in the direction of the coarsening.

There is also the incomplete line LU (ILLU) iterative method, used in the vector code, to consider. However, the ILLU method is not parallelizable in its current form and for this reason we chose not to implement it on the CM-5. Many researchers are working on developing a parallel ILU solver, and we are not aware of any algorithms or efforts to develop the more robust ILLU method on parallel computers.

8.9.1 Parallel Line Gauss–Seidel Relaxation

The CMSSL provides a tridiagonal solver that comes in two forms: one that performs the entire solution process and the other that splits the process up into a call to the LU factorization routine and another call to the LU solution routine. We can use the factorization routine and save the LU decompositions between smoothing steps, but that would
mean saving the factors for every grid level. Saving all the LU decompositions would be costly because the CMSSL also allocates its own temporary work space for each decomposition to be used in the solution phase. We only save about 30% on the execution time for a V-cycle by saving the LU decomposition, but it takes about six times the storage required when not saving the LU decompositions.

The CMSSL \textit{gen-tridiag-solve} routine can be used to solve both X and Y lines by just changing the \textit{vector-axis} parameter to point to the array axis that the diagonal elements lie on. All of the X (Y) lines of a single color can be solved in parallel. The zebra line Gauss-Seidel relaxation will take two tridiagonal line solve times, one for each color. The alternating zebra line Gauss-Seidel relaxation will take a total of four tridiagonal line solve times.

Since the CMSSL tridiagonal solver overwrites both the coefficient and right hand side arrays with the LU decomposition and solution respectively we need to copy the data into temporary work space arrays before calling the solver. Once again extra temporary storage is needed, but it is only allocated for the duration of the smoothing step. This temporary storage space is not saved for each grid, which could easily fill up memory reducing the size of problems that can be solved, but reused on each grid level.

### 8.9.2 CM-5 Tridiagonal Line Solver Using Cyclic Reduction

The tridiagonal systems from the line relaxation on the vector computers used Gaussian elimination with vectorization taking place by solving all the lines of one color simultaneously. We could also use this approach to obtain a parallel tridiagonal line solver by solving all the lines of a single color in parallel. However, this approach still leaves each line to be solved sequentially, and we can do better than that by using cyclic reduction.

The cyclic reduction algorithm is an example of a divide and conquer method.
A tridiagonal system of irreducible linear equations $LU = F$, where $L$ is of dimension $N = 2^n - 1$, can be represented as a matrix equation

$$LU = \begin{pmatrix} b_1 & c_1 \\ c_2 & b_2 & c_2 \\ \ddots & \ddots & \ddots \\ & \ddots & \ddots & \ddots \\ & & a_{N-1} & c_{N-1} \\ a_N & b_N \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{pmatrix} = F. \quad (8.2)$$

The basic idea is to solve for $u_i$ in terms of $u_{i-1}$ and $u_{i+1}$, providing that $b_i \neq 0$. We do this solution for all odd $i$ equations and substitute the expression for $u_i$ into the remaining equations. The result is a tridiagonal system of equations in $[N/2]$ variables. The procedure is applied recursively until only one equation remains. The single equation is then solved and the other variables are obtained through back substitution.

To simplify a more detailed description, let $u_0 = u_{N+1} = 0$ and let the subscripts represent the equation numbers and the superscripts denote the reduction and back substitution steps. Let $a_1^1 = a_i$, $b_1^1 = b_i$, $c_1^1 = c_i$, and $f_1^1 = f_i$; then the reduction step is given by:

$$\alpha \quad = \quad 2^k$$

$$\beta \quad = \quad -\frac{a_i^k}{b_i^{k-\alpha}}$$

$$\gamma \quad = \quad -\frac{c_i^k}{b_i^{k+\alpha}}$$

$$a_{i+1}^{k+1} = \beta_i a_i^{k-\alpha}$$

$$c_{i+1}^{k+1} = \gamma_i c_i^{k+\alpha}$$

$$b_{i+1}^{k+1} = b_i^k + \beta_i c_i^{k-\alpha} + \gamma_i a_i^{k+\alpha}$$

$$f_{i+1}^{k+1} = f_i^k + \beta_i f_i^{k-\alpha} + \gamma_i f_i^{k+\alpha}$$

where $i = \alpha, 2\alpha, 3\alpha, \cdots, (2^n-k-1)\alpha$, for the reduction steps $k = 1, 2, \cdots, n - 1$. After
the $n - 1$ reductions steps we are left with one equation, which when solved is

$$u_{2n-1} = \frac{f_{2n-1}^{n-1}}{b_{2n-1}^{n-1}}. \quad (8.10)$$

The back substitution is given by

$$u_i = \frac{f_i^k - a_i^k u_{i-\alpha} - b_i^k u_{i+\alpha}}{b_i^k} \quad (8.11)$$

where $i = \alpha, 3\alpha, 5\alpha, \cdots, (2^{n-k}-1)\alpha$ and $k = n - 2, n - 3, \cdots, 0$.

The cyclic reduction algorithm just described derives its parallelism by performing all the $i$ indexed equation simultaneously for each reduction step $k$. However, the number of processors needed at each reduction step is half that of the previous step.

Notice that the reduction process can be written to yield any of the unknowns. If we write a set of reduction steps to yield each unknown, after performing $n - 1$ reductions, the set of “single” equations can then be solved simultaneously using an equation similar to (8.10) to give the solution. This method no longer requires the back substitution step, and it also keeps all the processors busy at all the steps, giving a method that is twice as fast as the original cyclic reduction algorithm. This version of cyclic reduction can be found in [46] where it is referred to as the PARACR algorithm, and it is one of the tridiagonal solvers implemented in the CMSSL.

If the VP ratio is much greater than one, then the subgrids on each processor are large and the PARACR algorithm becomes inefficient. A better algorithm is to use block cyclic reduction [59] [49], which is also in the CMSSL. The block cyclic reduction performs the LU decomposition sequentially on each processor and cyclic reduction over the processors.
8.10 Coarsest Grid Solver

One of the advantages of the standard coarsening multigrid algorithm is that each coarser grid level takes only one fourth the amount of work as the previous grid level until the number of the coarse grid points becomes smaller than one grid point per processor; thereafter the time remains constant for the computations on a grid level.

The coarsest grid solver is a direct solver that uses an LU decomposition. The direct solver is slow on the CM-5 due to its sequential nature and is dependent on the number of grid points on the coarsest grid level. It is therefore important to make the coarsest grid level size small, so that the direct solver time is approximately equal to the smoother time on the grid level with one grid point per processor. A banded sparse matrix direct solver does not exist in the CMSSL, even though the documentation states CMSSL routines exist which are equivalent to those of Linpack and Lapack for solving general banded systems. In fact the CMSSL provides routines that only solve tridiagonal and pentadiagonal banded systems.

Another common choice for the coarsest grid solver is to use several iterations of one of the relaxation methods. This choice is not very practical on the CM-5 because of the constant time per iteration per grid level when there are fewer grid points then there are processors. Thus the solution time on the coarsest grid is proportional to the number of smoothing steps. The constant time for the coarsest grid level can only be reduced if a cheaper smoother can be found or if the number of coarse grids which have fewer grid points than processors is kept to a minimum. The problem with this approach is that the larger the system to be solved, the worse the reduction factor for the smoother. To add insult to injury, we are already using some of the cheapest and most effective smoothers. Using a relaxation method on the coarsest grid does not seem to help the situation and indicates that the use of a direct solver is probably better.
A third approach is to use standard coarsening for large grids and then switch to semi-coarsening for the coarser and coarsest grids. This approach solves a few of our problems because the coarsest grid can now be just one line which can be solved directly with the CMSSL’s tridiagonal solver. However, the new question is when to switch from standard to semi-coarsening. It was decided that the user would specify when the switch took place by setting the coarsest grid input parameter.

The optimum value for the switch will be dependent on the finest grid size and the number of processors. Recall that, when the switch to semi-coarsening takes place, coarsening will only happen in the y-direction and that the grid point distribution in the x-direction across the processors will remain fixed. The tridiagonal solve on the coarsest grid level solves one x-line of the size that was fixed when the switch took place, and the tridiagonal solution time is dependent on that size. The smaller the x-dimension size is when the switch takes place, the faster the tridiagonal solution. It therefore becomes necessary to balance the constant time spent performing standard coarsening multigrid, the constant time spent performing semi-coarsening multigrid, and the tridiagonal solution time. A parametric study has not been done, and is not planned, to determine the optimum values. However, from experience a reasonable choice for the switch is when the number of x-direction points is about one quarter the number of processors or less.

Another practical choice, from the programmer’s perspective, is to switch from standard to semi-coarsening when the VP ratio becomes less than one after coarsening. This choice is convenient and keeps the maximum number of processors active. This choice would seem to be very good, but the semi-coarsening performance compared to standard coarsening performance when VP ≤ 1 is highly dependent on the efficiency of the tridiagonal line solver used by the smoother. However, we now have the advantage of being able to use the CMSSL to perform the tridiagonal line solves.
If the coarsening stops before we reach a single grid line or equation, we are left with sparse banded system of equations to solve. The CMSSL does not provide a solver to handle this case except for the dense linear system solvers. Using the dense system solvers means that we would have to allocate more storage and copy the banded system’s data into it with the cost of general communication, the most expensive. The dense system solver will also perform many unneeded computations and communications that will involve matrix entries outside the banded area. All things said, the use of the dense system solvers is very inefficient.

The only solution left for us is to write our own system solver for banded systems. While this may sound attractive, it is not. Once again we are met with the challenge of trying to write a solver that is both efficient and competitive. It is very difficult to write codes in high level languages that can compete with the CMSSL. To make matters worse, the parallelism in Gaussian elimination is at best modest for a banded system, depending on the length of the band. However, if we are willing to sacrifice some memory, we can store the LU factors to save some execution time for the solution of the coarsest grid when several multigrid iterations are performed.

For a dense system of equations the best performance is obtained by using block cyclic ordering of the equations; see [59]. However, the performance gain assumes that $VP \gg 1$, and a sparse banded system with $VP < 1$ will actually perform much slower than most of the other methods.

To obtain an efficient banded system solver requires transferring the data from a grid point oriented data structure to a matrix oriented one, requiring general communication. For efficiency, if we have $N$ unknowns, we will need $N^2$ processors. The computations can then be done in order $N$ operations, but communications will add significantly to the execution time. All things considered, the best we can hope to do for the solver is order $N$ times a constant plus the communications times, which
include the data structure transfers. It should be noted that the constant can be on
the order of $N$ when $N$ is small.

The performance of an LU direct solver is not very attractive when $VP < 1$. The
most efficient solution, so far, is to switch to the semi-coarsening code at sometime
after the $VP < 1$.

8.11 Miscellaneous Software Issues

An interesting compiler deficiency is that a parameter passed into a subroutine
manifests poor performance if it is used as a loop control parameter. The way to avoid
this deficiency is to copy the parameter’s value into a local variable and then use that
variable as the loop control parameter. The poor performance might have to do with the
fact that the passed variable is usually scalar and is stored in the partition manager’s
scalar memory, requiring a broadcast communication every time the variable is needed.

8.11.1 Using Scalapack Scalapack is the same as Lapack but designed
for distributed memory parallel computers using the parallel basic linear algebra sub-
programs (PBLAS) and basic linear algebra communication subprograms (BLACS).
The Scalapack is available on the CM-5 using CM-PVM (parallel virtual machine) un-
der the CMMD message passing model. The SPMD data model is not compatible with
the CM-PVM CMMD model, and for this reason we can not use Scalapack.

If we assume that we had compatible programming models, another problem
with using Scalapack is that the data structures are all matrix oriented, instead of the
grid oriented data structure that we use. Scalapack also assumes that the matrices are
distributed to a grid of processors in a 2D block cyclic decomposition. This distribution
of data would require that we use costly general communications to copy the data into
the block cyclic format.
To top it all off, Scalapack is still under development and the two routines that we would need to perform the LU factorization and LU solution, PSGBTRF and PSGBTRS respectively, have not been implemented yet.

8.11.2 Poly-Shift Communication The PSHIFT communication routines in the CMSSL are also available on the CM-5. The PSHIFT routine achieved the overlapping of communications on the CM-2 and CM-200 because those computers used a hypercube data communication network. The PSHIFT routine on the CM-5 is not very effective because the fat-tree data communication network will not allow as many communications to be overlapped. The PSHIFT setup routine dynamically allocates memory and when a particular poly-shift stencil is no longer needed, the memory should be deallocated.

The PSHIFT routine can perform a maximum of two shifts per array dimension, one in each direction. The number of communications that can be overlapped is limited to at most four, but the size and shape of the stencil are not restricted. If an array has padding in the dimension in which communication is to take place then the PSHIFT will perform approximately the same as the equivalent calls to CSHIFT and/or EOSHIFT. The best performance is obtained when the subgrid lengths in the communication dimensions are all roughly equivalent.

The performance improvement for 2D 9-point stencils is barely noticeable. The PSHIFT routine should be able to do better than it does on the CM-5, but first it will have to be optimized for the fat-tree network.

8.12 2D Standard Coarsening Parallel Algorithm

Many parallel algorithms have been tried over the years in an effort to create an efficient parallel black box multigrid code. The code was first developed the CM-2
and then ported and modified for the CM-200, and finally a version was created for the CM-5. The algorithmic choices presented here are those that were made for the CM-5.

### 8.12.1 Data Structures

The data structures for the grid equations are grid point stencil oriented. There are no fictitious grid equations needed for the boundary as there were in the vector code. The references to neighboring grid points are made through the communications routine EOSHIFT, which gives a zero value for off grid references.

The previous trouble with the data structure layout is solved by the use of dynamic allocation using the Dynamic Memory Management Utilities (DMMU) developed by Bill Spangenberg of Los Alamos National Laboratory in conjunction with Thinking Machines Inc. The DMMU provide a way to allocate arrays dynamically with a given fixed geometry and to be able to use array aliasing to create an array of grid level arrays.

### 8.12.2 Coarsening

We used standard coarsening, taking every other fine grid point in both coordinate directions to form the coarse grid.

The data structures when $VP > 1$ are of the non-compatible compact type; see figure 8.5(c). When $VP \leq 1$, we used a natural grid layout that uses compatible grids as illustrated in figure 8.5(a).

The natural grid layout leaves more and more idle processors with every coarser grid level. However, since we wanted to use the CMSSL for the line solves we switched to semi-coarsening after a few coarse grid levels below $VP \leq 1$.

As a special note, the easiest way to implement the black box multigrid solver for $VP \leq 1$ is to use the semi-coarsening algorithm. This choice keeps the maximum number of processors busy and allows the direct use of the CMSSL tridiagonal solver.
8.12.3 Smoothers  We have implemented the multicolor ordering point, line, and alternating line Gauss–Seidel methods. The ILLU method was not implemented for the reasons given in section 8.9.

However, as mentioned before, we can use the CMSSL tridiagonal solver when $VP > 1$ and also if the semi-coarsening algorithm is used on all grid levels when $VP < 1$. When $VP < 1$ and the semi-coarsening algorithm is not used we will end up with non-contiguous active data in the lines to be solved, preventing us from using the CMSSL. We tried implementing data structure transformation routines, but they were found, as should be expected, to be clumsy and inefficient. We also implemented our own parallel tridiagonal solvers, but they were not very competitive, being about twice as slow as the CMSSL routine.

8.12.4 Coarsest Grid Solver  We tried a direct solver using LU factorization, but it turned out to be hard to implement and slow in its general form, unless the coarsest grid was always of a given fixed size. Instead, we chose to use the semi-coarsening algorithm in which case only a tridiagonal solver was needed. So now, the coarsening continues until only one line is left to solve, and that can be done by using the same tridiagonal solver that was used for the line solves of the smoother.

8.12.5 Grid Transfer Operators  There are three choices for the grid transfer operators, discussed in chapter 3, that were implemented. They are the ones discussed in sections 3.5.1, 3.5.3, and 3.6.1.

The two collapsing type methods were readily parallelizable and easily implemented. The computation of the grid transfer operator coefficients created a lot of temporary variables. It was difficult to find a good implementation that did not use too many temporaries and that could avoid having the compiler generate too many and fill up the available memory.
The grid transfer operators based on extensions to Schaffer's ideas were also parallelizable, but they depended on the availability of tridiagonal line solvers. The time to compute all the operators is also longer than the collapsing method because of the line solves.

8.12.6 Coarse Grid Operators  They are formed using the Galerkin coarse grid approximation using the grid transfer operators.

8.13 2D Semi–Coarsening Parallel Algorithm

The semi–coarsening code was originally implemented by Joel E. Dendy Jr., Michael Ida, and Jeff Rutledge on the CM-200. A better implementation was done by Bill Spangenberg, who wrote the Dynamic Memory Management Utilities (DMMU), on the CM-5. It is still possible to obtain an even better implementation of the semi-coarsening code on the CM-5, but this improvement has not been done because the code cannot be placed (at least not for the near future) in the public domain since it uses the proprietary DMMU.

8.13.1 Data Structures  The data structures are grid point stencil oriented with a different array data structure for the coefficients, the unknowns, and the right hand side.

8.13.2 Coarsening  Semi–coarsening in the y-direction was used, taking every other fine grid point in the y-directions to form the coarse grid.

Non-compatible grid data structures were used when \( VP > 1 \), and a compatible grid data structure otherwise, as was the case for the standard coarsening parallel code.
8.13.3 Smoothers  Red/black x–line Gauss–Seidel relaxation is used for the smoother. The CMSSL tridiagonal solver using cyclic reduction was used to solve the lines.

A better implementation for the line solves exists if block cyclic reduction is used when the finest grid level has $VP \gg 1$, since the subgrid size per processor will be large, and it makes more sense to use the vector units more efficiently by using sequential cyclic reduction on the subgrids of each processor.

8.13.4 Coarsest Grid Solver  The coarsening takes place until only one X grid line remains, and then the CMSSL tridiagonal solver is called to solve it exactly.

8.13.5 Grid Transfer Operators  The grid transfer operator is the one used in section 3.6.1 applied in only the y-direction. The CMSSL tridiagonal solver was also used.

8.13.6 Coarse Grid Operators  They are formed using the Galerkin coarse grid approximation using the grid transfer operators.

8.14 2D Parallel Timings

In the following tables we have reported both busy (B) and idle (I) times. Busy time is the execution time for the parallel processing nodes, while idle time is the sequential execution time and also the time to perform all communications. We are reporting times for various time shared partitions of the CM-5. The partitions are identified by the number of processing nodes (PN) namely, 32, 64, 128, 256, and 512 processing nodes. The CM-5 has a full configuration of 1024 processing nodes, but the full partition is not available under our time sharing system. The tables report timings, in seconds, for the average time of five runs for either the setup time or the average of
five V-cycles.

The standard coarsening timings are given in tables 8.3 and 8.4 for one $V(1,1)$-cycle and the setup respectively. We see the affects of the parallel overhead in the tables for small grids sizes and large partitions.

For a given partition we do not see the almost perfect scaling that is seen with the Cray Y-MP; a problem that is has four times the number of unknowns takes far less than four times the time. Nor do we see perfect scaleup with the number of processors; for the $1024 \times 1024$ case, the 128 takes about half the time of the 32 processor partition and the 512 processor partition takes two-thirds the time of the 128 processor partition.

We can also look at the parallel efficiency by examining the data for busy and idle times. The parallel efficiency for the standard coarsening algorithm is given in table 8.5. Note that the highest parallel efficiency is given for the largest grid size problem on the smallest number of processors. This should be expected since that combination produces the largest subgrid size per processor, which will be processed serially on each processor, keeping all the processors busy until the calculation is completed. We still see that the parallel efficiency ranges from 63 to 88, where the higher efficiencies are given for the larger grid sizes.

Tables 8.6 and 8.7 give timings for the semi-coarsening algorithm for the setup and one $V(1,1)$-cycle, respectively, for a range of grid sizes and processing node partitions.

The parallel timings for the semi-coarsening algorithm shows that we, again, do not have perfect scaling with the problem size nor do we have perfect scaleup with the number of processors. For the $1024 \times 1024$ case, the 128 processor partition takes about half the time of the 32 processor partition, and the 512 processor partition takes about two-thirds times the tie of the 128 processor partition.

The parallel efficiency for the semi-coarsening algorithm is given in table 8.8.
Table 8.3. Timings, in seconds, for the standard coarsening code performing one $V(1,1)$-cycle with zebra alternating line Gauss--Seidel on 32, 64, 128, 256, and 512 processing nodes of the CM-5, where the size N means an $N \times N$ fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
<th>256 PN</th>
<th>512 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>I</td>
<td>9.060E-2</td>
<td>1.113E-1</td>
<td>9.098E-2</td>
<td>1.039E-1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.844E-1</td>
<td>1.881E-1</td>
<td>1.892E-1</td>
<td>1.923E-1</td>
</tr>
<tr>
<td>16</td>
<td>I</td>
<td>1.050E-1</td>
<td>1.316E-1</td>
<td>1.232E-1</td>
<td>1.206E-1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>2.378E-1</td>
<td>2.778E-1</td>
<td>2.816E-1</td>
<td>2.844E-1</td>
</tr>
<tr>
<td>32</td>
<td>I</td>
<td>1.508E-1</td>
<td>1.348E-1</td>
<td>1.406E-1</td>
<td>1.752E-1</td>
</tr>
<tr>
<td>64</td>
<td>I</td>
<td>1.756E-1</td>
<td>2.090E-1</td>
<td>1.636E-1</td>
<td>1.828E-1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>3.558E-1</td>
<td>3.948E-1</td>
<td>4.076E-1</td>
<td>4.558E-1</td>
</tr>
<tr>
<td>128</td>
<td>I</td>
<td>1.794E-1</td>
<td>1.962E-1</td>
<td>1.828E-1</td>
<td>2.118E-1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4.520E-1</td>
<td>4.828E-1</td>
<td>4.774E-1</td>
<td>5.276E-1</td>
</tr>
<tr>
<td>256</td>
<td>I</td>
<td>2.346E-1</td>
<td>2.068E-1</td>
<td>2.060E-1</td>
<td>2.420E-1</td>
</tr>
<tr>
<td>512</td>
<td>I</td>
<td>2.132E-1</td>
<td>2.300E-1</td>
<td>2.542E-1</td>
<td>2.604E-1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.092E+0</td>
<td>9.188E-1</td>
<td>7.912E-1</td>
<td>7.822E-1</td>
</tr>
<tr>
<td>1024</td>
<td>I</td>
<td>3.240E-1</td>
<td>2.574E-1</td>
<td>2.676E-1</td>
<td>2.950E-1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>2.474E+0</td>
<td>1.731E+0</td>
<td>1.265E+0</td>
<td>1.132E+0</td>
</tr>
</tbody>
</table>
### Table 8.4. Timings, in seconds, for the setup phase of the standard coarsening code with zebra alternating line Gauss–Seidel on 32, 64, 128, 256, and 512 processing nodes of the CM-5, where the size N means an $N \times N$ fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
<th>256 PN</th>
<th>512 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>2.581E-1</td>
<td>4.332E-1</td>
<td>2.776E-1</td>
<td>4.141E-1</td>
<td>3.877E-1</td>
</tr>
<tr>
<td>B</td>
<td>3.454E-1</td>
<td>3.519E-1</td>
<td>3.530E-1</td>
<td>3.581E-1</td>
<td>3.627E-1</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>2.620E-1</td>
<td>4.833E-1</td>
<td>3.756E-1</td>
<td>3.067E-1</td>
<td>3.051E-1</td>
</tr>
<tr>
<td>B</td>
<td>4.880E-1</td>
<td>5.507E-1</td>
<td>5.556E-1</td>
<td>5.604E-1</td>
<td>5.710E-1</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>5.020E-1</td>
<td>3.419E-1</td>
<td>3.583E-1</td>
<td>5.578E-1</td>
<td>6.335E-1</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>5.616E-1</td>
<td>6.272E-1</td>
<td>4.143E-1</td>
<td>4.806E-1</td>
<td>4.620E-1</td>
</tr>
<tr>
<td>B</td>
<td>7.704E-1</td>
<td>8.378E-1</td>
<td>8.677E-1</td>
<td>9.574E-1</td>
<td>1.016E+0</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>5.480E-1</td>
<td>6.920E-1</td>
<td>5.000E-1</td>
<td>6.440E-1</td>
<td>5.590E-1</td>
</tr>
<tr>
<td>B</td>
<td>1.010E+0</td>
<td>1.044E+0</td>
<td>1.038E+0</td>
<td>1.135E+0</td>
<td>1.188E+0</td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>7.080E-1</td>
<td>5.750E-1</td>
<td>5.620E-1</td>
<td>6.490E-1</td>
<td>6.340E-1</td>
</tr>
<tr>
<td>B</td>
<td>1.450E+0</td>
<td>1.371E+0</td>
<td>1.293E+0</td>
<td>1.363E+0</td>
<td>1.372E+0</td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2.584E+0</td>
<td>2.096E+0</td>
<td>1.786E+0</td>
<td>1.728E+0</td>
<td>1.671E+0</td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1.198E+0</td>
<td>8.730E-1</td>
<td>8.420E-1</td>
<td>9.520E-1</td>
<td>8.090E-1</td>
</tr>
<tr>
<td>B</td>
<td>6.046E+0</td>
<td>4.113E+0</td>
<td>2.957E+0</td>
<td>2.528E+0</td>
<td>2.176E+0</td>
</tr>
</tbody>
</table>

### Table 8.5. Parallel efficiency for standard coarsening $V(1, 1)$-cycle using zebra alternating line Gauss–Seidel for the CM-5 with 32, 64, 128, 256, and 512 nodes. The results are given in percentages and N means an $N \times N$ grid.

<table>
<thead>
<tr>
<th>Size</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
<th>256 PN</th>
<th>512 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>67</td>
<td>63</td>
<td>68</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>16</td>
<td>69</td>
<td>68</td>
<td>70</td>
<td>70</td>
<td>65</td>
</tr>
<tr>
<td>32</td>
<td>66</td>
<td>71</td>
<td>71</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>64</td>
<td>56</td>
<td>58</td>
<td>62</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>128</td>
<td>52</td>
<td>58</td>
<td>67</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>256</td>
<td>73</td>
<td>75</td>
<td>82</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>512</td>
<td>84</td>
<td>80</td>
<td>76</td>
<td>75</td>
<td>73</td>
</tr>
<tr>
<td>1024</td>
<td>88</td>
<td>87</td>
<td>83</td>
<td>79</td>
<td>77</td>
</tr>
</tbody>
</table>

244
Table 8.6. Timings, in seconds, for the semi-coarsening code performing one $V(1,1)$-cycle on 32, 64, 128, 256, and 512 processing nodes of the CM-5, where the size $N$ means an $N \times N$ fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
<th>256 PN</th>
<th>512 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>5.036E-2</td>
<td>5.986E-2</td>
<td>5.830E-2</td>
<td>5.378E-2</td>
<td>5.758E-2</td>
</tr>
<tr>
<td>B</td>
<td>1.104E-1</td>
<td>1.125E-1</td>
<td>1.145E-1</td>
<td>1.175E-1</td>
<td>1.191E-1</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1.408E-1</td>
<td>1.525E-1</td>
<td>1.587E-1</td>
<td>1.631E-1</td>
<td>1.667E-1</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1.912E-1</td>
<td>1.912E-1</td>
<td>1.940E-1</td>
<td>2.090E-1</td>
<td>2.182E-1</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2.839E-1</td>
<td>2.582E-1</td>
<td>2.537E-1</td>
<td>2.580E-1</td>
<td>2.633E-1</td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>9.992E-2</td>
<td>1.083E-1</td>
<td>1.022E-1</td>
<td>1.095E-1</td>
<td>1.203E-1</td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1.058E-1</td>
<td>1.362E-1</td>
<td>1.337E-1</td>
<td>1.203E-1</td>
<td>1.330E-1</td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1.332E-1</td>
<td>1.510E-1</td>
<td>1.244E-1</td>
<td>1.343E-1</td>
<td>1.490E-1</td>
</tr>
<tr>
<td>B</td>
<td>2.653E+0</td>
<td>1.405E+0</td>
<td>1.102E+0</td>
<td>7.685E-1</td>
<td>6.932E-1</td>
</tr>
</tbody>
</table>
Table 8.7. Timings, in seconds, for the setup phase of the semi-coarsening code on 32, 64, 128, 256, and 512 processing nodes of the CM-5, where the size $N$ means an $N \times N$ fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
<th>256 PN</th>
<th>512 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>3.899E-2</td>
<td>2.441E-1</td>
<td>1.271E-1</td>
<td>2.317E-1</td>
<td>1.976E-1</td>
</tr>
<tr>
<td>16</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>5.260E-2</td>
<td>2.254E-1</td>
<td>1.156E-1</td>
<td>5.630E-2</td>
<td>5.710E-1</td>
</tr>
<tr>
<td></td>
<td>1.183E-1</td>
<td>1.173E-1</td>
<td>1.170E-1</td>
<td>1.204E-1</td>
<td>1.210E-1</td>
</tr>
<tr>
<td>32</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>6.940E-1</td>
<td>1.187E-1</td>
<td>7.480E-2</td>
<td>2.457E-1</td>
<td>3.146E-1</td>
</tr>
<tr>
<td></td>
<td>1.552E-1</td>
<td>1.646E-1</td>
<td>1.691E-1</td>
<td>1.679E-1</td>
<td>1.709E-1</td>
</tr>
<tr>
<td>64</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>2.066E-1</td>
<td>2.060E-1</td>
<td>2.113E-1</td>
<td>2.255E-1</td>
<td>2.321E-1</td>
</tr>
<tr>
<td>128</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>1.612E-1</td>
<td>1.393E-1</td>
<td>1.441E-1</td>
<td>1.013E-1</td>
<td>1.125E-1</td>
</tr>
<tr>
<td></td>
<td>3.004E-1</td>
<td>2.717E-1</td>
<td>2.693E-1</td>
<td>2.733E-1</td>
<td>2.817E-1</td>
</tr>
<tr>
<td>256</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>1.530E-1</td>
<td>1.560E-1</td>
<td>1.557E-1</td>
<td>1.154E-1</td>
<td>1.276E-1</td>
</tr>
<tr>
<td></td>
<td>5.006E-1</td>
<td>3.989E-1</td>
<td>3.811E-1</td>
<td>3.515E-1</td>
<td>3.502E-1</td>
</tr>
<tr>
<td>512</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>1.480E-1</td>
<td>1.623E-1</td>
<td>1.640E-1</td>
<td>1.274E-1</td>
<td>1.394E-1</td>
</tr>
<tr>
<td></td>
<td>1.025E+0</td>
<td>6.860E-1</td>
<td>6.029E-1</td>
<td>4.856E-1</td>
<td>4.650E-1</td>
</tr>
<tr>
<td>1024</td>
<td>I</td>
<td>B</td>
<td>I</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>1.610E-1</td>
<td>1.920E-1</td>
<td>1.710E-1</td>
<td>1.436E-1</td>
<td>1.570E-1</td>
</tr>
<tr>
<td></td>
<td>2.463E+0</td>
<td>1.476E+0</td>
<td>1.155E+0</td>
<td>8.036E-1</td>
<td>7.199E-1</td>
</tr>
</tbody>
</table>

Table 8.8. Parallel efficiency for semi-coarsening $V(1,1)$-cycle for the CM-5 with 32, 64, 128, 256, and 512 processing nodes. The results are given in percentages and $N$ means an $N \times N$ grid.

<table>
<thead>
<tr>
<th>Size</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
<th>256 PN</th>
<th>512 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>32 PN</td>
<td>64 PN</td>
<td>128 PN</td>
<td>256 PN</td>
<td>512 PN</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>59</td>
<td>53</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>16</td>
<td>67</td>
<td>65</td>
<td>66</td>
<td>69</td>
<td>67</td>
</tr>
<tr>
<td>32</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>68</td>
<td>66</td>
</tr>
<tr>
<td>64</td>
<td>72</td>
<td>62</td>
<td>74</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>128</td>
<td>77</td>
<td>74</td>
<td>73</td>
<td>72</td>
<td>71</td>
</tr>
<tr>
<td>256</td>
<td>83</td>
<td>78</td>
<td>78</td>
<td>75</td>
<td>73</td>
</tr>
<tr>
<td>512</td>
<td>90</td>
<td>83</td>
<td>81</td>
<td>79</td>
<td>77</td>
</tr>
<tr>
<td>1024</td>
<td>95</td>
<td>90</td>
<td>90</td>
<td>85</td>
<td>82</td>
</tr>
</tbody>
</table>
We again see that the highest parallel efficiency is obtained for the largest grid size problem on the least number of processors, and thus providing the largest subgrid for each processor to work on. The range of parallel efficiency values is now from 53 to 95. The increase in the values for the semi-coarsening algorithm over the standard coarsening algorithm is to be expected because the semi-coarsening algorithm keeps more processors busy during the smoother’s line solves, especially on the coarser grid levels.

We give a comparison for both the standard and semi-coarsening algorithms on the CM-5, Cray Y-MP, and a Sparc-5 workstation in table 8.9. The CM-5 timings are given for the fastest time for a given grid size and the processing partition that the time was obtained on are given in parentheses. The times in the table are the average time to complete one $V(1, 1)$-cycle for five $V$-cycles averaged over five separate runs.

The fastest times for a given grid size for the standard coarsening algorithm on the CM-5 are on the 32 processor partition for grid sizes up to $128 \times 128$, the 128 processor partition for $256 \times 256$ grid, and the 512 processor partition for grid sizes greater than or equal to $512 \times 512$. The semi-coarsening algorithm exhibits similar behavior where the 32 processor partition is the fastest for grid sizes up to $32 \times 32$, 128 processor partition for grid sizes between $64 \times 64$ and $128 \times 128$, the 256 processor partition for the $256 \times 256$ grid, and the 512 processor partition for grid sizes larger than $512 \times 512$.

Comparing the results from table 8.9 shows that vectorization plays an important role on the Cray Y-MP, even for the smallest problems. The Sparc-5 times show that the scaling argument holds, but that it is affected by caching issues. We see from the data that the Cray Y-MP time is still twice as fast as the 128 processor partition and 30% faster than the 512 processor partition. The Cray Y-MP codes are the fastest, but the CM-5 codes are catching up for large problems when enough processors are
Table 8.9. Timing comparison between the CM-5, Cray Y-MP, and Sparc-5 workstation for one $V(1,1)$-cycle in seconds, where $N$ means an $N \times N$ grid. The top entries are for the standard coarsening codes and the bottom entries are for the semi-coarsening codes, and * means that the problem was too big to fit into the available memory.

<table>
<thead>
<tr>
<th>Size</th>
<th>CM-5</th>
<th>Cray Y-MP</th>
<th>Sparc-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2.750E-1 (32)</td>
<td>5.270E-4</td>
<td>3.000E-4</td>
</tr>
<tr>
<td></td>
<td>1.083E-1 (512)</td>
<td>5.156E-4</td>
<td>5.000E-4</td>
</tr>
<tr>
<td>16</td>
<td>3.428E-1 (32)</td>
<td>1.016E-3</td>
<td>1.200E-2</td>
</tr>
<tr>
<td></td>
<td>1.640E-1 (32)</td>
<td>9.365E-4</td>
<td>1.000E-2</td>
</tr>
<tr>
<td>32</td>
<td>4.410E-1 (32)</td>
<td>2.019E-3</td>
<td>5.000E-2</td>
</tr>
<tr>
<td></td>
<td>2.080E-1 (32)</td>
<td>1.896E-3</td>
<td>3.200E-2</td>
</tr>
<tr>
<td>64</td>
<td>5.314E-1 (32)</td>
<td>4.579E-3</td>
<td>1.910E-1</td>
</tr>
<tr>
<td></td>
<td>2.608E-1 (128)</td>
<td>4.435E-3</td>
<td>1.383E-1</td>
</tr>
<tr>
<td>128</td>
<td>6.314E-1 (32)</td>
<td>1.285E-2</td>
<td>7.980E-1</td>
</tr>
<tr>
<td></td>
<td>3.463E-1 (128)</td>
<td>1.325E-2</td>
<td>5.988E-1</td>
</tr>
<tr>
<td>256</td>
<td>7.918E-1 (128)</td>
<td>4.429E-2</td>
<td>3.514E+0</td>
</tr>
<tr>
<td></td>
<td>4.416E-1 (256)</td>
<td>4.320E-2</td>
<td>2.666E+0</td>
</tr>
<tr>
<td>512</td>
<td>1.037E+0 (512)</td>
<td>1.654E-1</td>
<td>1.503E+1</td>
</tr>
<tr>
<td></td>
<td>5.795E-1 (512)</td>
<td>1.576E-1</td>
<td>1.209E+1</td>
</tr>
<tr>
<td>1024</td>
<td>1.427E+0 (512)</td>
<td>6.732E-1</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>9.028E-1 (512)</td>
<td>6.563E-1</td>
<td>*</td>
</tr>
</tbody>
</table>
available. When scaling is applied to take into account the difference in clock speeds and instructions per clock cycle between the Cray Y-MP and the CM-5, we see that the two are nearly identical for the $1024 \times 1024$ problem, but that the Cray Y-MP still has a very slight edge. This shows that the CM-5 codes not only suffer from the overhead associated with parallelization, but that the communication issues are the main bottleneck to beating the vector codes.
CHAPTER 9

BLACK BOX MULTIGRID IN THREE DIMENSIONS

9.1 Introduction

The development of a three dimensional black box multigrid solver essentially involves just extending the two dimensional version. The three dimensional methods provide the same functionality as the two dimensional black box multigrid methods. The basic multigrid algorithm and the multigrid components are essentially the same, except that the standard coarsening methods need alternating red/black plane relaxation to obtain a robust smoother. In addition, there are several changes in the implementation, especially for the parallel code. The 3D parallel methods use (alternating) red/black plane Gauss-Seidel relaxation, where the required plane solves are performed using a 2D multigrid method, which have been modified to solve all the planes of a single color simultaneously.

We will examine both the standard and semi-coarsening black box multigrid algorithms for problems in three dimensions. The examination will include the 3D algorithm implementations on vector (Cray Y-MP) and parallel (CM-5) computers.

The grid operator stencil in three dimensions is now assumed to fit into the 27-point cubic stencil. The 27-point stencil is illustrated in figure 9.1. Notice that for each fixed z (xy-plane) we use the same compass coefficient notation that were used for two dimensions with a prefix to indicate the z level index of the stencil. For the stencil at grid point \((i, j, k)\), the three prefixes are \(t\) for \((*, *, k+1)\), \(p\) for \((*, *, k)\), and
$b$ for $(*, *, k - 1)$.

9.1.1 Semi–Coarsening The semi-coarsening algorithm can be done in several ways. Recall that the semi-coarsening method used a smoother working orthogonal to the direction of the coarsening. The coarsening can be done in one of the coordinate directions, leaving the smoother to work on planes, or the coarsening can be done in two of the coordinate directions with the smoother working on lines. We have chosen to examine only semi-coarsening in the $z$ coordinate direction. Either of the other two coordinate direction would have been equally valid, but since we plan on using the 2D semi-coarsening algorithm to perform the plane solves, which is already written for xy-planes, we can avoid writing additional versions for the other planes.
Figure 9.1: Grid operator stencil in three dimensions.
CHAPTER 10

3D DISCRETIZATIONS

This chapter presents some of the discretizations that can be used on the convection–diffusion equation in three dimensions. The finite difference and finite volume discretizations in three dimensions are straightforward extensions of the two dimensional discretizations presented in chapter 2. We will present only a few examples in three dimensions.

The continuous three dimensional problem is given by

\[-\nabla \cdot (D \cdot \nabla u) + b \cdot \nabla u + cu = f, \quad (x, y) \in \Omega = (0, M_x) \times (0, M_y) \times (0, M_z) \quad (10.1)\]

where \(D\) is a \(3 \times 3\) tensor, \(\det D > 0\), and \(c \geq 0\). We will only be considering problems where \(D\) is diagonal in this chapter. In addition, \(D, c,\) and \(f\) are allowed to be discontinuous across internal interfaces in the domain \(\Omega\). The boundary conditions are given by

\[\frac{\partial u}{\partial n} + a u = g, \quad \text{on } \partial \Omega \quad (10.2)\]

where \(a\) and \(g\) are functions, and \(n\) is the outward unit normal vector. This allows us to represent Dirichlet, Neumann, and Robin boundary conditions.

The domain is assumed to be a rectangular parallelepiped, \(\Omega = (0, M_x) \times (0, M_y) \times (0, M_z)\), which is divided into uniform cells of length \(h_x = M_x / N_x\) by \(h_y = M_y / N_y\), by \(h_z = M_z / N_z\), where \(N_x, N_y,\) and \(N_z\) are the number of cells in the x–, y–, and z–directions respectively. The mesh need not be uniform, but such an assumption will simplify our discussions.
A finite element discretization on a regular tetrahedral mesh can also be used to derive the discrete system of equations which can be used for input to the black box multigrid methods.

10.1 Finite Difference Discretization

The anisotropic Poisson’s equation on a cube domain,

\[ Lu = -\varepsilon_x u_{xx} - \varepsilon_y u_{yy} - \varepsilon_z u_{zz} = f \quad \text{in } \Omega = (0,1)^3 \quad (10.3) \]

where \( u \) and \( f \) are functions of \( x, y, \) and \( z \), can be discretized by central finite differences with a uniform grid spacing, \( h = 1/N \) for \( N = n_x = n_y = n_z \), to get the 7-point stencil at grid point \((i,j,k)\):

\[
L^h \approx \frac{1}{h^3} \begin{bmatrix}
0 & -\varepsilon_y & 0 \\
-\varepsilon_x & 2(\varepsilon_x + \varepsilon_y + \varepsilon_z) & -\varepsilon_x \\
0 & -\varepsilon_y & 0 \\
\end{bmatrix}
\begin{bmatrix}
b_i \\
p_i \\
t_i \\
\end{bmatrix},
\]

where the stencil subscripts b, p, and t are short for the \( k - 1 \), \( k \), and \( k + 1 \) stencil planes respectively.

10.2 Finite Volume Discretization

There are several finite volume grids that can be used for discretization, but the two most common are the vertex and cell centered grid, just as in two dimensions. We will present only the finite volume discretization for the vertex centered finite volumes,

\[
G_v = \left\{ (x_i, y_j, z_k) : \begin{array}{l}
x_i = i h_x, \quad i = 0, \ldots, N_x, \\
y_j = j h_y, \quad j = 0, \ldots, N_y, \\
z_k = k h_z, \quad k = 0, \ldots, N_z
\end{array} \right\}, \qquad (10.5)
\]
with evaluation at the vertices. The discretization is best when the discontinuous interfaces align with the finite volume boundaries (surfaces). In this discretization \( D \), \( c \), and \( f \) are approximated by constant values in finite volume, \( \Omega_{i,j} \), whose centers are at the vertices.

**10.2.1 Interior Finite Volumes** The development is done the same as in chapter 2. However, instead of having four line integrals, we now have six surface integrals to evaluate over the finite volume. We will refer to the six surfaces as \( \Omega_{i-1} \), \( \Omega_{i+1} \), \( \Omega_{j-1} \), \( \Omega_{j+1} \), \( \Omega_{k-1} \), and \( \Omega_{k+1} \), where the subscripts indicate the fixed grid index.

The surface integral for \( \Omega_{i-1} \) is

\[
\int_{\Omega_{i-1}} D_x \frac{\partial u}{\partial x} dydz \approx \frac{h_y h_z}{h_x} \frac{2 D_{x,i,j,k} D_{x,i-1,j,k}}{D_{x,i,j,k} + D_{x,i-1,j,k}} (u_{i,j,k} - u_{i-1,j,k}).
\] (10.6)

The other five surface integrals are approximated similarly. The volume integrals are

\[
\int_V c udV \approx h_x h_y h_z c_{i,j,k}
\] (10.7)

and

\[
\int_V f dV \approx h_x h_y h_z f_{i,j,k}.
\] (10.8)
The stencil for grid point \((i,j,k)\) is given by

\[
\begin{bmatrix}
0 \\
0 - \frac{h_x h_z}{h_x} \alpha_{i,j,k} \\
0 \\
- \frac{h_x h_y}{h_z} \alpha_{i,j,k} \\
- \frac{h_x h_z}{h_y} \alpha_{i,j-1,k} \\
\end{bmatrix}
\begin{bmatrix}
\alpha_{i,j,k} \\
c_{i,j,k} \\
\alpha_{i,j-1,k} \\
\alpha_{i,j,k-1} \\
0 \\
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

(10.9)

where

\[
\alpha_{i,j,k} = \frac{2 D_{x,i,j,k} D_{x,i-1,j,k}}{D_{x,i,j,k} + D_{x,i-1,j,k}}
\]

(10.10)

\[
\alpha_{i,j,k} = \frac{2 D_{y,i,j,k} D_{y,i,j-1,k}}{D_{y,i,j,k} + D_{y,i,j-1,k}}
\]

(10.11)

\[
\alpha_{i,j,k} = \frac{2 D_{z,i,j,k} D_{z,i,j,k-1}}{D_{z,i,j,k} + D_{z,i,j,k-1}}
\]

(10.12)

and

\[
\sum = \frac{h_y h_z}{h_x} (\alpha_{i-1,j,k} + \alpha_{i,j,k}) + \frac{h_x h_z}{h_y} (\alpha_{i,j-1,k} + \alpha_{i,j,k}) + \frac{h_x h_y}{h_z} (\alpha_{i,j,k-1} + \alpha_{i,j,k})
\]

(10.13)

10.2.2 Edge Boundary Finite Volumes

Let the finite volume \(\Omega_{i,j,k}\) have its southern edge at the southern boundary \((y = 0)\) of the domain.
10.2.3 Dirichlet Boundary Condition \quad For the Dirichlet boundary condition we have \( u_{(s)} = g_{(s)} \), so that the surface integral over \( \Omega_{i,j-1,k} \) is

\[
\int_{\Omega_{i,j-1}} D_y \frac{\partial u}{\partial y} dx dz = 2h_x h_z D_y \frac{\partial u}{\partial y} (u_{i,j,k} - u_{(s)})
\]  

(10.14)

where \( u_{(s)} \) means to evaluate \( u \) at the grid point \((i, j - 1, k)\). We now get the stencil

\[
\begin{bmatrix}
0 \\
0 - \frac{h_z h_y}{h_x} \alpha_{i,j,k} \\
0 \\
- \frac{h_y h_z}{h_x} \alpha_{i,j,k} \\
- \frac{h_y h_z}{h_x} \alpha_{i-1,j,k} \\
\sum_h + h_x h_y h_z c_{i,j,k} - \frac{h_x h_y}{h_z} \alpha_{i,j,k} \\
0 \\
0 - \frac{h_x h_y}{h_z} \alpha_{i,j,k-1} \\
0
\end{bmatrix}
\]

(10.15)

where

\[
\sum_h = \frac{h_y h_z}{h_x} (\frac{\alpha_{i-1,j,k}}{\alpha_{i,j,k}} + \frac{\alpha_{i,j,k}}{\alpha_{i,j,k}}) + \frac{h_x h_z}{h_y} \alpha_{i,j,k-1} + \frac{h_x h_y}{h_z} \alpha_{i,j,k-1} + \alpha_{i,j,k} 
\]

(10.16)

and the \( \alpha \)'s are given as before; see equations (10.10), (10.11), and (10.12).

10.2.4 Neumann and Robin Boundary Conditions \quad The boundary condition along the southern boundary is

\[
\left( \frac{\partial u}{\partial n} + a \ u \right)_{(s)} = g_{(s)}.
\]  

(10.17)

We can make the approximation

\[
\left. \frac{\partial u}{\partial n} \right|_{(s)} = g_{(s)} - a_{(s)} u_{(s)}
\]  

(10.18)
\[ u(s) - u(p) = \frac{h_x}{2} (g(s) - a(s)u(s)), \text{ which gives} \]

\[ u(s) = \frac{\frac{h_x}{2} g(s) - u(p)}{1 + \frac{h_x}{2} a(s)}. \]  

The surface integral along the boundary is approximated by

\[
\int_{\Omega_{j-1}} D_y \frac{\partial u}{\partial y} dx dz \approx h_x h_z D_y,i,j,k \frac{\partial u}{\partial y} |_{(s)}
\]

\[ \approx -h_x h_z D_y,i,j,k \left( g(s) - a(s)u(s) \right) \]

\[ \approx \frac{2h_x h_z}{2 + h_y a(s)} D_y,i,j,k \left( a(s) u_{i,j,k} - g(s) \right). \]  

We now get the stencil

\[
\begin{bmatrix}
0 \\
0 - \frac{h_z h_y}{h_x} \alpha^x_{i,j,k} 0 \\
0 \\
- \frac{h_x h_y}{h_z} \alpha^y_{i,j,k} \\
\end{bmatrix} + h_x h_y h_z c_{i,j,k} + \tilde{B} C - \frac{h_y h_z}{h_x} \alpha^x_{i,j,k}
\]

\[
\begin{bmatrix}
0 \\
0 \\
0 - \frac{h_z h_y}{h_x} \alpha^x_{i,j,k-1} 0 \\
0 \\
\end{bmatrix} = 0
\]  

258
where where \( \sum \) is defined in equation (10.16), the \( \alpha \)'s are defined by equations (10.10), (10.11), and (10.12), and

\[
\bar{BC} = \frac{2h_x h_z a(s)}{2 + h_y a(s)} D_{y,i,j,k}.
\] (10.24)

The other boundary finite volume cases (faces, edges, and corners) can be easily deduced from the previous boundary conditions cases above.
CHAPTER 11

3D NONSYMMETRIC: GRID TRANSFER OPERATORS

The three dimensional grid transfer operators are the same as those used for the two dimensional grid transfer operators, except that the grid operators $L^h$ and $L^H$ now have 27-point stencils.

The three dimensional grid transfer coefficients are computed using the same type of grid decomposition method as were used in the second method of [29]. The computational method involves the formation of the grid transfer coefficients and the coarse grid operator by operator induced interpolation and Galerkin coarse grid approximation by performing consecutive semi-coarsening in each of the coordinate directions.

The grid transfer coefficients are computed by an extension of the same methods that were used for the two dimensional grids, that is, the collapsing methods and the extension of Schaffer’s ideas; see sections 3.5.1 through 3.6. The only difference is that instead of $A_{i-1}$, $A_{i+1}$, etc. representing points and lines respectively, they now represent points and planes.

The three dimensional grid transfer operator stencil is a little more complex than the two dimensional ones; see figure 11.1.

The computations of the grid transfer coefficients become quite clear if one draws several pictures; then the symmetry of the computations really stands out. The pictures are not presented here because they are hard to represent in a static
Figure 11.1: Grid transfer operator’s stencil in three dimensions.
monochrome mode.

11.1 3D Grid Transfer Operations

The fine grid points that are also coarse grid points use the identity as the interpolation operator. The coarse grid correction is then given by

\[ u^h_{ij} \leftarrow u^h_{ij} + u^H_{i,j-1,k} \]  

where \((x_{i,j}, y_{j,i}, z_{k}) = (x_{i,c}, y_{j,c}, z_{k,c})\) on the grid; here the interpolation coefficient is 1.

The fine grid points that are between two coarse grid points that share the same \(y_j\) and \(z_k\) coordinates use a two point relation for the interpolation. The coarse grid correction is given by

\[ u^h_{i-1,j,k} \leftarrow u^h_{i,j,k} + I_{i-1,j,c,k} \cdot u^H_{i,j,c,k} + I_{i,j-1,c,k} \cdot u^H_{i,j,c,k} \]  

where \(x_{i,c} = x_{i,f} < x_{i,c-1} < x_{i,c}, y_{j,c} = y_{j,f},\) and \(z_{k,c} = z_{k},\) on the grid, and the interpolation coefficients are \(I_{i-1,j,c,k}\) and \(I_{i,j-1,c,k}\).

The fine grid points that are between two coarse grid points that share the same \(x_i\) and \(z_k\) coordinates use a similar two point relation for the interpolation. The coarse grid correction is then given by

\[ u^h_{i,j-1,k} \leftarrow u^h_{i,j-1,k} + I_{i,j-1,c,k} \cdot u^H_{i,j,c,k} + I_{i-1,j,c,k} \cdot u^H_{i-1,j,c,k} \]  

where \(x_{i,c} = x_{i,f}, y_{j,c-1} < y_{j,c-1} < y_{j,c},\) and \(z_{k,c} = z_{k},\) on the grid, and the interpolation coefficients are \(I_{i-1,j,c,k}\) and \(I_{i-1,j,c,k}\).

The fine grid points that are between two coarse grid points that share the same \(x_i\) and \(y_j\) coordinates use a similar two point relation for the interpolation. The coarse grid correction is then given by

\[ u^h_{i,j-1,k} \leftarrow u^h_{i,j-1,k} + I_{i,j-1,c,k} \cdot u^H_{i,j,c,k} + I_{i-1,j,c,k} \cdot u^H_{i-1,j,c,k} \]  

where \(x_{i,c} = x_{i,f}, y_{j,c-1} < y_{j,c-1} < y_{j,c}\) on the grid, and the interpolation coefficients are \(I_{i-1,j,c,k}\) and \(I_{i-1,j,c,k}\).

The fine grid points that are between two coarse grid points that share the same \(x_i\) and \(z_k\) coordinates use a similar two point relation for the interpolation. The coarse grid correction is then given by

\[ u^h_{i,j,k-1} \leftarrow u^h_{i,j,k-1} + I_{i,j,k-1,c} \cdot u^H_{i,j,c,k} + I_{i,j,k-1,c} \cdot u^H_{i,j,c,k} \]  

where \(x_{i,c} = x_{i,f}, y_{j,c-1} < y_{j,c-1} < y_{j,c},\) and \(z_{k,c} = z_{k},\) on the grid, and the interpolation coefficients are \(I_{i,j-1,c,k}\) and \(I_{i,j-1,c,k}\).
where \(x_{i_c} = x_{i_f}, y_{j_c} = y_{j_f}\), and \(z_{k_c-1} < z_{k_f-1} < z_{k_c}\) on the grid, and the interpolation coefficients are \(I_{i_c,j_c,k_c-1}^{zzz}\) and \(I_{i_c,j_c,k_c}^{zmn}\).

For the fine grid points that share \(z_k\) coordinates, but do not share either a \(x_i\) or a \(y_j\) coordinate with the coarse grid, we use a four point relation for the interpolation, and the coarse grid correction is given by

\[
\begin{align*}
\frac{u_i^h}{l_{i_f-1,j_f-1,k_f}} & \leftarrow \frac{u_i^h}{l_{i_f-1,j_f-1,k_f}} \\
& + I_{i_c-1,j_c-1,k_c}^{yzsw} \cdot u^H_{i_c-1,j_c-1,k_c} + I_{i_c-1,j_c,k_c}^{yznw} \cdot u^H_{i_c-1,j_c,k_c} \\
& + I_{i_c,j_c,k_c}^{zne} \cdot u^H_{i_c,j_c,k_c} + I_{i_c,j_c-1,k_c}^{zse} \cdot u^H_{i_c,j_c-1,k_c},
\end{align*}
\] (11.5)

where \(x_{i_c} < x_{i_f} < x_{i_c+1}, y_{j_c} < y_{j_f} < y_{j_c+1}\), and \(z_{k_c} = z_{k_f}\), with interpolation coefficients \(I_{i_c-1,j_c-1,k_c}^{yzsw}, I_{i_c-1,j_c,k_c}^{yznw}, I_{i_c,j_c,k_c}^{zne}\), and \(I_{i_c,j_c-1,k_c}^{zse}\).

For the fine grid points that share \(x_i\) coordinates, but do not share either a \(y_j\) or a \(z_k\) coordinate with the coarse grid, we use a four point relation for the interpolation, and the coarse grid correction is given by

\[
\begin{align*}
\frac{u_i^h}{l_{i_f,j_f-1,k_f-1}} & \leftarrow \frac{u_i^h}{l_{i_f,j_f-1,k_f-1}} \\
& + I_{i_c,j_c-1,k_c-1}^{yzsw} \cdot u^H_{i_c,j_c-1,k_c-1} + I_{i_c,j_c-1,k_c}^{yznw} \cdot u^H_{i_c,j_c-1,k_c} \\
& + I_{i_c,j_c,k_c}^{zne} \cdot u^H_{i_c,j_c,k_c} + I_{i_c,j_c,k_c-1}^{zse} \cdot u^H_{i_c,j_c,k_c-1},
\end{align*}
\] (11.6)

where \(x_{i_c} = x_{i_f}, y_{j_c} < y_{j_f} < y_{j_c+1}\), and \(z_{k_c} < z_{k_f} < z_{k_c+1}\), with interpolation coefficients \(I_{i_c,j_c-1,k_c-1}^{yzsw}, I_{i_c,j_c-1,k_c}^{yznw}, I_{i_c,j_c,k_c}^{zne}\), and \(I_{i_c,j_c,k_c-1}^{zse}\).

For the fine grid points that share \(y_j\) coordinates, but do not share either a \(x_i\) or a \(z_k\) coordinate with the coarse grid, we use a four point relation for the interpolation, and the coarse grid correction is given by

\[
\begin{align*}
\frac{u_i^h}{l_{i_f-1,j_f,k_f-1}} & \leftarrow \frac{u_i^h}{l_{i_f-1,j_f,k_f-1}} \\
& + I_{i_c-1,j_c,k_c-1}^{yzsw} \cdot u^H_{i_c-1,j_c,k_c-1} + I_{i_c-1,j_c,k_c}^{yznw} \cdot u^H_{i_c-1,j_c,k_c} \\
& + I_{i_c,j_c,k_c}^{zne} \cdot u^H_{i_c,j_c,k_c} + I_{i_c,j_c,k_c-1}^{zse} \cdot u^H_{i_c,j_c,k_c-1},
\end{align*}
\] (11.7)

263
where \( x_{i_c} < x_{i_f} < x_{i_c+1}, y_{j_c} = y_{j_f}, \) and \( z_{k_c} < z_{k_f} < z_{k_c+1} \), with interpolation coefficients \( I^{xxw} \), \( I^{xyw} \), \( I^{xzw} \), \( I^{xne} \), \( I^{xxs} \), \( I^{xws} \), \( I^{yws} \), and \( I^{zws} \).

Lastly, for the fine grid points that do not share either a \( x_i \), \( y_j \), or \( z_k \) coordinates with the coarse grid, we use an eight point relation for the interpolation, and the coarse grid correction formula is given by

\[
\begin{align*}
\hat{u}^h_{i_f,j_f,k_f} &= \hat{u}^h_{i_f,j_f,k_f} \\
&+ I^{bw}_{i-1,j_c-1,k_c} \cdot u^H_{i-1,j_c-1,k_c} + I^{bw}_{i-1,j_c,k_c} \cdot u^H_{i-1,j_c,k_c} \\
&+ I^{bne}_{i-1,j_c-1,k_c} \cdot u^H_{i-1,j_c-1,k_c} + I^{bne}_{i-1,j_c,k_c} \cdot u^H_{i-1,j_c,k_c} \\
&+ I^{bw}_{i-1,j_c-1,k_c-1} \cdot u^H_{i-1,j_c-1,k_c-1} + I^{bw}_{i-1,j_c,k_c-1} \cdot u^H_{i-1,j_c,k_c-1} \\
&+ I^{bse}_{i-1,j_c-1,k_c-1} \cdot u^H_{i-1,j_c-1,k_c-1} + I^{bse}_{i-1,j_c,k_c-1} \cdot u^H_{i-1,j_c,k_c-1}.
\end{align*}
\]

where \( x_{i_c} < x_{i_f} < x_{i_c+1}, y_{j_c} < y_{j_f} < y_{j_c+1}, \) and \( z_{k_c} < z_{k_f} < z_{k_c+1} \), with interpolation coefficients \( I^{bw} \), \( I^{bw}_{i-1,j_c-1,k_c} \), \( I^{bw}_{i-1,j_c,k_c} \), \( I^{bne} \), \( I^{bse} \), \( I^{bw}_{i-1,j_c-1,k_c-1} \), \( I^{bw}_{i-1,j_c,k_c-1} \), \( I^{bse}_{i-1,j_c-1,k_c-1} \), \( I^{bse}_{i-1,j_c,k_c-1} \), \( I^{bw} \), \( I^{bse} \), \( I^{bse}_{i-1,j_c-1,k_c-1} \), and \( I^{bse}_{i-1,j_c,k_c-1} \).

The prolongation operators also have a correction term, containing the residual, added to them to obtain an \( O(h^2) \) error at the boundaries. The correction is similar to the one employed in the two dimensional case; see 3.1.1.

### 11.2 3D Nonsymmetric Grid Operator \( L^h \): Collapsing Methods

To illustrate the grid transfer operators, we will present the nonsymmetric collapsing method in three dimensions. From this discussion it should be clear how to extend the other grid transfer operators from two to three dimensions.

The \( I_{xyw} \) coefficient is computed by

\[
I_{xyw} = \frac{\sigma[West]}{\sigma N_b + \sigma C_b + \sigma S_b + \sigma N_p + \sigma C_p + \sigma S_p + \sigma N_t + \sigma C_t + \sigma S_t} \quad (11.9)
\]
If, however, $R_{\Sigma}$ is small, then

$$I_{xyw} = \frac{\sigma[West]}{\sigma[West] + \sigma[East]}$$

(11.10)

where

$$\sigma[West] = \sigma NW_t + \sigma W_t + \sigma SW_t$$
$$+ \sigma NW_p + \sigma W_p + \sigma SW_p$$
$$+ \sigma NW_b + \sigma W_b + \sigma SW_b,$$  

(11.11)

and

$$\sigma[East] = \sigma NE_t + \sigma E_t + \sigma SE_t$$
$$+ \sigma NE_p + \sigma E_p + \sigma SE_p$$
$$+ \sigma NE_b + \sigma E_b + \sigma SE_b,$$  

(11.12)

are the west and east planes of the grid operator stencil. In (11.9)-(11.12), $Iw$ is evaluated at $(x_{i-1}, y_{j}, z_{k})$, and the other coefficients on the right hand side are evaluated at $(x_{i-1}, y_{j+1}, z_{k})$ for the $L^h$ components. Let

$$\gamma = \min \{|\sigma[West]|, |\sigma[East]|, 1.\}.$$  

(11.13)

Then by small we mean that

$$R_{\Sigma} < -\gamma \cdot \sigma [R_{\Sigma} - C_p],$$

(11.14)

where

$$R_{\Sigma} = SW_t + S_t + SE_t + W_t + C_t + E_t + NW_t + N_t + NE_t$$
$$+ SW_p + S_p + SE_p + W_p + C_p + E_p + NW_p + N_p + NE_p$$
$$+ SW_b + S_b + SE_b + W_b + C_b + E_b + NW_b + N_b + NE_b.$$  

(11.15)
Prolongation coefficients which are computed in a similar way are $I_{xye}$, $I_{xys}$, $I_{xyn}$, $I_{xzn}$, and $I_{xz}$.

The prolongation coefficients $I_{xyw}$, $I_{xye}$, $I_{xys}$, $I_{xyse}$, $I_{xzn}$, $I_{yzn}$, $I_{yze}$, $I_{yse}$, and $I_{yse}$ can all be computed in a similar fashion. The computation of these coefficients is analogous to the computation of $I_{nw}$, $I_{ne}$, $I_{sw}$, and $I_{se}$ from section 3.5.1. We now illustrate these computations by computing the prolongation coefficient $I_{xyw}$.

\[
I_{xyw} = \frac{\sigma[NWzLine] + \sigma[NzLine] \cdot I_{xyw} + \sigma[WzLine] \cdot I_{xyn}}{C_p} \quad (11.16)
\]

where

\[
\sigma[NzLine] = \sigma N_t + \sigma N_p + \sigma N_b, \quad (11.17)
\]
\[
\sigma[WzLine] = \sigma W_t + \sigma W_p + \sigma W_b, \quad (11.18)
\]
\[
\sigma[NWzLine] = \sigma NW_t + \sigma NW_p + \sigma NW_b, \quad (11.19)
\]

and where the notation means to take the line in the $z$–direction that contains the given grid operator coefficients. If, however, $R_{\Sigma}$ is small, then

\[
I_{xyw} = \frac{\sigma[NWzLine] + \sigma[NzLine] \cdot I_{xyw} + \sigma[WzLine] \cdot I_{xyn}}{\sigma[R_{\Sigma} - C_p]} \quad (11.20)
\]

Let

\[
\gamma = \min \{|\sigma[NWzLine]|, |\sigma[NzLine]|, |\sigma[NEzLine]|, |\sigma[WzLine]|, |\sigma[SWzLine]|, |\sigma[SzLine]|, |\sigma[SEzLine]|, 1\}.
\]

Then by small we mean that

\[
R_{\Sigma} < -\gamma \cdot \sigma \cdot |R_{\Sigma} - C_p|, \quad (11.22)
\]

266
where $R_\Sigma$ is defined in equation (11.15). $I_{xyw}$, $I_{xy}$ and $I_{yn}$ are evaluated at $(x_{i-1}, y_{j}, z_{k_\Sigma})$ and $\sigma[NzLine]$, $\sigma[WzLine]$, and $\sigma[NWzLine]$ are evaluated at $(x_{i-1}, y_{j-1}, z_{k_f})$.

Finally, the last eight prolongation coefficients, $I_{tnw}$, $I_{tn}$, $I_{tsw}$, $I_{ts}$, $I_{bnw}$, $I_{bne}$, $I_{bsw}$, and $I_{bse}$, are used to interpolate to fine grid points which do not align with any of the coarse grid lines. They can all be computed in a similar fashion, which will be illustrated for $I_{tnw}$.

$$I_{tnw} = \sigma N W_t + \sigma W_p \cdot I_{yzne} + \sigma N W_p \cdot I_{xzn} + \sigma N_p \cdot I_{xznw} + \sigma W_t \cdot I_{yn} + \sigma N_t \cdot I_{wy} + \sigma C_t \cdot I_{wnw}$$

$$\frac{C_p}{\sigma [R_\Sigma - C_p]}.$$  \hspace{1cm} (11.23)

If, however, $R_\Sigma$ is small, then

$$I_{tnw} = \sigma N W_t + \sigma W_p \cdot I_{yzne} + \sigma N W_p \cdot I_{xzn} + \sigma N_p \cdot I_{xznw} + \sigma W_t \cdot I_{yn} + \sigma N_t \cdot I_{wy} + \sigma C_t \cdot I_{wnw}$$

$$\frac{\sigma [R_\Sigma - C_p]}{\sigma [R_\Sigma - C_p]}.$$  \hspace{1cm} (11.24)

Let

$$\gamma = \min \{|\sigma[West]|, |\sigma[North]|, |\sigma[East]|,$$

$$|\sigma[South]|, |\sigma[Top]|, |\sigma[Bottom]|, 1\}. \hspace{1cm} (11.25)$$

Then by small we mean that

$$R_\Sigma < -\gamma \cdot \sigma [R_\Sigma - C_p]$$

(11.26)

where $R_\Sigma$ is defined in equation (11.15). $I_{yzne}$, $I_{xzn}$, $I_{xznw}$, $I_{ynw}$, $I_{xyw}$, and $I_{yn}$ are evaluated at $(x_{i-1}, y_{j}, z_{k_\Sigma-1})$ and $\sigma[West]$, $\sigma[North]$, $\sigma[East]$, $\sigma[South]$, $\sigma[Top]$, and $\sigma[Bottom]$ are evaluated at $(x_{i-1}, y_{j-1}, z_{k_f-1})$. Note that, $\sigma[Top]$ and $\sigma[Bottom]$ are just the sum of the grid operator coefficients on the top ($k+1$) and bottom ($k-1$) planes of the grid operator stencil respectively.
The restriction operator coefficients are computed in the same way as above, but instead of using the symmetric part of the grid operator, $\sigma L$, we use the transpose of the grid operator, $L^T$.

### 3D Grid Transfer Operator Variations
With the information above on how to compute the basic grid transfer operators, it is easy to see how to extend all of the grid transfer operator variations, that we discussed in chapter 3, from 2D to 3D.

#### 3D Coarse Grid Operator
The three dimensional coarse grid operator is computed in the same way as the second method in [29]. The computational method involves the formation of the grid transfer coefficients and the coarse grid operator using auxiliary grids. The coarse grid operator is formed in a series of steps using a series of semi-coarsening auxiliary grids.

Define an auxiliary grid $\tilde{G}_z^H = G_x^h \times G_y^h \times G_z^H$, which is just the grid $G^h$ coarsened in the z-direction only. Now we define the grid transfer operator, $(J^h_{H})_z : \tilde{G}_z^H \rightarrow G^h$. The grid transfer operator, $(J^h_{H})_z$, can be constructed using any of the methods discussed, as can the other two grid transfer operators discussed. We now define the partial coarse grid operator to be

$$(L^H)_z = \left( (J^h_{H})_z \right)^T L^h (J^h_{H})_z. \tag{11.27}$$

In a similar fashion we define $\tilde{G}_{yz}^H = G_x^h \times G_y^H \times G_z^H$, and the grid transfer operator $(J^h_{H})_{yz} : \tilde{G}_{yz}^H \rightarrow \tilde{G}_z^H$. The associated coarse grid operator is defined by

$$(L^H)_{yz} = \left( (J^h_{H})_{yz} \right)^T (L^H)_z (J^h_{H})_{yz}. \tag{11.28}$$

Finally, in a similar fashion, we define the coarse grid, $G^H = G_x^H \times G_y^H \times G_z^H$, and the
grid transfer operator \( J^h_H : G^H \rightarrow C^H_{yz} \). The coarse grid operator is finally obtained by

\[
L^H = (J^h_H)^T (L^H)_{yz} J^h_H.
\]  

(11.29)

The formation of the coarse grid operator in this way saves 31\% and 50\% of the operations for the seven and twenty-seven point grid operators respectively. As an added bonus the coding is much less complex and easier to debug.
CHAPTER 12

3D SMOOTHERS

There are several choices of relaxation methods that can be used for the 3D smoother. We have chosen to look at point, line, and plane Gauss–Seidel relaxation methods using either lexicographic or multi-color ordering.

12.1 Point Gauss–Seidel

The point Gauss–Seidel method in three dimensions is the same as it is in two, but now there are more choices for the sweeping direction. We have chosen to only look at the lexicographic ordering, the red/black ordering for 7-point operators, and an 8-color ordering for 27-point operators. The red/black ordering is given by

\[
\text{Red} : \quad i + j + k \quad \text{even} \\
\text{Black} : \quad i + j + k \quad \text{odd} ,
\]

(12.1)
and the 8–color ordering is given by

\begin{align*}
\text{Black} & : i \text{ odd}, \; j \text{ odd} \; k \text{ odd} \\
\text{Red} & : i \text{ odd}, \; j \text{ odd} \; k \text{ even} \\
\text{orange} & : i \text{ odd}, \; j \text{ even} \; k \text{ odd} \\
\text{yellow} & : i \text{ odd}, \; j \text{ even} \; k \text{ even} \\
\text{Green} & : i \text{ even}, \; j \text{ odd} \; k \text{ odd} \\
\text{Blue} & : i \text{ even}, \; j \text{ odd} \; k \text{ even} \\
\text{violet} & : i \text{ even}, \; j \text{ even} \; k \text{ odd} \\
\text{white} & : i \text{ even}, \; j \text{ even} \; k \text{ even}.
\end{align*}

(12.2)

12.2 Line Gauss–Seidel

We have three choices for the direction of the lines, for line Gauss–Seidel relaxation, either x-lines, y-lines, or z-lines. We can also look at alternating line relaxation, as we did in two dimensions, except that now we have four possibilities; x- and y-lines, y- and z-lines, x- and z-lines, or x-, y-, and z-lines.

As before we can look at different orderings of the the lines. Lexicographic is a common choice, but it can not be parallelized. We can get better convergence and obtain parallelism and vectorization, across lines as in the two dimensional case, by using a zebra (red–black) ordering of the lines.

For standard coarsening, the only choice of smoother, which might be robust, is alternating zebra line Gauss–Seidel relaxation. It will be a good smoother for some of the convection problems, but not others because each coordinate line direction sweep can handle anisotropies and convections with components in its coordinate direction. However, anisotropies in a plane not being swept by the lines will exhibit poor smoothing.
12.3 Plane Gauss–Seidel

In three dimensions we can now perform plane relaxation, which is analogous to line relaxation in two dimensions. Plane relaxation can be performed in several ways; xy-plane, yz-plane, xz-plane, or alternating plane relaxation. These methods can also be done in a lexicographic or red/black ordering of the planes.

We need a robust method for our smoother and these can be found among those that perform plane relaxations [11]. In general, red/black ordering will give better results than lexicographic because it removes the directional dependencies that are associated with a sweeping direction. However, plane relaxation can not reduce the error orthogonal to the plane, and hence we must use alternating plane relaxation to obtain a robust smoother.

Alternating red/black plane Gauss–Seidel relaxation is the most robust because it takes into account the three coordinate directions for anisotropies and convection. One iteration of the method is performed by performing red/black xy-plane Gauss–Seidel relaxation followed by red/black yz-plane Gauss–Seidel relaxation, and finally followed by red/black xz-plane Gauss–Seidel relaxation.

The question now arises as to how to efficiently perform the plane solves needed by the smoother. In 2D we used a cyclic reduction tri-diagonal solver to perform the line solves, but in 3D we are stuck with having to solve a sparse banded system. To perform LU factorization and solve for each plane would be very time consuming. We could save some time by saving the LU decompositions, but at the expense of memory. However, there is a better solution to our problem: we can use a 2D multigrid method to perform the plane solves. We have chosen to use the 2D black box multigrid method for the planes solves because it was designed for just such a mission. By using the 2D multigrid method we still need extra memory, but not as much as the LU method, and
we can also perform multigrid much quicker than LU.

One possible drawback to using the 2D multigrid method is that it is not an exact solver. This should not be much of a problem since the relaxation method gives only an improved approximation to the solution for each iteration. However, we do not want to degrade the convergence of the relaxation by providing poor approximations for the plane solves. We have found that it is usually sufficient to use a single $V(1,1)$-cycle in the 2D black box multigrid method with alternating zebra line Gauss–Seidel relaxation to obtain essentially the same results for the red/black plane Gauss–Seidel relaxation as when LU factorization is used. Depending on the convection characteristics it is sometimes better to use either a $V(2,1)$-cycle, $W(1,1)$-cycle, or several $V(1,1)$-cycles; however, this improvement in the plane solve accuracy is moot since the relaxation method can fail even when exact plane solves are used.
CHAPTER 13

LOCAL MODE ANALYSIS IN THREE DIMENSIONS

Local mode analysis of 3D smoothers is somewhat sparse in the literature and does not have adequate coverage for the range of problems that we wish to solve. In addition, there are only hints in the literature for how to perform local mode analysis for color relaxation in three dimensions, and we are unaware of the appearance elsewhere of the detailed analysis that we have presented in this chapter.

The local (Fourier) mode analysis was described in section 5.3 for two dimensions, and we now extend it to three dimensions.

13.1 Overview of 3D Local Mode Analysis

The continuous problem is discretized into a system of algebraic equations

\[ Lu = f \]  \hspace{1cm} (13.1)

where the grid \( G \) is defined by

\[ G = \left\{ \begin{array}{l} x_i = i \cdot h_x, \quad i = 1, \ldots, n_x \quad h_x = \frac{1}{n_x-1} \\ (x_i, y_j, z_k) \in \mathbb{R} \quad y_j = j \cdot h_y, \quad j = 1, \ldots, n_y \quad h_y = \frac{1}{n_y-1} \\ z_k = k \cdot h_z, \quad k = 1, \ldots, n_z \quad h_z = \frac{1}{n_z-1} \end{array} \right\} \]  \hspace{1cm} (13.2)
The grid operator $L$ can be represented in stencil notation as

$$
\begin{bmatrix}
NW_t & N_t & NE_t \\
W_t & C_t & E_t \\
SW_t & S_t & SE_t
\end{bmatrix}_t 
\quad 
\begin{bmatrix}
NW_p & N_p & NE_p \\
W_p & C_p & E_p \\
SW_p & S_p & SE_p
\end{bmatrix}_p
\quad 
\begin{bmatrix}
NW_b & N_b & NE_b \\
W_b & C_b & E_b \\
SW_b & S_b & SE_b
\end{bmatrix}_b
$$

(13.3)

where the subscripts $b$, $p$, and $t$ stand for the bottom $(k-1)$, plane $(k)$, and top $(k+1)$ levels of the stencil.

If the continuous problem has constant coefficients and periodic boundary conditions, then the stencils of $[L]$, $[M]$, and $[N]$ are independent of the grid points $(i, j, k)$. The eigenfunctions of the smoothing amplification matrix $S$ are

$$
\Phi_{i,j,k}(\theta) = e^{(i\theta_x+j\theta_y+k\theta_z)} 
\quad \theta \in \Theta,
$$

(13.4)

where $\theta = (\theta_x, \theta_y, \theta_z)$, and $\Theta$ is defined as

$$
\Theta = \left\{ (\theta_x, \theta_y, \theta_z) : \theta_i = \frac{2\pi l_i}{n_i}, l_i = \frac{n_i}{2} - 1, \frac{n_i}{2}, \ldots, \frac{n_i}{2}; i = x, y, z \right\}.
$$

(13.5)

If $n_x, n_y$, and $n_z$ are assumed to be even, then the corresponding eigenvalues of $S$ are

$$
\lambda(\theta) = \frac{\sum_{\kappa} N(\kappa) e^{(i\theta_x+l_y\theta_y+l_z\theta_z)}}{\sum_{\kappa} M(\kappa) e^{(i\theta_x+l_y\theta_y+l_z\theta_z)}} 
\quad \theta \in \Theta,
$$

(13.6)

where $\kappa = (l_x, l_y, l_z)$ is a vector.

We now define the sets of rough and smooth frequencies for the grid $G$, when the ratio between the fine and coarse grid spacings is two. The smooth frequencies are defined as

$$
\Theta_s = \Theta \cap \left( -\frac{\pi}{2}, \frac{\pi}{2} \right)^3,
$$

(13.7)
and the rough frequencies as

\[ \Theta_r = \Theta \setminus \Theta_s. \]  

(13.8)

The Fourier smoothing factor is then defined to be

\[ \mu = \max_{\theta \in \Theta_r} \{ |\lambda(\theta)| \}, \]  

(13.9)

just as it was in two dimensions. The smoothing factor can be made grid size independent by changing the definition of \( \Theta \) to be

\[ \Theta = \{(\theta_x, \theta_y, \theta_z) : \theta_i \in [-\pi, \pi], \ i = x, y, z \}. \]  

(13.10)

For the case of multi-color relaxation, the \( \Phi_{i,j,k}(\theta) \) are again not eigenfunctions any more, but certain subspaces spanned by their linear combinations are still invariant. Instead of four invariant subspaces, as in two dimensions, we now have eight invariant subspaces, which are defined as

\[ \theta^1 = \Theta_s \cap [-\frac{\pi}{2}, \frac{\pi}{2}]^3 \]

\[ \theta^2 = (\theta^1_x - \text{sign}(\theta^1_y^1)x, \theta^1_y - \text{sign}(\theta^1_y^1)y, \theta^1_z - \text{sign}(\theta^1_z^1)z) \]

\[ \theta^3 = (\theta^1_x - \text{sign}(\theta^1_y^1)x, \theta^1_y - \text{sign}(\theta^1_y^1)y, 0) \]

\[ \theta^4 = (\theta^1_x - \text{sign}(\theta^1_y^1)x, 0, \theta^1_z - \text{sign}(\theta^1_z^1)z) \]

\[ \theta^5 = (0, \theta^1_y - \text{sign}(\theta^1_y^1)y, \theta^1_z - \text{sign}(\theta^1_z^1)z) \]

\[ \theta^6 = \theta^1 - (0, 0, \theta^1_z - \text{sign}(\theta^1_z^1)z) \]

\[ \theta^7 = \theta^1 - (0, \theta^1_y - \text{sign}(\theta^1_y^1)y, 0) \]

\[ \theta^8 = \theta^1 - (\theta^1_x - \text{sign}(\theta^1_x^1)x, 0, 0), \]

and \( \Phi(\theta) \) is now written as

\[ \Phi(\theta) = (\Phi(\theta^1), \ldots, \Phi(\theta^8)). \]  

(13.12)

The error before smoothing is now

\[ e^0(\theta) = c_0 \Phi(\theta), \]  

(13.13)
and after smoothing it is

$$e^1(\theta) = (S(\theta)c_0)\Phi(\theta),$$

(13.14)

where $S(\theta)$ is the $8 \times 8$ amplification matrix, and $c_0$ is a vector of dimension 8. The amplification matrix is computed in the same way as in the two dimensional case.

For multi-colored relaxations, the definition of the Fourier smoothing factor, $\mu$, has to be modified. The rough Fourier modes are now given by

$$\Phi(\theta^2), \ldots, \Phi(\theta^8)$$

(13.15)

and the smooth Fourier modes are now represented by

$$\Phi(\theta^1), \quad \theta^1_x \neq -\frac{\pi}{2}, \quad \theta^1_y \neq -\frac{\pi}{2}, \quad \text{or} \quad \theta^1_z \neq -\frac{\pi}{2}. \quad (13.16)$$

All of these values must be added to $\Theta_r$. We now define a projection operator, $Q(\theta)$, for $\Phi(\theta)$ onto the Fourier modes, which is represented by the diagonal $8 \times 8$ matrix

$$Q(\theta) = \begin{pmatrix}
\delta(\theta) & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{pmatrix}$$

(13.17)

where

$$\delta(\theta) = \begin{cases}
1 & \theta_i = -\frac{\pi}{2}, \quad \text{for} \quad i = x, y, z \\
0 & \text{otherwise}
\end{cases}.$$  

(13.18)

Define $\Theta^1_i = \theta^1$, and the multi-color definition for the Fourier smoothing factor is given by

$$\mu = \max_{\theta \in \Theta^1} \{ \rho [Q(\theta)S(\theta)] \}.$$  

(13.19)
where \( \rho \) denotes the spectral radius.

The definitions for the smoothing factor can be modified, as in the two dimensional case, to take into account the Dirichlet boundary conditions.

### 13.2 Three Dimensional Model Problems

The domain \( \Omega \) is the unit cube for the three dimensional model problems:

1. \(-\Delta u = f\)
2. \(-\varepsilon_1 u_{xx} - \varepsilon_2 u_{yy} - \varepsilon_3 u_{zz} = f\)
   
   a) \(0 < \varepsilon_1 \approx \varepsilon_2 \approx \varepsilon_3\)
   
   b) \(0 < \varepsilon_1 \ll \varepsilon_2 \approx \varepsilon_3\)
   
   c) \(0 < \varepsilon_1 \approx \varepsilon_2 \ll \varepsilon_3\)
   
   d) \(0 < \varepsilon_1 \ll \varepsilon_2 \ll \varepsilon_3\)

3. \(-\varepsilon \Delta u - u_x = f\)
4. \(-\varepsilon \Delta u + u_x = f\)
5. \(-\varepsilon \Delta u - u_y = f\)
6. \(-\varepsilon \Delta u + u_y = f\)
7. \(-\varepsilon \Delta u - u_z = f\)
8. \(-\varepsilon \Delta u + u_z = f\)
9. \(-\varepsilon \Delta u + u_x + u_y = f\)
10. \(-\varepsilon \Delta u + u_x - u_y = f\)
11. \(-\varepsilon \Delta u - u_x + u_y = f\)
12. \(-\varepsilon \Delta u - u_x - u_y = f\)
\[-\varepsilon \Delta u + u_x + u_z = f\]
\[-\varepsilon \Delta u + u_x - u_z = f\]
\[-\varepsilon \Delta u - u_x + u_z = f\]
\[-\varepsilon \Delta u - u_x - u_z = f\]
\[-\varepsilon \Delta u + u_y + u_z = f\]
\[-\varepsilon \Delta u + u_y - u_z = f\]
\[-\varepsilon \Delta u - u_y + u_z = f\]
\[-\varepsilon \Delta u - u_y - u_z = f\]
\[-\varepsilon \Delta u + u_x + u_y + u_z = f\]
\[-\varepsilon \Delta u + u_x + u_y - u_z = f\]
\[-\varepsilon \Delta u + u_x - u_y + u_z = f\]
\[-\varepsilon \Delta u + u_x - u_y - u_z = f\]
\[-\varepsilon \Delta u - u_x + u_y + u_z = f\]
\[-\varepsilon \Delta u - u_x + u_y - u_z = f\]
\[-\varepsilon \Delta u - u_x - u_y + u_z = f\]
\[-\varepsilon \Delta u - u_x - u_y - u_z = f\]

where \(\Delta u = u_{xx} + u_{yy}\), \(\varepsilon = 10^{-p}\) for \(p = 0, 1, \ldots, 5\), and the \(\pm\) are to be taken in all possible combinations.
13.3 Local Mode Analysis for Point Gauss-Seidel Relaxation

Local mode analysis results are presented for lexicographical and red/black ordering for point Gauss-Seidel relaxations.

Point Gauss-Seidel relaxation with lexicographic ordering gives the splitting

\[
[M] = \begin{bmatrix}
0 & C_b & 0 \\
0 & W_p & C_p & 0 \\
0 & S_p & 0
\end{bmatrix},
\begin{bmatrix}
0 \\
0 & 0 & 0 \\
\end{bmatrix}_b,
\begin{bmatrix}
0 \\
0 & 0 & 0 \\
\end{bmatrix}_p
\]

(13.20)

\[
[N] = \begin{bmatrix}
0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}_b,
\begin{bmatrix}
0 & -N_p \\
0 & 0 & -E_p \\
0 & 0 & -C_t
\end{bmatrix}_p,
\begin{bmatrix}
0 \\
0 & 0
\end{bmatrix}_t
\]

(13.21)

The amplification factor \( \lambda(\theta) \) is given by

\[
\lambda(\theta) = \frac{|E_p e^{i\theta} + N_p e^{i\theta} + C_t e^{i\theta}|}{|C_b e^{-i\theta} + W_p e^{-i\theta} + C_p + S_p e^{-i\theta}|}
\]

(13.22)

Red/black point Gauss-Seidel relaxation has the amplification matrix

\[
S(\theta) = \frac{1}{2} \begin{bmatrix}
a & a & 0 & 0 & 0 & 0 & 0 & 0 \\
b & b & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & c & 0 & 0 & 0 & 0 & c \\
0 & 0 & 0 & e & 0 & 0 & e & 0 \\
0 & 0 & 0 & g & g & 0 & 0 \\
0 & 0 & 0 & h & h & 0 & 0 \\
0 & 0 & 0 & f & 0 & f & 0 \\
0 & 0 & d & 0 & 0 & 0 & 0 & d
\end{bmatrix}
\]

(13.23)
where \( a = 1 + \alpha \), \( b = 1 - \alpha \), \( c = \beta (1 + \beta) \), \( d = 1 - \beta \), \( e = \gamma (1 + \gamma) \), \( f = 1 - \gamma \), \( g = \eta (1 + \eta) \), \( h = 1 - \eta \), and

\[
\begin{align*}
\alpha &= \frac{-C_t e^{-i\theta_t} - S_p e^{-i\theta_y} - W_p e^{-i\theta_x} - E_p e^{i\theta_x} - N_p e^{i\theta_y} - C_t e^{i\theta_t}}{C}, \\
\beta &= \frac{C_t e^{-i\theta_t} + S_p e^{-i\theta_y} - W_p e^{-i\theta_x} - E_p e^{i\theta_x} + N_p e^{i\theta_y} + C_t e^{i\theta_t}}{C}, \\
\gamma &= \frac{C_t e^{-i\theta_t} - S_p e^{-i\theta_y} + W_p e^{-i\theta_x} + E_p e^{i\theta_x} - N_p e^{i\theta_y} + C_t e^{i\theta_t}}{C}, \\
\eta &= \frac{-C_t e^{-i\theta_t} + S_p e^{-i\theta_y} + W_p e^{-i\theta_x} + E_p e^{i\theta_x} + N_p e^{i\theta_y} - C_t e^{i\theta_t}}{C}.
\end{align*}
\]

The eigenvalues of \( Q(\theta)S(\theta) \) are

\[
\begin{align*}
\lambda_1(\theta) &= 0 \\
\lambda_2(\theta) &= 0 \\
\lambda_3(\theta) &= 0 \\
\lambda_4(\theta) &= 0 \\
\lambda_5(\theta) &= \frac{1}{2} (1 + \beta^2) \\
\lambda_6(\theta) &= \frac{1}{2} (1 + \gamma^2) \\
\lambda_7(\theta) &= \frac{1}{2} (1 - \alpha + \delta(\theta)(1 + \alpha)) \\
\lambda_8(\theta) &= \frac{1}{2} (1 + \eta^2).
\end{align*}
\]

The results of local mode analysis for the model problems from section 13.2 are shown in tables 13.1 through 13.4. The smoothing factors were computed numerically with the grid spacing \( h = 1 \) and the angles \( \theta_x \), \( \theta_y \), and \( \theta_z \) were sampled at two degree increments.

Table 13.1 shows the results of the smoothing analysis for pure diffusion type problems. The point Gauss–Seidel relaxation is a reasonable smoother for Poisson’s equation, but not for anisotropic problems.
Table 13.1. Smoothing factor $\mu$ for point Gauss–Seidel relaxation in lexicographical (pGS-lex) and red/black (r/b-pGS) ordering for the indicated anisotropic diffusion problems (see section 13.2) using central finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p_x$ $p_y$ $p_z$</th>
<th>pGS-lex</th>
<th>r/b-pGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0</td>
<td>0.9093</td>
<td>0.9526</td>
</tr>
<tr>
<td></td>
<td>3 0 0</td>
<td>0.9990</td>
<td>0.9993</td>
</tr>
<tr>
<td></td>
<td>5 0 0</td>
<td>0.9999</td>
<td>0.9998</td>
</tr>
<tr>
<td>2b</td>
<td>1 1 0</td>
<td>0.8472</td>
<td>0.9187</td>
</tr>
<tr>
<td></td>
<td>3 3 0</td>
<td>0.9981</td>
<td>0.9988</td>
</tr>
<tr>
<td></td>
<td>5 5 0</td>
<td>0.9999</td>
<td>0.9998</td>
</tr>
<tr>
<td>2c</td>
<td>2 1 0</td>
<td>0.9821</td>
<td>0.9907</td>
</tr>
<tr>
<td></td>
<td>5 3 0</td>
<td>0.9999</td>
<td>0.9998</td>
</tr>
</tbody>
</table>

Table 13.2. Smoothing factor $\mu$ for point Gauss–Seidel relaxation in lexicographical (pGS-lex) and red/black (r/b-pGS) ordering for the indicated convection–diffusion problems (see section 13.2) using central and upstream finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p$</th>
<th>pGS-lex</th>
<th>r/b-pGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0.6459</td>
<td>0.7515</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8742</td>
<td>0.8808</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9983</td>
<td>0.9978</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.5550</td>
<td>0.7515</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5617</td>
<td>0.8808</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5524</td>
<td>0.9978</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.6459</td>
<td>0.7515</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8742</td>
<td>0.8808</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9983</td>
<td>0.9978</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.5550</td>
<td>0.7515</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5617</td>
<td>0.8808</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5524</td>
<td>0.9978</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.6459</td>
<td>0.7515</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8742</td>
<td>0.8808</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9983</td>
<td>0.9978</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.5550</td>
<td>0.7515</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5617</td>
<td>0.8808</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5524</td>
<td>0.9978</td>
</tr>
</tbody>
</table>
Table 13.3. Smoothing factors for point Gauss–Seidel relaxation with lexicographic and red/black ordering for the indicated convection–diffusion problems (see section 13.2); $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>pGS-lex</th>
<th>r/b-pGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>0.5460</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5482</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5059</td>
<td>0.9988</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.6385</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8715</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9988</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.6385</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8715</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9988</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.6999</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9251</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9991</td>
<td>0.9988</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.5460</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5489</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.1312</td>
<td>0.9988</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.6384</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8715</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9988</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.6385</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8715</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9988</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.6998</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9251</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9991</td>
<td>0.9988</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0.5459</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5484</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5423</td>
<td>0.9988</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.6385</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8715</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9988</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0.6385</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8715</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9988</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.6998</td>
<td>0.7779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9251</td>
<td>0.9247</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9991</td>
<td>0.9988</td>
</tr>
</tbody>
</table>
Table 13.4. Smoothing factor $\mu$ for point Gauss–Seidel relaxation in lexicographical (pGS-lex) and red/black (r/b-pGS) ordering for the indicated convection–diffusion problems (see section 13.2) using central and upstream finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>pGS-lex</th>
<th>r/b-pGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0</td>
<td>0.4176</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.1295</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.0017</td>
<td>0.7031</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0.5749</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7970</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9967</td>
<td>0.7031</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0.5749</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7969</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9967</td>
<td>0.7031</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0.6513</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8851</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9984</td>
<td>0.7031</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0.5749</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7969</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9968</td>
<td>0.7031</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>0.6513</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8851</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9985</td>
<td>0.7031</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0.6513</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8851</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9984</td>
<td>0.7031</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>0.7048</td>
<td>0.7161</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9199</td>
<td>0.7067</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9990</td>
<td>0.7031</td>
</tr>
</tbody>
</table>
The tables 13.2 through 13.4 show the results of the smoothing analysis for convection–diffusion problems. Most of the smoothing factors approach one as the convection terms become more dominant, which implies that point Gauss–Seidel is not a good smoother for these types of problems. However, lexicographic point Gauss–Seidel relaxation exhibits good smoothing properties when the convection characteristic coincides with that of the sweeping direction.

13.4 Local Mode Analysis for Line Gauss–Seidel Relaxation

The line Gauss-Seidel relaxation can be implemented in many ways for three dimensional problems. It can be done by lines in any of the three axis directions. The ordering of the lines of unknowns can be done in many ways. Local mode analysis results are presented for lexicographical and zebra (red/black) ordering for x–line Gauss–Seidel relaxations and alternating line Gauss–Seidel relaxation.

X–line Gauss–Seidel relaxation with lexicographic ordering gives the splitting

\[
[M] = \begin{bmatrix}
0 & 0 \\
0 & C_b & 0 \\
0 & 0 & 0 \\
\end{bmatrix}_b, \quad
\begin{bmatrix}
0 & W_p & C_p & E_p \\
\end{bmatrix}_p, \quad
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}_t
\] (13.28)

\[
[N] = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}_b, \quad
\begin{bmatrix}
-N_p & 0 \\
0 & 0 \\
0 & -C_t & 0 \\
\end{bmatrix}_p, \quad
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{bmatrix}_t
\] (13.29)

The amplification factor \(\lambda(\theta)\) is given by

\[
\lambda(\theta) = \frac{|N_p e^{i\theta_y} + C_t e^{i\theta_z}|}{|C_b e^{-i\theta_z} + W_p e^{-i\theta_x} + E_p e^{i\theta_x} + C_p + S_p e^{-i\theta_y}|}. \tag{13.30}
\]
Zebra x-line Gauss-Seidel relaxation has the amplification matrix

\[
S(\theta) = \frac{1}{2} \begin{pmatrix}
a & 0 & -a & 0 & 0 & 0 & 0 \\
0 & c & 0 & 0 & 0 & 0 & -c \\
b & 0 & -b & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & e & -e & 0 & 0 \\
0 & 0 & 0 & f & -f & 0 & 0 \\
0 & 0 & 0 & 0 & g & -g & 0 \\
0 & 0 & 0 & 0 & 0 & h & -h \\
0 & d & 0 & 0 & 0 & 0 & -d \\
\end{pmatrix}
\]

where \(a = \alpha(1+\alpha), b = \alpha(1-\alpha), c = \beta(1+\beta), d = \beta(1-\beta), e = \gamma(1+\gamma), f = \gamma(1-\gamma),\)
\(g = \eta(1+\eta), h = \eta(1-\eta),\) and

\[
\alpha = -\frac{C_b e^{-\theta_s} + S_p e^{-\theta_v} + N_p e^{\theta_v} + C_t e^{\theta_s}}{W_p e^{-\theta_s} + C_p + E_p e^{\theta_s}}, \quad (13.32)
\]

\[
\beta = \frac{C_b e^{-\theta_s} + S_p e^{-\theta_v} + N_p e^{\theta_v} + C_t e^{\theta_s}}{-W_p e^{-\theta_s} + C_p - E_p e^{\theta_s}}, \quad (13.33)
\]

\[
\gamma = \frac{C_b e^{-\theta_s} - S_p e^{-\theta_v} - N_p e^{\theta_v} + C_t e^{\theta_s}}{-W_p e^{-\theta_s} + C_p - E_p e^{\theta_s}}, \quad (13.34)
\]

\[
\eta = \frac{C_b e^{-\theta_s} - S_p e^{-\theta_v} - N_p e^{\theta_v} + C_t e^{\theta_s}}{W_p e^{-\theta_s} + C_p + E_p e^{\theta_s}}. \quad (13.35)
\]

The eigenvalues of \(Q(\theta)S(\theta)\) are

\[
\lambda_1(\theta) = 0
\]

\[
\lambda_2(\theta) = 0
\]

\[
\lambda_3(\theta) = 0
\]

\[
\lambda_4(\theta) = 0
\]

\[
\lambda_5(\theta) = \beta^2
\]

\[
\lambda_6(\theta) = \gamma^2
\]
\begin{align*}
\lambda_7(\theta) &= \eta^2 \\
\lambda_8(\theta) &= \frac{1}{2} \alpha (\alpha - 1 + \delta(\theta)(1 + \eta)).
\end{align*}

The alternating line Gauss–Seidel relaxation with lexicographic ordering amplification factor \( \lambda(\theta) \) is given by

\[ \lambda(\theta) = \lambda_{xlg_s}(\theta) \lambda_{ylgs}(\theta) \lambda_{zlg_s}(\theta) \quad (13.36) \]

where \( \lambda_{xlg_s}(\theta) \), \( \lambda_{ylgs}(\theta) \), and \( \lambda_{zlg_s}(\theta) \) are the x–, y–, and z–line Gauss–Seidel amplification factors respectively, given by

\[ \lambda_{xlg_s}(\theta) = \frac{|N_p e^{i\theta_y} + C_t e^{i\theta_z}|}{|C_k e^{-i\theta_z} + W_p e^{-i\theta_y} + E_p e^{i\theta_x} + C_p + S_p e^{-i\theta_y}|} \quad (13.37) \]

\[ \lambda_{ylgs}(\theta) = \frac{|E_p e^{i\theta_y} + C_t e^{i\theta_z}|}{|C_k e^{-i\theta_z} + W_p e^{-i\theta_y} + C_p + S_p e^{-i\theta_y} + N_p e^{i\theta_y}|} \quad (13.38) \]

\[ \lambda_{zlg_s}(\theta) = \frac{|E_p e^{i\theta_y} + N_p e^{i\theta_y}|}{|C_k e^{-i\theta_z} + W_p e^{-i\theta_y} + C_p + S_p e^{-i\theta_y} + C_t e^{i\theta_z}|}. \quad (13.39) \]

The zebra alternating line Gauss–Seidel relaxation amplification matrix \( S(\theta) \) is given by

\[ S(\theta) = S_{xlg_s}(\theta) S_{ylgs}(\theta) S_{zlg_s}(\theta) \quad (13.40) \]

where \( S_{xlg_s}(\theta) \), \( S_{ylgs}(\theta) \), and \( S_{zlg_s}(\theta) \) are the zebra x–, y–, and z–line Gauss–Seidel amplification matrices respectively. The zebra x–line Gauss–Seidel amplification matrix \( S_{xlg_s}(\theta) \) is given in equation (13.31), and the amplification matrices for \( S_{ylgs}(\theta) \) and
\( S_{\text{z}l_g}(\theta) \) are given by

\[
S_{\text{y}l_g}(\theta) = \frac{1}{2} \begin{pmatrix}
  a & 0 & 0 & -a & 0 & 0 & 0 \\
  0 & c & 0 & 0 & 0 & -c & 0 \\
  0 & 0 & e & 0 & -e & 0 & 0 \\
  b & 0 & 0 & -b & 0 & 0 & 0 \\
  0 & f & 0 & -f & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & g & 0 & -g \\
  0 & d & 0 & 0 & 0 & -d & 0 \\
  0 & 0 & 0 & 0 & 0 & h & -h
\end{pmatrix}
\] (13.41)

where \( a = \alpha(1+\alpha) \), \( b = \alpha(1-\alpha) \), \( c = \beta(1+\beta) \), \( d = \beta(1-\beta) \), \( e = \gamma(1+\gamma) \), \( f = \gamma(1-\gamma) \), \( g = \eta(1+\eta) \), \( h = \eta(1-\eta) \), and

\[
\alpha = \frac{C_l e^{\theta \gamma} + W_p e^{\theta \gamma} + E_p e^{\theta \gamma} + C_t e^{\theta \gamma}}{S_p e^{\theta \gamma} + C_p + N_p e^{\theta \gamma}}, \tag{13.42}
\]

\[
\beta = \frac{C_l e^{\theta \gamma} + W_p e^{\theta \gamma} + E_p e^{\theta \gamma} + C_t e^{\theta \gamma}}{-S_p e^{\theta \gamma} + C_p - N_p e^{\theta \gamma}}, \tag{13.43}
\]

\[
\gamma = \frac{C_l e^{\theta \gamma} - W_p e^{\theta \gamma} - E_p e^{\theta \gamma} + C_t e^{\theta \gamma}}{-S_p e^{\theta \gamma} + C_p - N_p e^{\theta \gamma}}, \tag{13.44}
\]

\[
\eta = \frac{C_l e^{\theta \gamma} - W_p e^{\theta \gamma} - E_p e^{\theta \gamma} + C_t e^{\theta \gamma}}{S_p e^{\theta \gamma} + C_p + N_p e^{\theta \gamma}} \tag{13.45}
\]

and

\[
S(\theta) = \frac{1}{2} \begin{pmatrix}
  a & 0 & 0 & 0 & -a & 0 & 0 & 0 \\
  0 & c & 0 & 0 & 0 & -c & 0 & 0 \\
  0 & 0 & e & 0 & -e & 0 & 0 & 0 \\
  0 & f & 0 & -f & 0 & 0 & 0 & 0 \\
  b & 0 & 0 & -b & 0 & 0 & 0 & 0 \\
  0 & d & 0 & 0 & 0 & -d & 0 & 0 \\
  0 & 0 & 0 & 0 & 0 & g & -g & 0 \\
  0 & 0 & 0 & 0 & 0 & h & -h & 0
\end{pmatrix} \tag{13.46}
\]
Table 13.5. Smoothing factor $\mu$ for x-, y-, and z-line and alternating line Gauss-Seidel relaxation in lexicographic ordering, xlGS, ylGS, zlGS, and alGS respectively, for the indicated anisotropic diffusion problems (see section 13.2) using central finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>xlGS</th>
<th>ylGS</th>
<th>zlGS</th>
<th>alGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.1096</td>
</tr>
<tr>
<td>2b</td>
<td>3.0</td>
<td>0.9999</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.9950</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>2c</td>
<td>1.0</td>
<td>0.8462</td>
<td>0.8461</td>
<td>0.5000</td>
<td>0.3396</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.5000</td>
<td>0.4976</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.5000</td>
<td>0.5000</td>
</tr>
<tr>
<td>2d</td>
<td>2,1</td>
<td>0.9821</td>
<td>0.9804</td>
<td>0.8347</td>
<td>0.8036</td>
</tr>
<tr>
<td></td>
<td>5,3</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9804</td>
<td>0.9804</td>
</tr>
</tbody>
</table>

where $a = \alpha(1+\alpha)$, $b = \alpha(1-\alpha)$, $c = \beta(1+\beta)$, $d = \beta(1-\beta)$, $e = \gamma(1+\gamma)$, $f = \gamma(1-\gamma)$, $g = \eta(1+\eta)$, $h = \eta(1-\eta)$, and

$$
\alpha = - \frac{S_p e^{-\phi_p} + W_p e^{-\phi_p} + E_p e^{\phi_p} + N_p e^{\phi_p}}{C_b e^{-\phi_p} + C_p + C_l e^{\phi_p}}, \quad (13.47)
$$

$$
\beta = \frac{S_p e^{-\phi_p} + W_p e^{-\phi_p} + E_p e^{\phi_p} + N_p e^{\phi_p}}{-C_b e^{-\phi_p} + C_p - C_l e^{\phi_p}}, \quad (13.48)
$$

$$
\gamma = \frac{S_p e^{-\phi_p} - W_p e^{-\phi_p} - E_p e^{\phi_p} + N_p e^{\phi_p}}{-C_b e^{-\phi_p} + C_p - C_l e^{\phi_p}}, \quad (13.49)
$$

$$
\eta = \frac{S_p e^{-\phi_p} - W_p e^{-\phi_p} - E_p e^{\phi_p} + N_p e^{\phi_p}}{C_b e^{-\phi_p} + C_p + C_l e^{\phi_p}}. \quad (13.50)
$$

The zebra alternating line Gauss-Seidel amplification matrix $S(\theta)$ can be computed numerically and then its eigenvalues can be found and evaluated on $\Theta_z$.

The results of local mode analysis for the model problems from section 13.2 are shown in tables 13.5 through 13.8. The smoothing factors were computed numerically with the grid spacing $h = 1$ and the angles $\theta_x$, $\theta_y$, and $\theta_z$ were sampled at 2 degree increments.

Tables 13.6 through 13.8 show the smoothing factors for the convection-diffusion model problems for lexicographic line Gauss-Seidel relaxation. The smoothing
Table 13.6. Smoothing factor $\mu$ for $x$, $y$, and $z$-line and alternating line Gauss–Seidel relaxation in lexicographic ordering, $xlGS$, $ylGS$, $zlGS$, and $alGS$ respectively, for the indicated convection–diffusion problems (see section 13.2) using central and upwind finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>$xlGS$</th>
<th>$ylGS$</th>
<th>$zlGS$</th>
<th>$alGS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0.5047</td>
<td>0.6059</td>
<td>0.6059</td>
<td>0.1626</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5411</td>
<td>0.8751</td>
<td>0.8751</td>
<td>0.3782</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.1250</td>
<td>0.9984</td>
<td>0.9984</td>
<td>0.1241</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.5047</td>
<td>0.5000</td>
<td>0.4999</td>
<td>0.1049</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5411</td>
<td>0.4992</td>
<td>0.4992</td>
<td>0.1089</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.1250</td>
<td>0.1119</td>
<td>0.1119</td>
<td>0.0010</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.6059</td>
<td>0.5047</td>
<td>0.6059</td>
<td>0.1625</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8751</td>
<td>0.5411</td>
<td>0.8751</td>
<td>0.3783</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9984</td>
<td>0.5000</td>
<td>0.9984</td>
<td>0.4976</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.5000</td>
<td>0.5047</td>
<td>0.4999</td>
<td>0.1049</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5000</td>
<td>0.5411</td>
<td>0.4992</td>
<td>0.1089</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.4472</td>
<td>0.1041</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.6059</td>
<td>0.6059</td>
<td>0.5047</td>
<td>0.1625</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8751</td>
<td>0.8751</td>
<td>0.5411</td>
<td>0.3783</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9984</td>
<td>0.9984</td>
<td>0.5000</td>
<td>0.4976</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.5047</td>
<td>0.1049</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.5411</td>
<td>0.1085</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.1041</td>
</tr>
</tbody>
</table>
Table 13.7. Smoothing factor for x-, y-, and z-line and alternating line Gauss-Seidel with lexicographic ordering, xlGS, ylGS, zlGS, and alGS respectively, for convection-diffusion problems (see section 13.2); where \( \varepsilon = 10^{-p} \).

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>xlGS</th>
<th>ylGS</th>
<th>zlGS</th>
<th>alGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>0.4680</td>
<td>0.4681</td>
<td>0.4998</td>
<td>0.1011</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4655</td>
<td>0.4654</td>
<td>0.4995</td>
<td>0.1037</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.4125</td>
<td>0.4125</td>
<td>0.4645</td>
<td>0.0779</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.6089</td>
<td>0.4680</td>
<td>0.5999</td>
<td>0.1551</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8778</td>
<td>0.4655</td>
<td>0.8569</td>
<td>0.3397</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9981</td>
<td>0.4125</td>
<td>0.9977</td>
<td>0.4106</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.4680</td>
<td>0.6089</td>
<td>0.5999</td>
<td>0.1551</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4655</td>
<td>0.8778</td>
<td>0.8569</td>
<td>0.3397</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.4125</td>
<td>0.9984</td>
<td>0.9980</td>
<td>0.4106</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.6089</td>
<td>0.6087</td>
<td>0.6667</td>
<td>0.2190</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8778</td>
<td>0.8778</td>
<td>0.9167</td>
<td>0.6666</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9981</td>
<td>0.9984</td>
<td>0.9990</td>
<td>0.9945</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.4681</td>
<td>0.4999</td>
<td>0.4680</td>
<td>0.1011</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4648</td>
<td>0.4992</td>
<td>0.4645</td>
<td>0.1039</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.0844</td>
<td>0.1114</td>
<td>0.0846</td>
<td>7.9E-4</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.6089</td>
<td>0.5999</td>
<td>0.4680</td>
<td>0.1551</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8778</td>
<td>0.8569</td>
<td>0.4645</td>
<td>0.3393</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9981</td>
<td>0.9977</td>
<td>0.0846</td>
<td>0.0842</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.4681</td>
<td>0.6000</td>
<td>0.6088</td>
<td>0.1551</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4648</td>
<td>0.8571</td>
<td>0.8779</td>
<td>0.3390</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.0844</td>
<td>0.9980</td>
<td>0.9984</td>
<td>0.0841</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.4681</td>
<td>0.6666</td>
<td>0.6088</td>
<td>0.2190</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4648</td>
<td>0.9167</td>
<td>0.8779</td>
<td>0.6666</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.0844</td>
<td>0.9990</td>
<td>0.9984</td>
<td>0.9947</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0.5000</td>
<td>0.4681</td>
<td>0.4681</td>
<td>0.1010</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.5000</td>
<td>0.4654</td>
<td>0.4654</td>
<td>0.1034</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.4472</td>
<td>0.4472</td>
<td>0.1000</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.6000</td>
<td>0.6089</td>
<td>0.4681</td>
<td>0.1551</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8571</td>
<td>0.8778</td>
<td>0.4654</td>
<td>0.3391</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9984</td>
<td>0.4472</td>
<td>0.4454</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0.6000</td>
<td>0.4681</td>
<td>0.6088</td>
<td>0.1551</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8571</td>
<td>0.4654</td>
<td>0.8779</td>
<td>0.3391</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.4472</td>
<td>0.9984</td>
<td>0.4454</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.6667</td>
<td>0.6089</td>
<td>0.6088</td>
<td>0.2190</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9167</td>
<td>0.8778</td>
<td>0.8779</td>
<td>0.6665</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9990</td>
<td>0.9984</td>
<td>0.9984</td>
<td>0.9950</td>
</tr>
</tbody>
</table>
Table 13.8. Smoothing factor $\mu$ for $x$-, $y$-, and $z$-line and alternating line Gauss–Seidel relaxation in lexicographic ordering, xIGS, yIGS, zIGS, and aIGS respectively, for the indicated convection–diffusion problems (see section 13.2) using central and upwind finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>xIGS</th>
<th>yIGS</th>
<th>zIGS</th>
<th>aIGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0</td>
<td>0.3956</td>
<td>0.3956</td>
<td>0.3956</td>
<td>0.0395</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.1302</td>
<td>0.1302</td>
<td>0.1302</td>
<td>9.1E-4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.0017</td>
<td>0.0017</td>
<td>0.0017</td>
<td>1.8E-9</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0.5699</td>
<td>0.5699</td>
<td>0.3956</td>
<td>0.1195</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8488</td>
<td>0.8488</td>
<td>0.1302</td>
<td>0.0852</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9977</td>
<td>0.9977</td>
<td>0.0017</td>
<td>0.0015</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0.5699</td>
<td>0.3956</td>
<td>0.5699</td>
<td>0.1195</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8489</td>
<td>0.1302</td>
<td>0.8488</td>
<td>0.0853</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9977</td>
<td>0.0017</td>
<td>0.9977</td>
<td>0.0015</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0.6686</td>
<td>0.5699</td>
<td>0.5699</td>
<td>0.1662</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9175</td>
<td>0.8488</td>
<td>0.8488</td>
<td>0.2927</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9989</td>
<td>0.9977</td>
<td>0.9977</td>
<td>0.3422</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0.3956</td>
<td>0.5699</td>
<td>0.5699</td>
<td>0.1194</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.1302</td>
<td>0.8488</td>
<td>0.8488</td>
<td>0.0853</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.0017</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.0015</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>0.5699</td>
<td>0.6686</td>
<td>0.5699</td>
<td>0.1662</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8488</td>
<td>0.9175</td>
<td>0.8488</td>
<td>0.2928</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9977</td>
<td>0.9990</td>
<td>0.9980</td>
<td>0.3424</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0.5699</td>
<td>0.5699</td>
<td>0.6686</td>
<td>0.1663</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8489</td>
<td>0.8489</td>
<td>0.9174</td>
<td>0.2928</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9977</td>
<td>0.9980</td>
<td>0.9990</td>
<td>0.3424</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>0.6686</td>
<td>0.6686</td>
<td>0.6686</td>
<td>0.2122</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9175</td>
<td>0.9175</td>
<td>0.9174</td>
<td>0.3612</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9989</td>
<td>0.9990</td>
<td>0.9990</td>
<td>0.3949</td>
</tr>
</tbody>
</table>
Table 13.9. Smoothing factor $\mu$ for zebra $x$, $y$, and $z$-line and alternating line Gauss–Seidel relaxation, ZxlGS, ZylGS, ZzlGS, and ZalGS respectively, for the indicated anisotropic diffusion problems (see section 13.2) using central finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ZxlGS</th>
<th>ZylGS</th>
<th>ZzlGS</th>
<th>ZalGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,0</td>
<td>0.9070</td>
<td>0.8264</td>
<td>0.8264</td>
<td>0.6195</td>
</tr>
<tr>
<td></td>
<td>3,0</td>
<td>0.9990</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.9950</td>
</tr>
<tr>
<td></td>
<td>5,0</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>2b</td>
<td>1,0</td>
<td>0.8403</td>
<td>0.8403</td>
<td>0.2500</td>
<td>0.1736</td>
</tr>
<tr>
<td></td>
<td>3,0</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.2500</td>
<td>0.2490</td>
</tr>
<tr>
<td></td>
<td>5,0</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.2500</td>
<td>0.2500</td>
</tr>
<tr>
<td>2c</td>
<td>1,0</td>
<td>0.9821</td>
<td>0.9803</td>
<td>0.8264</td>
<td>0.7956</td>
</tr>
<tr>
<td></td>
<td>3,0</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.9803</td>
<td>0.9803</td>
</tr>
</tbody>
</table>

Factors for line relaxation are good when the convection term characteristics are in the same direction as the lines. The smoothing factor becomes better (smaller) the more the convection terms dominate if the characteristics are in the direction of the lines. If the characteristics are not in the direction of the lines, then the smoothing factor degenerates, quickly approaching one the more the convection terms dominate.

13.5 Local Mode Analysis for Plane Gauss–Seidel Relaxation

We analyze plane Gauss–Seidel relaxation for $xy$–plane and alternating plane using lexicographic and zebra ordering.

$xy$–plane Gauss–Seidel relaxation with lexicographic ordering gives the splitting

$$[M] = \begin{bmatrix} 0 & C_b & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} N_p \\ W_p & C_p & E_p \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 & 0 & 0 \\ S_p & 0 \end{bmatrix}_p (13.51)$$
Table 13.10. Smoothing factor $\mu$ for zebra x-, y-, and z-line and alternating line Gauss-Seidel relaxation, ZxIGS, ZyIGS, ZzIGS, and ZaIGS respectively, for the indicated convection-diffusion problems (see section 13.2) using central and upwind finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ZxIGS</th>
<th>ZyIGS</th>
<th>ZzIGS</th>
<th>ZaIGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0.3200</td>
<td>0.5102</td>
<td>0.5102</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2500</td>
<td>0.7656</td>
<td>0.7656</td>
<td>0.1406</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.2480</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.3200</td>
<td>0.5102</td>
<td>0.5102</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2500</td>
<td>0.7656</td>
<td>0.7656</td>
<td>0.1406</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.2480</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.5102</td>
<td>0.3200</td>
<td>0.5102</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7656</td>
<td>0.2500</td>
<td>0.7656</td>
<td>0.1406</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.2480</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.5102</td>
<td>0.3200</td>
<td>0.5102</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7656</td>
<td>0.2500</td>
<td>0.7656</td>
<td>0.1406</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.2480</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.5102</td>
<td>0.5102</td>
<td>0.3200</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7656</td>
<td>0.7656</td>
<td>0.2500</td>
<td>0.1406</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.2480</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.5102</td>
<td>0.5102</td>
<td>0.3200</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7656</td>
<td>0.7656</td>
<td>0.2500</td>
<td>0.1406</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.2480</td>
</tr>
</tbody>
</table>
Table 13.11. Smoothing factor for zebra x-, y-, and z-line and alternating line Gauss–Seidel relaxation, ZxIGS, ZyIGS, ZzIGS, and ZalIGS respectively, for convection–diffusion problems (see section 13.2); where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ZxIGS</th>
<th>ZyIGS</th>
<th>ZzIGS</th>
<th>ZalIGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9901</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9901</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9901</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9901</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.0749</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.3846</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.9127</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.5625</td>
<td>0.3846</td>
<td>0.3846</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.8521</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.4599</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9980</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9901</td>
</tr>
</tbody>
</table>
Table 13.12. Smoothing factor $\mu$ for zebra x-, y-, and z-line and alternating line Gauss-Seidel relaxation, ZxlGS, ZylGS, ZzlGS, and ZalGS respectively, for the indicated convection-diffusion problems (see section 13.2) using central and upwind finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ZxIGS</th>
<th>ZyIGS</th>
<th>ZzIGS</th>
<th>ZalGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6900</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.4390</td>
<td>0.0342</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0968</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1553</td>
</tr>
</tbody>
</table>
The amplification factor $\lambda(\theta)$ is given by

$$\lambda(\theta) = \frac{|C_t e^{i\theta_z}|}{|C_k e^{-i\theta_z} + S_p e^{-i\theta_y} + W_p e^{-i\theta_y} + C_p + E_p e^{i\theta_y} + N_p e^{i\theta_y}|}. \quad (13.53)$$

Zebra xy–plane Gauss–Seidel relaxation has the amplification matrix

$$S(\theta) = \frac{1}{2} \begin{pmatrix}
a & 0 & 0 & 0 & -a & 0 & 0 \\
0 & c & 0 & 0 & -c & 0 & 0 \\
0 & 0 & e & 0 & 0 & -e & 0 \\
0 & 0 & 0 & g & 0 & 0 & -g \\
0 & d & 0 & 0 & -d & 0 & 0 \\
b & 0 & 0 & 0 & -b & 0 & 0 \\
0 & f & 0 & 0 & 0 & -f & 0 \\
0 & 0 & 0 & h & 0 & 0 & -h
\end{pmatrix} \quad (13.54)$$

where $a = \alpha(1 + \alpha)$, $b = \alpha(1 - \alpha)$, $c = \beta(1 + \beta)$, $d = \beta(1 - \beta)$, $e = \gamma(1 + \gamma)$, $f = \gamma(1 - \gamma)$, $g = \eta(1 + \eta)$, $h = \eta(1 - \eta)$, and

$$\alpha = \frac{-C_k e^{-i\theta_z} - C_t e^{i\theta_z}}{S_p e^{-i\theta_y} + W_p e^{-i\theta_y} + C_p + E_p e^{i\theta_y} + N_p e^{i\theta_y}}. \quad (13.55)$$

$$\beta = \frac{-C_k e^{-i\theta_z} - C_t e^{i\theta_z}}{S_p e^{-i\theta_y} + W_p e^{-i\theta_y} - C_p + E_p e^{i\theta_y} + N_p e^{i\theta_y}}. \quad (13.56)$$

$$\gamma = \frac{-C_k e^{-i\theta_z} - C_t e^{i\theta_z}}{S_p e^{-i\theta_y} - W_p e^{-i\theta_y} + C_p - E_p e^{i\theta_y} + N_p e^{i\theta_y}}. \quad (13.57)$$

$$\eta = \frac{C_k e^{-i\theta_z} + C_t e^{i\theta_z}}{S_p e^{-i\theta_y} - W_p e^{-i\theta_y} + C_p - E_p e^{i\theta_y} + N_p e^{i\theta_y}}. \quad (13.58)$$

The eigenvalues of $Q(\theta)S(\theta)$ are

$$\lambda_1(\theta) = 0$$
The alternating plane Gauss-Seidel relaxation with lexicographic ordering has the amplification factor $\lambda(\theta)$ given by

$$\lambda(\theta) = \lambda_{xy^{pgs}}(\theta) \lambda_{yz^{pgs}}(\theta) \lambda_{xz^{pgs}}(\theta)$$

(13.59)

where $\lambda_{xy^{pgs}}(\theta), \lambda_{yz^{pgs}}(\theta),$ and $\lambda_{xz^{pgs}}(\theta)$ are the xy-, yz-, and xz-plane Gauss-Seidel amplification factors respectively. The xy-plane Gauss-Seidel amplification factor is given by equation (13.53), and the others by

$$\lambda_{xy^{pgs}}(\theta) = \frac{|E_p e^{i \theta_p}|}{|C_b e^{-i \theta_p} + S_p e^{-i \theta_y} + W_p e^{-i \theta_x} + C_p + N_p e^{i \theta_y} + C_t e^{i \theta_x}|}$$

(13.60)

$$\lambda_{xz^{pgs}}(\theta) = \frac{|N_p e^{i \theta_y}|}{|C_b e^{i \theta_p} + S_p e^{i \theta_y} + W_p e^{i \theta_x} + C_p + E_p e^{i \theta_x} + C_t e^{i \theta_p}|}$$

(13.61)

The zebra alternating plane Gauss-Seidel relaxation amplification matrix $S(\theta)$ is given by

$$S(\theta) = S_{xy^{pgs}}(\theta) S_{yz^{pgs}}(\theta) S_{xz^{pgs}}(\theta)$$

(13.62)

where $S_{xy^{pgs}}(\theta), S_{yz^{pgs}}(\theta),$ and $S_{xz^{pgs}}(\theta)$ are the zebra xy-, yz-, and xz-plane Gauss-Seidel amplification matrices respectively. The zebra xy-plane Gauss-Seidel amplification matrix $S_{xy^{pgs}}(\theta)$ was given in equation (13.54), and the amplification matrices for
\[ S_{y\z\theta_s}(\theta) \text{ and } S_{x\z\theta_s}(\theta) \text{ are given by} \]

\[
S_{y\z\theta_s}(\theta) = \frac{1}{2} \begin{pmatrix}
  a & 0 & 0 & 0 & 0 & 0 & -a \\
  0 & c & -c & 0 & 0 & 0 & 0 \\
  0 & d & -d & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & e & 0 & -e & 0 \\
  0 & 0 & 0 & g & 0 & -g & 0 \\
  0 & 0 & 0 & f & 0 & -f & 0 \\
  b & 0 & 0 & 0 & 0 & 0 & -b 
\end{pmatrix}
\]

where \( a = \alpha(1+\alpha), b = \alpha(1-\alpha), c = \beta(1+\beta), d = \beta(1-\beta), e = \gamma(1+\gamma), f = \gamma(1-\gamma), g = \eta(1+\eta), h = \eta(1-\eta), \) and

\[
\alpha = \frac{|-W_p e^{-i\theta_s} - E_p e^{i\theta_s}|}{|C_\ell e^{i\theta_s} + S_p e^{-i\theta_s} + C_p + N_p e^{i\theta_s} + C_\ell e^{i\theta_s}|},
\]

\[
\beta = \frac{|-W_p e^{-i\theta_s} - E_p e^{i\theta_s}|}{|C_\ell e^{-i\theta_s} + S_p e^{i\theta_s} - C_p + N_p e^{i\theta_s} + C_\ell e^{-i\theta_s}|},
\]

\[
\gamma = \frac{|-W_p e^{-i\theta_s} - E_p e^{i\theta_s}|}{|C_\ell e^{-i\theta_s} - S_p e^{i\theta_s} - C_p - N_p e^{i\theta_s} + C_\ell e^{-i\theta_s}|},
\]

\[
\eta = \frac{|W_p e^{-i\theta_s} + E_p e^{i\theta_s}|}{|C_\ell e^{i\theta_s} - S_p e^{-i\theta_s} + C_p - N_p e^{-i\theta_s} + C_\ell e^{i\theta_s}|},
\]

and

\[
S_{x\z\theta_s}(\theta) = \frac{1}{2} \begin{pmatrix}
  a & 0 & 0 & 0 & 0 & -a & 0 \\
  0 & c & 0 & -c & 0 & 0 & 0 \\
  0 & 0 & e & 0 & 0 & -e & 0 \\
  0 & d & 0 & -d & 0 & 0 & 0 \\
  0 & 0 & 0 & g & 0 & 0 & -g \\
  0 & 0 & f & 0 & 0 & -f & 0 \\
  b & 0 & 0 & 0 & 0 & -b & 0 \\
  0 & 0 & 0 & h & 0 & 0 & -h 
\end{pmatrix}
\]
Table 13.13. Smoothing factor $\mu$ for $xy$-, $xz$-, $yz$-, and alternating plane Gauss–Seidel relaxation in lexicographic ordering, $xyplGS$, $xzplGS$, $yzplGS$, and $AplGS$ respectively, for the indicated anisotropic diffusion problems (see section 13.2) using central finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>$xyplGS$</th>
<th>$xzplGS$</th>
<th>$yzplGS$</th>
<th>$AplGS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.4472</td>
<td>0.4472</td>
<td>0.4472</td>
<td>0.0497</td>
</tr>
<tr>
<td>2b</td>
<td>3.0</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.4472</td>
<td>0.4454</td>
</tr>
<tr>
<td>2c</td>
<td>5.0</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.4472</td>
<td>0.4472</td>
</tr>
<tr>
<td>2d</td>
<td>2,1</td>
<td>0.9980</td>
<td>0.8333</td>
<td>0.4472</td>
<td>0.3654</td>
</tr>
<tr>
<td></td>
<td>5,3</td>
<td>0.9999</td>
<td>0.9804</td>
<td>0.4472</td>
<td>0.4384</td>
</tr>
</tbody>
</table>

where $a = \alpha(1+\alpha)$, $b = \alpha(1-\alpha)$, $c = \beta(1+\beta)$, $d = \beta(1-\beta)$, $e = \gamma(1+\gamma)$, $f = \gamma(1-\gamma)$, $g = \eta(1+\eta)$, $h = \eta(1-\eta)$, and

\[
\alpha = \frac{\left| -S_p e^{-i\theta_y} - N_p e^{i\theta_y} \right|}{\left| C_p e^{-i\theta_x} + W_p e^{-i\theta_x} + C_p + E_p e^{i\theta_x} + C_t e^{i\theta_x} \right|}. \quad (13.69)
\]

\[
\beta = \frac{\left| -S_p e^{-i\theta_y} - N_p e^{i\theta_y} \right|}{\left| C_p e^{-i\theta_x} + W_p e^{-i\theta_x} - C_p + E_p e^{i\theta_x} + C_t e^{i\theta_x} \right|}. \quad (13.70)
\]

\[
\gamma = \frac{\left| -S_p e^{-i\theta_y} - N_p e^{i\theta_y} \right|}{\left| C_p e^{-i\theta_x} - W_p e^{-i\theta_x} - C_p - E_p e^{i\theta_x} + C_t e^{i\theta_x} \right|}. \quad (13.71)
\]

\[
\eta = \frac{\left| S_p e^{-i\theta_y} + N_p e^{i\theta_y} \right|}{\left| C_p e^{-i\theta_x} - W_p e^{-i\theta_x} + C_p - E_p e^{i\theta_x} + C_t e^{i\theta_x} \right|}. \quad (13.72)
\]

The zebra alternating plane Gauss–Seidel amplification matrix $S(\theta)$ can be computed and the eigenvalues can then be found and evaluated on $\Theta_s$.

The results of local mode analysis for the model problems from section 13.2 are shown in tables 13.13 through 13.20. The smoothing factors were computed numerically with the grid spacing $h = 1$ and the angles $\theta_x$, $\theta_y$, and $\theta_z$ were sampled at 2 degree increments.

Table 13.13 shows the results of the local mode analysis for diffusion problems from section 13.2. We can see that single plane relaxation does not yield good
Table 13.14. Smoothing factor $\mu$ for $xy$-, $xz$-, $yz$-, and alternating plane Gauss–Seidel relaxation in lexicographic ordering, $xyplGS$, $xzplGS$, $yzplGS$, and $ApI/GS$ respectively, for the indicated convection–diffusion problems (see section 13.2) using central and upstream finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$p$</th>
<th>$xyplGS$</th>
<th>$xzplGS$</th>
<th>$yzplGS$</th>
<th>$ApI/GS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0.4535</td>
<td>0.4535</td>
<td>0.6323</td>
<td>0.0750</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4878</td>
<td>0.4878</td>
<td>0.9135</td>
<td>0.1335</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.9990</td>
<td>0.0035</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.4535</td>
<td>0.4535</td>
<td>0.3333</td>
<td>0.0500</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4878</td>
<td>0.4878</td>
<td>0.3333</td>
<td>0.0514</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.3333</td>
<td>2.0E-4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.4535</td>
<td>0.6323</td>
<td>0.4535</td>
<td>0.0751</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4878</td>
<td>0.9135</td>
<td>0.4878</td>
<td>0.1335</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.9990</td>
<td>0.5000</td>
<td>0.1488</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.4535</td>
<td>0.3333</td>
<td>0.4535</td>
<td>0.0500</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4878</td>
<td>0.3333</td>
<td>0.4878</td>
<td>0.0516</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.3333</td>
<td>0.5000</td>
<td>0.0497</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.6323</td>
<td>0.4535</td>
<td>0.4535</td>
<td>0.0751</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9135</td>
<td>0.4878</td>
<td>0.4878</td>
<td>0.1329</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9990</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.1488</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.3333</td>
<td>0.4535</td>
<td>0.4535</td>
<td>0.0500</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.3333</td>
<td>0.4878</td>
<td>0.4878</td>
<td>0.0517</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.3333</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.0497</td>
</tr>
</tbody>
</table>
Table 13.15. Smoothing factor for xy-, xz-, yz-, and alternating plane Gauss–Seidel relaxation with lexicographic ordering, \( \text{xyplGS}, \text{xzplGS}, \text{yzplGS}, \) and \( \text{AplGS} \) respectively, for convection–diffusion problems (see section 13.2); where \( \varepsilon = 10^{-p} \).

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>xyplGS</th>
<th>xzplGS</th>
<th>yzplGS</th>
<th>AplGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>0.4587</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.0502</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4918</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.0518</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.0364</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.4587</td>
<td>0.6355</td>
<td>0.3333</td>
<td>0.0754</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4918</td>
<td>0.9147</td>
<td>0.3333</td>
<td>0.1336</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.9987</td>
<td>0.3333</td>
<td>0.1201</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.4587</td>
<td>0.3333</td>
<td>0.6355</td>
<td>0.0754</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4918</td>
<td>0.3333</td>
<td>0.9147</td>
<td>0.1336</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.3333</td>
<td>0.9987</td>
<td>0.1201</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.4587</td>
<td>0.6355</td>
<td>0.6355</td>
<td>0.1136</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.4918</td>
<td>0.9147</td>
<td>0.9147</td>
<td>0.3507</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.5000</td>
<td>0.9987</td>
<td>0.9987</td>
<td>0.3961</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.3333</td>
<td>0.4587</td>
<td>0.3333</td>
<td>0.0502</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.3333</td>
<td>0.4918</td>
<td>0.3333</td>
<td>0.0518</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.3333</td>
<td>0.5000</td>
<td>0.3333</td>
<td>2.0E-4</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.6355</td>
<td>0.4587</td>
<td>0.3333</td>
<td>0.0754</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9147</td>
<td>0.4918</td>
<td>0.3333</td>
<td>0.1340</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>0.5000</td>
<td>0.3333</td>
<td>0.0034</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.3333</td>
<td>0.4587</td>
<td>0.6355</td>
<td>0.0754</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.3333</td>
<td>0.4918</td>
<td>0.9147</td>
<td>0.1338</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.3333</td>
<td>0.5000</td>
<td>0.9987</td>
<td>0.0034</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.6355</td>
<td>0.4587</td>
<td>0.6355</td>
<td>0.1136</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9147</td>
<td>0.4918</td>
<td>0.9147</td>
<td>0.3520</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>0.5000</td>
<td>0.9987</td>
<td>0.0060</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.4587</td>
<td>0.0501</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.4918</td>
<td>0.0516</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.5000</td>
<td>0.0497</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.6355</td>
<td>0.3333</td>
<td>0.4587</td>
<td>0.0754</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9147</td>
<td>0.3333</td>
<td>0.4918</td>
<td>0.1338</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>0.3333</td>
<td>0.5000</td>
<td>0.1488</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0.3333</td>
<td>0.6355</td>
<td>0.4587</td>
<td>0.0754</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.3333</td>
<td>0.9147</td>
<td>0.4918</td>
<td>0.1338</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.3333</td>
<td>0.9987</td>
<td>0.5000</td>
<td>0.1488</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.6355</td>
<td>0.6355</td>
<td>0.4587</td>
<td>0.1136</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9147</td>
<td>0.9147</td>
<td>0.4918</td>
<td>0.3508</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>0.9987</td>
<td>0.5000</td>
<td>0.4454</td>
</tr>
</tbody>
</table>
Table 13.16. Smoothing factor μ for xy-, xz-, yz-, alternating plane Gauss–Seidel relaxation in lexicographic ordering, xypIGS, xzpIGS, yzpIGS, and AplIGS respectively, for the indicated convection–diffusion problems (see section 13.2) using central and upstream finite differences; where ε = 10−p.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>xypIGS</th>
<th>xzpIGS</th>
<th>yzpIGS</th>
<th>AplIGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0</td>
<td>0.2852</td>
<td>0.2852</td>
<td>0.2852</td>
<td>0.0170</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.0713</td>
<td>0.0713</td>
<td>0.0713</td>
<td>3.4E-4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.6E-4</td>
<td>8.6E-4</td>
<td>8.6E-4</td>
<td>6.3E-10</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0.6382</td>
<td>0.2852</td>
<td>0.2852</td>
<td>0.0373</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9150</td>
<td>0.0713</td>
<td>0.0713</td>
<td>0.0038</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>8.6E-4</td>
<td>8.6E-4</td>
<td>6.2E-7</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0.2852</td>
<td>0.6382</td>
<td>0.2852</td>
<td>0.0373</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.0713</td>
<td>0.9150</td>
<td>0.0713</td>
<td>0.0038</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.6E-4</td>
<td>0.9987</td>
<td>8.6E-4</td>
<td>6.2E-7</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0.6382</td>
<td>0.6382</td>
<td>0.2852</td>
<td>0.0603</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9150</td>
<td>0.9150</td>
<td>0.0713</td>
<td>0.0256</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>0.9987</td>
<td>8.6E-4</td>
<td>3.5E-4</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0.2852</td>
<td>0.2852</td>
<td>0.6382</td>
<td>0.0373</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.0713</td>
<td>0.0713</td>
<td>0.9150</td>
<td>0.0038</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.6E-4</td>
<td>8.6E-4</td>
<td>0.9987</td>
<td>6.2E-7</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>0.6382</td>
<td>0.2852</td>
<td>0.6382</td>
<td>0.0603</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9150</td>
<td>0.0713</td>
<td>0.9150</td>
<td>0.0256</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>8.6E-4</td>
<td>0.9987</td>
<td>3.5E-4</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0.2852</td>
<td>0.6382</td>
<td>0.6382</td>
<td>0.0603</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.0713</td>
<td>0.9150</td>
<td>0.9150</td>
<td>0.0256</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.6E-4</td>
<td>0.9987</td>
<td>0.9987</td>
<td>3.5E-4</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>0.6382</td>
<td>0.6382</td>
<td>0.6382</td>
<td>0.0978</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.9150</td>
<td>0.9150</td>
<td>0.9150</td>
<td>0.1760</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9987</td>
<td>0.9987</td>
<td>0.9987</td>
<td>0.2020</td>
</tr>
</tbody>
</table>
smoothing factors when the dominant plane is not the plane of the relaxation method.

Tables 13.14 through 13.16 show local mode smoothing factors for convection–diffusion problems from section 13.2. Good smoothing factors are obtained in most cases except when the sweeping direction is against the convection flow.

Table 13.17 shows the results of the local mode analysis for diffusion problems from section 13.2. Relaxation in a single plane by zebra ordering does not give a good smoothing factor unless the problem is either isotropic or the anisotropies lie in the plane of relaxation.

Tables 13.18 through 13.20 show local mode smoothing factors for convection–diffusion problems from section 13.2. Good smoothing factors are obtained if the convection lies in the plane of relaxation, otherwise the smoothing factor degrades quickly with the increased dominance of the convection terms over the diffusion term.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ZxyplGS</th>
<th>ZxzplGS</th>
<th>ZyzplGS</th>
<th>AZplGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0037</td>
</tr>
<tr>
<td>2b</td>
<td>3</td>
<td>0.9980</td>
<td>0.9980</td>
<td>0.1249</td>
<td>0.1240</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.9999</td>
<td>0.9999</td>
<td>0.0154</td>
<td>0.0154</td>
</tr>
<tr>
<td>2c</td>
<td>3</td>
<td>0.9980</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0160</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.9999</td>
<td>0.2500</td>
<td>0.2500</td>
<td>8.4E-4</td>
</tr>
<tr>
<td>2d</td>
<td>5,3,0</td>
<td>0.9803</td>
<td>0.8264</td>
<td>0.1250</td>
<td>0.0915</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.9999</td>
<td>0.9803</td>
<td>0.0289</td>
<td>0.0210</td>
</tr>
</tbody>
</table>
Table 13.18. Smoothing factor $\mu$ for Zebra $xy$-, $xz$-, $yz$-, and alternating plane Gauss-Seidel relaxation, $ZxyplGS$, $ZxzplGS$, $ZyzplGS$, $AZplGS$ respectively, for the indicated convection-diffusion problems (see section 13.2) using central and upstream finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p$</th>
<th>$ZxyplGS$</th>
<th>$ZxzplGS$</th>
<th>$ZyzplGS$</th>
<th>$AZplGS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.3600</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.7347</td>
<td>0.0279</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.0032</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.3600</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.7347</td>
<td>0.0279</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.0032</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.2500</td>
<td>0.3600</td>
<td>0.2500</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2500</td>
<td>0.7347</td>
<td>0.2500</td>
<td>0.0279</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.0032</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.2500</td>
<td>0.3600</td>
<td>0.2500</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2500</td>
<td>0.7347</td>
<td>0.2500</td>
<td>0.0279</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.2500</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.0032</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.3600</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0279</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0032</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.3600</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0279</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0032</td>
</tr>
</tbody>
</table>

305
Table 13.19. Smoothing factor for zebra $xy$, $xz$, $yz$, and alternating plane Gauss–Seidel relaxation, $ZxyplGS$, $ZxzplGS$, $ZyzplGS$, and $AZplGS$ respectively, for convection–diffusion problems (see section 13.2); where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>$p$</th>
<th>$ZxyplGS$</th>
<th>$ZxzplGS$</th>
<th>$ZyzplGS$</th>
<th>$AZplGS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.3600</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.7347</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.9960</td>
<td>0.1233</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.2358</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.1233</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.1233</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.1233</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.3600</td>
<td>0.3600</td>
<td>0.1542</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.7347</td>
<td>0.7347</td>
<td>0.2355</td>
<td>0.0729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.1250</td>
<td>0.1233</td>
</tr>
</tbody>
</table>
Table 13.20. Smoothing factor $\mu$ for zebra xy-, xz-, yz-, and alternating plane Gauss-Seidel relaxation, ZxypIGS, ZxzpIGS, ZyzpIGS, and AZpIGS respectively, for the indicated convection-diffusion problems (see section 13.2) using central and upstream finite differences; where $\varepsilon = 10^{-p}$.

<table>
<thead>
<tr>
<th>problem</th>
<th>p</th>
<th>ZxypIGS</th>
<th>ZxzpIGS</th>
<th>ZyzpIGS</th>
<th>AZpIGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.2436</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.6944</td>
<td>0.0323</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.9960</td>
<td>0.0420</td>
</tr>
</tbody>
</table>
CHAPTER 14

3D VECTOR ALGORITHM CONSIDERATIONS

Most of the vectorization issues for the black box multigrid algorithms were already covered earlier in chapter 6. We mention here only those issues that are either new or different from those for the 2D algorithms.

14.1 3D Smoother

There are several choices for the smoother, but only alternating plane relaxation gives a robust smoother. Point relaxation is possible, but it is only useful for isotropic diffusion equations. Line relaxation is also possible, but only alternating line relaxation is useful; however, it can perform poorly if the convection characteristics do not align with the grid by either being skewed or varying.

The robust 3D smoother of choice is alternating red/black plane Gauss-Seidel relaxation where the plane solves are computed using the 2D black box multigrid method. On the vector (and sequential) computers a loop was set up to cycle through all of the red planes and then another loop for all of the black planes. The 2D solver was called each time a plane needed to be solved. We had a choice of either writing three 2D black box multigrid solvers; one for each of the three plane data structures, xy-, yz-, and xz-planes respectively, or to use only one 2D black box multigrid solver and transfer the data back and forth from the plane data structures to that of the 2D solver. In our case, we chose to use the later approach, using the 2D vector codes that were developed earlier in this thesis for xy-plane oriented data.
14.2 Data Structures and Memory

For the 3D grid levels we need storage for the grid equations (unknowns, coefficients, and right hand side), the grid transfer operators, and temporary work space. Let \( N_x, N_y, \) and \( N_z \) be the number of grid points in the x-, y-, and z-directions respectively. We can compute how much storage will be needed by adding up the amount used for each grid point. We need 27 locations for the coefficient matrix and 1 each for the unknown and right hand side. For the standard coarsening method we need 52 locations for both the grid transfer operator coefficients, and another 5 for temporary work. For the semi-coarsening method we need 4 locations for both the grid transfer operator coefficients, and another 2 for temporary work. We will ignore the amount of storage for the coarsest grid direct solver because it will remain constant and small when compared to the rest. This means that we need 86 locations for the standard coarsening and 35 for the semi-coarsening. However, we do not have grid transfer coefficients stored on the finest grid so we can subtract 52 and 4 locations from the total for the standard and semi-coarsening methods respectively. The amount of storage required for the 3D data structures, excluding storage for the planes solves, is

\[
\left( 86 \left( 1 + \frac{1}{8} + \cdots \right) - 52 \right) N_x N_y N_z = \frac{324}{7} N_x N_y N_z, \tag{14.1}
\]

\[
\left( 35 \left( 1 + \frac{1}{2} + \cdots \right) - 4 \right) N_x N_y N_z = 66 N_x N_y N_z \tag{14.2}
\]

for the standard and semi-coarsening methods respectively. If we have only a 7-point operator on the finest grid we do not need to store the other 20 coefficients and then the storage requirements become \( \frac{184}{7} N_x N_y N_z \) and \( 46 N_x N_y N_z \) for the standard and semi-coarsening methods respectively.

We now need to address how we are going to handle the storage for the plane solves. Recall that the storage for the 2D black box multigrid method (xy-plane) is
24N_xN_y and 30N_xN_y for the standard and semi-coarsening methods respectively for 9-point fine grid operators, and for a 5-point fine grid operator they will both be reduced by 4N_xN_y.

We now have to decide how we are going to handle the plane solves for the smoother, leading to two obvious choices. The first choice we will call the small storage scheme (SSS) and the other will be called the full storage scheme (FSS). The SSS code will only require storage for the 3D data and enough additional storage for the largest plane, on the 3D finest grid, to be solved using 2D black box multigrid. The FSS will require storage for the 3D data plus storage for all the planes solved by 2D black box multigrid on all the 3D grid levels.

To understand how much storage is required by the FSS it will be useful to refer to figure 14.1. The shading with arrows pointing from a higher level down to a lower level indicates the sub-partitioning of the higher level. The top line of figure 14.1 refers to data partitioning of the 3D grid levels, where grid level m is the finest grid. This is analogous to the 2D data structures used earlier. We have a large array that is partitioned for the storage of each array needed for each of the 3D grid levels. The second line of figure 14.1 refers to the storage for the three groups of plane solves needed by the smoother to perform alternating red/black plane Gauss-Seidel relaxation. The third line of figure 14.1 indicates the number of plane data structures needed for one red/black plane Gauss-Seidel relaxation. The fourth line of figure 14.1 is the data partitioning of the 2D grid levels for a single plane solve by black box multigrid. The total storage for the FSS is

$$\frac{900}{7}N_xN_yN_z$$.

Note that the FSS scheme has included storage on a 3D grid level for all the grid operator coefficients used in the plane solves, but this data is just a copy of the
Figure 14.1: 3D full storage scheme memory data structure
3D grid operator coefficients on a plane. We can eliminate this duplicate storage for identical data. However, this means that we will have to add back in enough storage for the largest plane to hold the 2D multigrid data structures. In addition, we will also have to add routines for copying 3D grid operator coefficients and all the saved 2D coarse grid operator coefficients for a given plane into the newly added storage each time a plane needs to be solved for the smoother. We can call this method the nearly full storage scheme (NFSS), which requires

\[
\frac{56}{7} N_x N_y N_z + 24 \text{MAX} \{N_x N_y, N_y N_z, N_x N_z\} .
\]

storage, while the SSS requires

\[
\frac{324}{7} N_x N_y N_z + 24 \text{MAX} \{N_x N_y, N_y N_z, N_x N_z\} .
\]

storage, where MAX\{ – \} stands for the maximum function. The NFSS only takes about 74% more storage than SSS does, and 60% less than FSS.

The NFSS is even more attractive when we consider the computing time as compared to SSS. The SSS does not save the grid transfer or coarse grid operators, and hence it will have to perform the setup of these operators every time. The 2D setup time is between one and two times that of the execution of one V(1,1)-cycle. The 3D smoother, performing plane solves, for SSS will double its execution time compared to NFSS. Because of vectorization issues the NFSS smoother does not quite perform twice as fast as the SSS smoother, and since the smoother is only one of the multigrid components we only see a speedup factor of about 1.5 for NFSS over SSS.

NFSS will not be quite as fast as FSS because of the additional time needed to copy the grid operator coefficients, but this time should be fairly small if not negligible. The NFSS approach is clearly the winning strategy because it minimizes the storage requirements and maximizes the speed of execution.
We also have the possibility for both an FSS, NFSS, and SSS implementation for the semi-coarsening algorithm. The semi-coarsening storage required for the SSS is

\[66N_xN_yN_z + 30 \text{MAX } \{N_xN_y, N_yN_z, N_xN_z\},\]

while the NFSS is

\[100N_xN_yN_z + 30 \text{MAX } \{N_xN_y, N_yN_z, N_xN_z\},\]

and the FSS is

\[126N_xN_yN_z.\]

### 14.3 3D Standard Coarsening Vector Algorithm

We have discussed several issues concerning the black box multigrid components, vectorization, and programming on the Cray Y-MP. We will explicitly state what choices we made for the vector algorithm, as we did for the 2D vector algorithm.

We have implemented the code being aware of all the vectorization issues and using the most practical and efficient choices that have been discussed so far.

#### 14.3.1 Coarsening

We used standard coarsening, taking every other fine grid point in each of the coordinate directions to form the coarse grid.

#### 14.3.2 Data Structures

The data structures for the grid equations are grid point stencil oriented. The mesh of unknowns has been augmented with a border of fictitious zero equations in the same way as we did in the 2D code. The border is used to avoid having to write special code to handle the boundary of the grid. This arrangement makes the code easier to write and more efficient for vector operations.

There are several arrays to hold the grid equations: the discrete coefficient array, the array of unknowns, and the right hand side array. There are also several
extra auxiliary arrays to hold the grid transfer operator coefficients, the residual, and the 2D plane problems for the smoother. The 2D plane problems contain the storage space for all the 3D coarse grids to be solved by the 2D black box multigrid solver for the alternating zebra plane relaxation.

Each grid level has its own data structure of the appropriate size that has been allocated, via pointers, as part of a larger linear array for each data type structure. This arrangement makes memory management for the number of grid levels easier.

14.3.3 Smoothers We have implemented the multicolor ordering point and alternating zebra plane Gauss–Seidel methods. The plane solves are performed using the 2D black box multigrid method where each plane’s data is copied into the 2D solvers data structures.

14.3.4 Coarsest Grid Solver The coarsest grid solver is a direct solver using LU factorization, which is performed by the Linpack routine SGBSL.

14.3.5 Grid Transfer Operators There are three choices for the grid transfer operators, discussed in chapter 11. They are analogous to the three that were implemented for the 2D standard coarsening method in sections 3.5.1, 3.5.3, and 3.6.1.

14.3.6 Coarse Grid Operators The coarse grid operators are formed using the Galerkin coarse grid approximation, which uses the grid transfer operators and the fine grid operator.

14.4 3D Semi–Coarsening Vector Algorithm

The semi–coarsening code was originally implemented by Joel E. Dendy Jr. We have re-implemented it in a slightly more efficient form to gain a speedup of about 2 over the previous vectorized version while maintaining and improving the portability.
of the code. The new implementation has kept all the functionality of the previous version.

14.4.1 Data Structures The data structures for the grid equations are the same as those for the standard coarsening code including the fictitious border equations. However, we only need storage for the xy-planes used by the smoother.

14.4.2 Coarsening Semi-coarsening in the z-direction was used, taking every other fine grid point in the z-direction to form the coarse grid.

14.4.3 Smoothers Zebra xy-plane Gauss-Seidel relaxation is used for the smoother. The plane solves are performed using the 2D semi-coarsening black box multigrid method.

14.4.4 Coarsest Grid Solver The coarsest grid solver is either the direct LU factorization Linpack solver or a single call to the 2D semi-coarsening black box multigrid method when the coarsening has continued until only one plane is left to solve.

14.4.5 Grid Transfer Operators The grid transfer operator is analogous to the 2D one used in section 3.6.1, but extended to 3D and applied in the z-direction.

14.4.6 Coarse Grid Operators The coarse grid operators are formed using the Galerkin coarse grid approximation, using the grid transfer and fine grid operators.
14.5 Timing Results for 3D Test Problems

In this section we present some timing results of the various codes for comparing the performance of the codes and their various components.

To illustrate how fast these codes perform in solving a problem, we examine the timing results for solving Poisson’s equation for five V-cycles; see table 14.1. Table 14.1 gives the timing results, in seconds, for various stages of the program execution for various grid sizes. The grid is square, so in the first column where \( n = 9 \), we mean a grid of size \( 9 \times 9 \times 9 \) and so forth for the rest of the column entries. The second column gives the total setup time, which involves the time it takes to form all of the grid transfer operators, generate all the coarse grid operators, and perform any decompositions needed for the smoother. The third column gives the total time for smoothing. The fourth column gives the total time for the direct solver. The last column contains the average time it took to complete one \( V(1,1) \)-cycle.

We observe that the code runs fairly quickly, and that it appears to scale with respect to the grid size. However, the scaling is not what we expect, that is a factor of 8 for the standard coarsening and a factor of 2 for the semi-coarsening. We do not see these speedups for several reasons, but the main one has to do with the fact that we are performing five \( V(1,1) \)-cycles. Each of these 3D V-cycles is using alternating plane solves for the smoother, and each plane is solved using the 2D black box multigrid method. Another big reason for the difference in the scaling has to do with vectorization issues.

We also note that the total setup time is about 0.55 times that of the average cycle time, and in addition it is about 0.12 the total smoothing time for one iteration. In the 2D codes we saw just the opposite, where the setup time was greater than the average cycling time, but because of the overhead involved in using the 2D multigrid
Table 14.1. Multigrid component timings for standard (top four lines) and semi-coarsening (bottom four lines) in seconds for various grid sizes versus the time for five $V(1,1)$-cycles. The standard coarsening method uses zebra alternating plane Gauss–Seidel for the smoother, which uses a 2D multigrid method, and the grid transfer operator is the nonsymmetric hybrid collapsing method.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Total Setup $n$</th>
<th>Total Smoothing $n$</th>
<th>Direct Solver $n$</th>
<th>Average per Cycle $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4.671E-2</td>
<td>3.791E-1</td>
<td>5.571E-4</td>
<td>8.466E-2</td>
</tr>
<tr>
<td>17</td>
<td>1.753E-1</td>
<td>1.502E+0</td>
<td>5.610E-4</td>
<td>3.324E-1</td>
</tr>
<tr>
<td>33</td>
<td>7.051E-1</td>
<td>5.916E+0</td>
<td>5.579E-4</td>
<td>1.321E+0</td>
</tr>
<tr>
<td>65</td>
<td>3.031E+0</td>
<td>2.351E+1</td>
<td>5.583E-4</td>
<td>5.281E+0</td>
</tr>
<tr>
<td>9</td>
<td>5.854E-2</td>
<td>2.915E-1</td>
<td>9.234E-3</td>
<td>7.554E-2</td>
</tr>
<tr>
<td>17</td>
<td>2.330E-1</td>
<td>1.495E+0</td>
<td>1.855E-2</td>
<td>2.991E-1</td>
</tr>
<tr>
<td>33</td>
<td>1.073E+0</td>
<td>4.956E+0</td>
<td>3.918E-2</td>
<td>1.317E+0</td>
</tr>
<tr>
<td>65</td>
<td>5.503E+0</td>
<td>2.536E+1</td>
<td>1.006E-1</td>
<td>6.952E+0</td>
</tr>
</tbody>
</table>
method in the smoother and the simplified tensor composition for the grid operator setup, the setup is now faster in 3D. A more detailed examination of these relationships between the various multigrid components is given below.

The rest of the tables in this section give the results for one multigrid $V(1, 1)$-cycle. The results are separated by multigrid components for easier comparison, and each table is further broken down into the type of multigrid algorithm. All times are given in seconds of CPU time on the Cray Y-MP in single processor mode.

The time to perform the LU decomposition of the coarsest grid $(3 \times 3 \times 3)$ problem for the direct solver is $7.148E - 4$ seconds. The direct solver on the coarsest grid level $(3 \times 3 \times 3, \text{standard coarsening})$ takes $1.116E - 4$ seconds. These times are constant for all of the standard coarsening algorithms that use the direct solver.

The amount of work to perform the grid transfers depends on the grid size and on the type of coarsening used. A comparison between standard and semi-coarsening is given in table 14.2. As one would expect, semi-coarsening grid transfers are faster than standard coarsening grid transfers.

Table 14.3 gives the timings results for two standard coarsening smoothers and the semi-coarsening smoother. Note that the time for the alternating zebra plane and semi-coarsening are fairly close, and that the point Gauss-Seidel method is roughly 17 times faster.

Table 14.2. Timings in seconds for multigrid grid transfer components for one $V(1, 1)$-cycle for various grid sizes; comparing standard and semi-coarsening methods.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Standard Coarsening</th>
<th>Semi-Coarsening</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prolongation</td>
<td>Restriction</td>
</tr>
<tr>
<td>9</td>
<td>5.427E-4</td>
<td>5.714E-4</td>
</tr>
<tr>
<td>17</td>
<td>2.513E-3</td>
<td>1.981E-3</td>
</tr>
<tr>
<td>33</td>
<td>7.518E-3</td>
<td>7.560E-3</td>
</tr>
<tr>
<td>65</td>
<td>2.681E-2</td>
<td>2.934E-2</td>
</tr>
</tbody>
</table>
Table 14.3. Timings for the total smoothing time in seconds for one multigrid $V(1,1)$-cycle for various grid sizes and smoothers.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>Total Smoothing Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/BPGS</td>
</tr>
<tr>
<td>9</td>
<td>4.515E-3</td>
</tr>
<tr>
<td>17</td>
<td>1.747E-2</td>
</tr>
<tr>
<td>33</td>
<td>6.879E-2</td>
</tr>
<tr>
<td>65</td>
<td>2.688E-1</td>
</tr>
</tbody>
</table>

The ratio of time spent smoothing versus the time spent doing grid transfers is given in Table 14.4. The ratio of smoothing to grid transfers shows that the smoother is the dominant computation in the multigrid cycling algorithm. It also shows that the use of plane smoothing via 2D multigrid dominates the computations completely.

Recall from section 5.9 that 4-direction point Gauss–Seidel relaxation is a good smoother for isotropic 2D problems using standard coarsening. We can extend this method to 3D problems by using an 8-direction point Gauss–Seidel relaxation that will give us good smoothing and be robust for isotropic problems. The method should also be attractive for both the execution time and memory usage. We can see from Table 14.3 that the execution time for the red/black point method is about one sixteenth of the alternating plane method, and since lexicographic ordering performs nearly identically to red/black ordering on the Cray Y-MP, we should get the 8-direction point Gauss–Seidel relaxation method to perform in half the time that the alternating plane relaxation takes. As an additional bonus the 8-direction point Gauss–Seidel method does not require any extra storage over that required to store the 3D grid equation data structures and the 3D grid transfer operator coefficients.
Table 14.4. Standard coarsening with grid transfer operators based on the hybrid collapsing method. Timing ratios (smoothing/grid transfer) for one $V(1,1)$-cycle for various grid sizes.

<table>
<thead>
<tr>
<th>Grid Size $n$</th>
<th>R/BPGS</th>
<th>AZplGS</th>
<th>SCBMG3</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4.05</td>
<td>68.05</td>
<td>101.8</td>
</tr>
<tr>
<td>17</td>
<td>3.89</td>
<td>66.84</td>
<td>149.2</td>
</tr>
<tr>
<td>33</td>
<td>4.56</td>
<td>78.45</td>
<td>120.2</td>
</tr>
<tr>
<td>65</td>
<td>4.79</td>
<td>83.74</td>
<td>88.07</td>
</tr>
</tbody>
</table>

14.6 Numerical Results for 3D Test Problem 1

Problem 1 is a Poisson problem defined by

\[-\Delta u = f \quad \text{on} \quad \Omega = (0, 32.) \times (0, 32.) \times (0, 32.)\]
\[
\frac{\partial u}{\partial n} = 0 \quad \text{on} \quad x = 0, \text{ or } y = 0, \text{ or } z = 0. \quad (14.3)
\]
\[
\frac{\partial u}{\partial n} - \frac{1}{2} u = 0 \quad \text{on} \quad x = 32, \text{ or } y = 32, \text{ or } z = 32.
\]

We discretize the equation using finite volumes with central differencing. The numerical results are given in tables 14.5 through 14.6.

14.7 Numerical Results for 3D Test Problem 2

Problem 1 is a discontinuous four-cube junction problem defined by

\[-\nabla \cdot D \nabla u + cu = f \quad \text{on} \quad \Omega = (0, 32.) \times (0, 32.) \times (0, 32.)\]
\[
\frac{\partial u}{\partial n} = 0 \quad \text{on} \quad x = 0, \text{ or } y = 0, \text{ or } z = 0. \quad (14.4)
\]
\[
D \frac{\partial u}{\partial n} - \frac{1}{2} u = 0 \quad \text{on} \quad x = 32, \text{ or } y = 32, \text{ or } z = 32.
\]

where the domain is split into the regions,

\[R_1 = \{(x,y,z) : 0 < x < 16, \ 0 < y < 16, \ 0 < z < 16\},\]
\[R_2 = \{(x,y,z) : 16 \leq x < 32, \ 0 < y < 16, \ 0 < z < 16\},\]
Table 14.5. Number of $V(1,1)$-cycles using standard coarsening and zebra alternating plane Gauss–Seidel for the smoother for the hybrid collapsing grid transfer operator, and the first, last and average convergence factor (CF).

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>11</td>
<td>1.223E-1</td>
<td>1.767E-1</td>
<td>2.311E-1</td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>8.981E-2</td>
<td>2.821E-1</td>
<td>2.423E-1</td>
</tr>
<tr>
<td>33</td>
<td>12</td>
<td>4.211E-2</td>
<td>2.645E-1</td>
<td>2.724E-1</td>
</tr>
<tr>
<td>65</td>
<td>15</td>
<td>4.801E-2</td>
<td>3.943E-1</td>
<td>2.876E-1</td>
</tr>
</tbody>
</table>

Table 14.6. Number of $V(1,1)$-cycles using semi-coarsening zebra alternating plane Gauss–Seidel for the smoother for the hybrid collapsing grid transfer operator, and the first, last and average convergence factor (CF).

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1,1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
<td>1.703E-1</td>
<td>2.798E-1</td>
<td>2.556E-1</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>1.886E-1</td>
<td>2.922E-1</td>
<td>2.723E-1</td>
</tr>
<tr>
<td>33</td>
<td>16</td>
<td>2.222E-1</td>
<td>2.955E-1</td>
<td>2.795E-1</td>
</tr>
<tr>
<td>65</td>
<td>18</td>
<td>3.231E-1</td>
<td>2.989E-1</td>
<td>2.876E-1</td>
</tr>
</tbody>
</table>
Table 14.7. Number of $V(1, 1)$-cycles when $c = 0$ and using standard coarsening and zebra alternating plane Gauss–Seidel for the smoother for the nonsymmetric collapsing grid transfer operator, and the first, last and average convergence factor (CF).

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>15</td>
<td>2.523E-1</td>
<td>4.333E-1</td>
<td>3.667E-1</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>3.041E-1</td>
<td>5.324E-1</td>
<td>4.441E-1</td>
</tr>
<tr>
<td>33</td>
<td>27</td>
<td>3.331E-1</td>
<td>6.144E-1</td>
<td>5.329E-1</td>
</tr>
<tr>
<td>65</td>
<td>36</td>
<td>4.801E-1</td>
<td>7.464E-1</td>
<td>6.267E-1</td>
</tr>
</tbody>
</table>

$R_3 = \{(x, y, z) : 0 < x < 16, 16 < y \leq 32, 0 < z < 16\}$,

$R_4 = \{(x, y, z) : 16 \leq x < 32, 16 \leq y < 32, 0 < z < 16\}$,

$R_5 = \{(x, y, z) : 0 < x < 16, 0 < y < 16, 16 \leq z < 32\}$ ,

$R_6 = \{(x, y, z) : 16 \leq x < 32, 0 < y < 16, 16 \leq z < 32\}$ ,

$R_7 = \{(x, y, z) : 0 < x < 16, 16 \leq y < 32, 16 \leq z < 32\}$ ,

$R_8 = \{(x, y, z) : 16 \leq x < 32, 16 \leq y < 32, 16 \leq z < 32\}$ ;

then let

$$D = \begin{cases} 1. & \text{for regions 2, 3, 5, 8} \\ 1000. & \text{for regions 1, 4, 6, 7} \end{cases}$$ , \hspace{1cm} (14.5)

and

$$f = \begin{cases} 1. & \text{for regions 2, 3, 5, 8} \\ 0. & \text{for regions 1, 4, 6, 7} \end{cases}$$ . \hspace{1cm} (14.6)

We discretize the equation using finite volumes with central differencing. The numerical results are given in tables 14.7 through 14.10. The methods all perform roughly the same with the semi-coarsening method coming in last.
Table 14.8. Number of $V(1, 1)$-cycles when $c = 0$ and using standard coarsening and zebra alternating plane Gauss–Seidel for the smoother for the hybrid collapsing grid transfer operator, and the first, last and average convergence factor (CF).

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>15</td>
<td>2.544E-1</td>
<td>4.339E-1</td>
<td>3.679E-1</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>3.038E-1</td>
<td>5.281E-1</td>
<td>4.276E-1</td>
</tr>
<tr>
<td>33</td>
<td>27</td>
<td>3.348E-1</td>
<td>6.159E-1</td>
<td>5.336E-1</td>
</tr>
<tr>
<td>65</td>
<td>36</td>
<td>4.833E-1</td>
<td>7.878E-1</td>
<td>6.308E-1</td>
</tr>
</tbody>
</table>

Table 14.9. Number of $V(1, 1)$-cycles when $c = \frac{1}{3n}$ and using standard coarsening and zebra alternating plane Gauss–Seidel for the smoother for the nonsymmetric collapsing grid transfer operator, and the first, last and average convergence factor (CF).

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
<td>9.112E-1</td>
<td>4.210E-1</td>
<td>3.677E-1</td>
</tr>
<tr>
<td>17</td>
<td>14</td>
<td>1.123E-1</td>
<td>5.223E-1</td>
<td>4.121E-1</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>2.388E-1</td>
<td>6.100E-1</td>
<td>5.629E-1</td>
</tr>
<tr>
<td>65</td>
<td>26</td>
<td>3.522E-1</td>
<td>6.363E-1</td>
<td>6.230E-1</td>
</tr>
</tbody>
</table>

Table 14.10. Number of $V(1, 1)$-cycles when $c = \frac{1}{3\pi}$ and using standard coarsening and zebra alternating plane Gauss–Seidel for the smoother for the hybrid collapsing grid transfer operator, and the first, last and average convergence factor (CF).

<table>
<thead>
<tr>
<th>Grid Size $n$ $(n \times n)$</th>
<th>Iterations $V(1, 1)$</th>
<th>First CF</th>
<th>Last CF</th>
<th>Average CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
<td>9.104E-1</td>
<td>4.209E-1</td>
<td>3.679E-1</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>1.334E-1</td>
<td>5.231E-1</td>
<td>4.276E-1</td>
</tr>
<tr>
<td>33</td>
<td>22</td>
<td>2.241E-1</td>
<td>6.129E-1</td>
<td>5.336E-1</td>
</tr>
<tr>
<td>65</td>
<td>26</td>
<td>3.480E-1</td>
<td>6.376E-1</td>
<td>6.308E-1</td>
</tr>
</tbody>
</table>

323
CHAPTER 15

PARALLEL 3D BLACK BOX MULTIGRID

The parallel 3D black box multigrid methods are similar to the parallel 2D methods. However, unlike the 3D vector versions, the smoother does not use the parallel 2D black box multigrid method in its original form.

15.1 3D Standard Coarsening Parallel Algorithm Modifications

Just like the 2D parallel standard coarsening method, we run into the same problems with performing the tridiagonal line solves. We have chosen to modify the coarsening to use standard coarsening until the VP ratio is less than or equal to one and then switch to the semi-coarsening algorithm.

This approach has several benefits, which include easier coding and faster execution. As will be seen, the semi-coarsening algorithm is the fastest for VP ratio less than or equal to one, but the standard coarsening method is actually faster when the VP ratio is greater than one.

15.2 3D Parallel Smoother

On a parallel computer we could use the same approach as we did on the vector computers by calling the 2D solver, but this would not be very efficient since we know that all the red (black) planes can be solved for simultaneously. Instead,
we can modify the 2D solver to solve all of the red (black) planes simultaneously by introducing the third coordinate axis to the data structure. This is good news because it will cut down on the overhead associated with calling the 2D solver over and over again. It indeed means that we will use more memory for the 2D solver because of the third coordinate axis, but the performance gain justifies this decision.

However, we are again faced with the choice between creating three 2D plane solvers, one for each of the 2D planes needed for alternating plane relaxation, or creating one 2D plane solver and transferring the information from the 3D solver data structures to the 2D plane solver data structures, with a copying routine. The first choice of creating three 2D plane solvers is rather prohibitive for two reasons. First, it requires much more storage for all the data structures needed for the three 2D plane solvers, but it would also be the fastest implementation for performance. Secondly, the size of the final executable code, which is already quite large, more than doubles, meaning that there is that much less storage available for solving large grid size problems. We decided to use only one 2D plane solver, sacrificing some performance gains in order to save memory for solving larger problems. We may have saved some space by not writing three 2D multigrid solvers, but we now have to transfer data between the 3D and 2D data structures. These data transfers require the use of the less efficient communications. In either case, we have to transfer the unknowns and the right hand sides of the grid equations, followed by transferring the solution back, all of which uses inefficient general communications for the yz- and xz-plane solves. By having only one 2D (xy-plane) version we also have to transfer the plane’s coefficient matrix to the 2D fine grid coefficient matrix. If we did not transfer the coefficients we would need nearly double the storage; recall section 14.2.

We have also decided not to save the LU decompositions in the 2D plane solver because of the concerns for the amount of memory it would require with the additional
coordinate (third) axis; hence performance is again reduced. However, experimentation has shown that we only save from 25% to 40% of the execution time required to perform one V-cycle, but it cost four times the storage required for a tridiagonal solve. In actuality, the storage costs is closer to six times because of the temporary work space allocated by the CMSSL tridiagonal solver.

15.3 3D Data Structures and Communication

The data structure allocation and layout are again handled by the use of DMMU. The data structures have the same storage requirements as the 3D vector versions for both the standard and semi-coarsening methods respectively. However, instead of pointers to the various grid level data, we use congruent array aliases, allowing for indexing the desired grid level’s data directly.

15.4 3D Parallel Timings

The following tables we have reported both busy (B) and idle (I) times. Busy time is the execution time for the parallel processing nodes, while idle time is the sequential execution time and also the time to perform all communications. We are reporting times for various time shared partitions of the CM-5. The partitions are identified by the number of processing nodes (PN) namely, 32, 64, and 128 processing nodes. The tables report timings, in seconds, for the average time of five runs for either the setup time or the average of five V-cycles.

The standard coarsening timings are given in tables 15.1 and 15.2 for one V(1,1)-cycle and the setup respectively. We see the affects of the parallel overhead in the tables for small grids sizes and large partitions. The "**" mean that no data was obtained because of a failure in the codes. We believe that the "no data" runs failed because the standard coarsening case has data alignment trouble on coarser grids. The
Table 15.1. Timings, in seconds, for the standard coarsening code performing one \( V(1, 1) \)-cycle with zebra alternating plane Gauss–Seidel on 32, 64, 128 processing nodes of the CM-5, where the size \( N \) means an \( N \times N \times N \) fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size (N)</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td>8</td>
<td>5.003E-1</td>
<td>8.793E-1</td>
<td>**</td>
</tr>
<tr>
<td>16</td>
<td>8.586E-1</td>
<td>1.565E+0</td>
<td>**</td>
</tr>
<tr>
<td>32</td>
<td>1.236E+0</td>
<td>2.982E+0</td>
<td>**</td>
</tr>
<tr>
<td>64</td>
<td>1.687E+0</td>
<td>7.920E+0</td>
<td>**</td>
</tr>
</tbody>
</table>

Table 15.2. Timings, in seconds, for the setup phase of the standard coarsening code with zebra alternating plane Gauss–Seidel on 32, 64, 128 processing nodes of the CM-5, where the size \( N \) means an \( N \times N \times N \) fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size (N)</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td>8</td>
<td>1.684E+0</td>
<td>2.403E+0</td>
<td>**</td>
</tr>
<tr>
<td>16</td>
<td>2.932E+0</td>
<td>4.464E+0</td>
<td>**</td>
</tr>
<tr>
<td>32</td>
<td>4.389E+0</td>
<td>8.911E+0</td>
<td>**</td>
</tr>
<tr>
<td>64</td>
<td>5.650E+0</td>
<td>2.457E+1</td>
<td>**</td>
</tr>
</tbody>
</table>

327
Table 15.3. Parallel efficiency for standard coarsening $V(1,1)$-cycle using zebra alternating plane Gauss-Seidel for the CM-5 with 32, 64, and 128 nodes. The results are given in percentages and $N$ means an $N \times N \times N$ grid.

<table>
<thead>
<tr>
<th>Size</th>
<th>CM-5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32 PN</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>16</td>
<td>65</td>
</tr>
<tr>
<td>32</td>
<td>71</td>
</tr>
<tr>
<td>64</td>
<td>82</td>
</tr>
</tbody>
</table>

alignment trouble only happens when using the DMMU and using the 64 processor or 256 processor partitions. The DMMU was designed and tested for the semi-coarsening algorithm. For the standard coarsening algorithm, the DMMU appears to have trouble in aligning the coarse grid data points with their closely related finer grid points on a subgrid. For the 64 and 256 processor partitions the DMMU can not keep the coarsening confined to the subgrid, as is done in the semi-coarsening code.

As in the 2D parallel timings, we again do not see perfect scaling with respect to the grid size nor scaleup with the processing partition size. The parallel efficiencies are given in table 15.3, and they exhibit the same behavior that was seen in the 2D timings.

Tables 15.4 through 15.6 present the timing data for the semi-coarsening algorithm. However, now we do not see any problem with the DMMU.

Finally, we give a comparison on three different computers for one $V(1,1)$-cycles and a variety of grid sizes. The CM-5 timings are given for the 32, 64, and 128 processing partitions.
Table 15.4. Timings, in seconds, for the semi-coarsening code performing one $V(1,1)$-cycle on 32, 64, 128 processing nodes of the CM-5, where the size $N$ means an $N \times N \times N$ fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size $N$</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td>8</td>
<td>7.256E-1</td>
<td>1.235E+0</td>
<td>7.650E-1</td>
</tr>
<tr>
<td>16</td>
<td>1.264E+0</td>
<td>2.271E+0</td>
<td>1.190E+0</td>
</tr>
<tr>
<td>32</td>
<td>1.793E+0</td>
<td>4.630E+0</td>
<td>1.823E+0</td>
</tr>
<tr>
<td>64</td>
<td>2.412E+0</td>
<td>1.234E+1</td>
<td>2.480E+0</td>
</tr>
</tbody>
</table>

Table 15.5. Timings, in seconds, for the setup phase of the semi-coarsening code on 32, 64, 128 processing nodes of the CM-5, where the size $N$ means an $N \times N \times N$ fine grid. Busy and Idle refer to the parallel and communication/sequential time respectively.

<table>
<thead>
<tr>
<th>Size $N$</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td>8</td>
<td>1.103E+0</td>
<td>1.481E+0</td>
<td>1.415E+0</td>
</tr>
<tr>
<td>16</td>
<td>2.180E+0</td>
<td>2.739E+0</td>
<td>1.909E+0</td>
</tr>
<tr>
<td>32</td>
<td>2.248E+0</td>
<td>5.626E+0</td>
<td>2.872E+0</td>
</tr>
<tr>
<td>64</td>
<td>3.100E+0</td>
<td>1.514E+1</td>
<td>3.340E+0</td>
</tr>
</tbody>
</table>

Table 15.6. Parallel efficiency for semi-coarsening $V(1,1)$-cycle for the CM-5 with 32, 64, and 128 nodes. The results are given in percentages and $N$ means an $N \times N \times N$ grid.

<table>
<thead>
<tr>
<th>Size $N$</th>
<th>32 PN</th>
<th>64 PN</th>
<th>128 PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>63</td>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>32</td>
<td>72</td>
<td>69</td>
<td>68</td>
</tr>
<tr>
<td>64</td>
<td>84</td>
<td>80</td>
<td>77</td>
</tr>
</tbody>
</table>
Table 15.7. Timing comparison between the CM-5 and Cray Y-MP computers for one $V(1, 1)$-cycle in seconds, where N means an $N \times N \times N$ grid. The top entries are for the standard coarsening codes and the bottom entries are for the semi-coarsening codes, and ** means that the problem failed to execute.

<table>
<thead>
<tr>
<th>Size N</th>
<th>CM-5 32 PN</th>
<th>CM-5 64 PN</th>
<th>CM-5 128 PN</th>
<th>Cray Y-MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1.3796</td>
<td>**</td>
<td>1.5956</td>
<td>2.332E-2</td>
</tr>
<tr>
<td></td>
<td>1.9606</td>
<td>2.0520</td>
<td>2.0172</td>
<td>3.742E-2</td>
</tr>
<tr>
<td>16</td>
<td>2.4236</td>
<td>**</td>
<td>2.6378</td>
<td>1.398E-1</td>
</tr>
<tr>
<td></td>
<td>3.5350</td>
<td>3.4320</td>
<td>3.5870</td>
<td>1.825E-1</td>
</tr>
<tr>
<td>32</td>
<td>4.2180</td>
<td>**</td>
<td>4.2850</td>
<td>8.795E-1</td>
</tr>
<tr>
<td></td>
<td>6.4230</td>
<td>6.0050</td>
<td>5.8440</td>
<td>9.107E-1</td>
</tr>
<tr>
<td>64</td>
<td>9.6070</td>
<td>**</td>
<td>7.3220</td>
<td>4.663E+0</td>
</tr>
<tr>
<td></td>
<td>14.800</td>
<td>12.354</td>
<td>10.562</td>
<td>4.696E+0</td>
</tr>
</tbody>
</table>
APPENDIX A

OBTAINING THE BLACK BOX MULTIGRID CODES

The black box multigrid codes and a User's Guide can be obtained via anonymous FTP through MGNet, MGNet's web site, or by contacting the author. MGNet stands for the MultiGrid Network.

MGNet’s FTP site:

casper.cs.yale.edu

use "anonymous" for the username

and your e-mail address for a password.

MGNet has a World Wide Web page that can be accessed via the URL:

http://na.cs.yale.edu/mgnet/www/mgnet.html

The author's E-mail address is:

na.bandy@na-net.ornl.gov

Additional copies of this thesis may be obtained by contacting the author or by downloading a copy from the University of Colorado at Denver Mathematics Department’s Web page via the URL:
http://www-math.cudenver.edu/

Any comments, insights, and suggestions would be greatly appreciated. Thank you.
APPENDIX B

COMPUTER SYSTEMS USED FOR NUMERICAL RESULTS

B.1 Cray Y-MP

Manufacturer: Cray Research Inc.

Hardware Specifics: Machine RHO at Los Alamos National Laboratory

computer = Cray Y-MP 8/64
model = 832
serial number = 1054
number CPUs = 8

clock cycle = 6.0 nanoseconds (166666666 cycles/second)
word length = 64 bits
memory size = 67108864 words (67 MWords)
memory speed = 102.0 nanoseconds (17 clock cycles)
memory banks = 256
memory bank busy = 30.0 nanoseconds (5 clock cycles)
instruction buffer size = 32
number of clusters = 9
Operating System: UNICOS version 7.0.6.1

FORTRAN Programming Environment:

- CF77 version 6.0.4.1
- GPP version 6.0.4.1
- FPP version 6.0
- FMP version 6.0.4.0
- CFT77 version 6.0.4.12 (some are done with 6.0.4.10)
- segldr version 7.0i

Cray Library: Craylib version 1.2

Manufacturer: Cray Research Inc.

Hardware Specifics: Machine GAMMA at Los Alamos National Laboratory

- Computer = Cray Y-MP 8/2048 (M90)
- Model =
- Serial number = 2806
- Number CPUs = 8
- Clock cycle = 6.0 nanoseconds (1666666666 cycles/second)
- Word length = 64 bits
- Memory size = 2147483648 words (2.147 GWords)
- Memory speed = 162.0 nanoseconds (27 clock cycles)
- Memory banks = 256
memory bank busy = 120.0 nanoseconds (20 clock cycles)

instruction buffer size = 32

number of clusters = 9

Operating System: UNICOS version 8.0.3

FORTRAN Programming Environment:

CF77 version 6.0.4.1
GPP version 6.0.4.1
FPP version 6.0
FMP version 6.0.4.0
CFT77 version 6.0.4.10
segldr version 8.0i

Cray Library: Craylib version 1.2

B.2 CM–5

manufacturer: Thinking Machines Inc.

Hardware Specifics:

computer = CM–5 at Advanced Computing Laboratory (ACL),
Los Alamos National Laboratory

number CPUs = 1024 Sparc-2 node CPUs, 4 vector units per node
(4096 vector units) (vector length of 16)
memory = 32 MBytes per CPU node
Four HIPPI interfaces; 120 GBytes rotating storage

Operating System: CMOST version 7.4.0 (based on SunOS 4.1.3U1b)

CM Run-Time-System: CMRTS 8.1

CM-FORTRAN Programming Environment:
  CM Fortran Driver Release 2.2.11
  Connection Machine Fortran Version 2.2 (CMF)
  Compiler runtime library Version: 2.2 (LIBCMFCOMPILER)

CM Scientific Software Library:
  CMSSL version 4.0
  Slicewise runtime library version CMRTS_CM5_8_1_6 (LIBCMRTS)
BIBLIOGRAPHY


