Adaptive OpenCL (ACL) Execution in GPU Architectures

Dan Connors
Department of Electrical Engineering
University of Colorado Denver
Denver, Colorado
Dan.Connors@ucdenver.edu

Kyle Dunn
Department of Electrical Engineering
University of Colorado Denver
Denver, Colorado
Kyle.Dunn@ucdenver.edu

Jeff Wiencrot
Department of Electrical Engineering
University of Colorado Denver
Denver, Colorado
Jeffrey.Wiencrot@ucdenver.edu

ABSTRACT
Open Compute Language (OpenCL) has been proposed as a platform-independent, parallel execution model to target heterogeneous systems, including multiple central processing units, graphics processing units (GPUs), and digital signal processors (DSPs). OpenCL parallelism scales with the available resources and hardware generational improvements due to the data-parallel nature of its kernels. Such parallel expressions must adhere to a rigid execution model, essentially forcing the run-time system to behave as a batch-scheduler for small, local workgroups of a larger global problem. In many scenarios, especially in the real-time computing environments of mobile computing, a mobile system must adapt to system constraints and problem characteristics. This paper investigates the concept of Adaptive OpenCL (ACL) to explore algorithm support for dynamically adapting data-model properties and runtime machine characteristics. We show that certain algorithms can be structured to dynamically balance problem correctness and performance.

1. INTRODUCTION
General-purpose computing on graphics processing units (GPGPU) makes use of a graphics-processing unit (GPU), traditionally purposed only for computer graphics. In the past few years, there has been a rapid adoption of GPGPU parallel computing techniques for both high-performance computing and mobile domains. Open Compute Language (OpenCL) [1] has been proposed as a platform-independent, parallel execution framework capable of scaling across hardware models (CPU, GPU, DSP, and accelerators), generations, and platforms. At the core of OpenCL’s design are both a virtual execution and virtual architecture model, allowing code to be targeted for a specific hardware implementation using run-time compilers. With this promise of scalability, there is significant interest in developing OpenCL-based solutions for many computationally-intensive application domains such as image processing, computer vision, computational photography, and 3D visualization & processing.

However, there are issues in mapping all algorithms to OpenCL development. Typically, OpenCL is used for data-level parallelism, the simultaneous processing of multiple data elements with the same function or task. Data-level parallelism directly contrasts many other types of parallelism in which unique tasks or functions for specific data elements may be assigned. In addition, data-level parallel execution is most commonly carried out on GPU hardware using a rigid scheduling model in which no specific element has priority and the entire set of elements are processed. This rigid execution model prevents the balance of trade-offs between domain-specific characteristics with the utilization and efficiency of the computer system. In short, OpenCL is not currently defined to be adaptive.

To make the case for adaptation in OpenCL, we investigate a specific task in computer vision related to keypoint tracking. Applied to stabilization between video frames. Based on the Scale Invariant Feature Transform (SIFT) [2], interesting points on an image (frame) can be extracted to provide a “feature description” of the scene’s objects. Common uses of feature tracking include object recognition, mapping and navigation, image stitching, 3D modeling, and video tracking. While, most algorithms run to completion, as their solution provides a single answer (potentially from a set of many possible) after performing some amount of computation, feature search and mapping are different. In these search-style algorithms, a fixed amount of time (or other allocation) can be used to explore a solution space. Similar approaches are used in the application of space walking for design alternatives [3] or iterative compilation for optimizing applications [4]. Moreover, in mobile computing environments, there may be real-time processing constraints as well as the need to eliminate unnecessary processing to conserve energy.

Incidentally, an Anytime Algorithm is an algorithm that can return a valid solution to a problem even when interrupted at any time before the completion. Within this class of algorithm, a solution is expected to improve with the available execution time. An anytime algorithm on GPUs [5] has been demonstrated as a feedback-control system to throttle new GPU requests and their precision to achieve controllable computation tardiness. In a similar way, our work is dedicated to exploring an approach known as Adaptive OpenCL (ACL) in the context of anytime algorithms. Our ACL framework implements a programmer interface to dynamically vary OpenCL tasks. Rather than modify the number of kernels or type of kernels deployed, the ACL system modifies the kernel execution at the workgroup level. In this way, the system breaks the rigid requirements that all workgroups of an OpenCL kernel carry out the same expression. Instead, the ACL system dynamically adjusts the expression for each workgroup executed based on runtime behavior.

The ACL deployment exploits two major characteristics: (1) model-specific and (2) run-time behaviors. Model-specific properties have been exploited in load-balancing environments [6], and amount to distributing work based on correlating computation requirements to specific model values. For run-time behaviors, there are specific properties of GPUs that govern processing efficiencies and these must be addressed in any adaptive system. In this paper, we concentrate on the exploitation of model-specific values related to feature matching in video stabilization. While there is a tremendous potential to dynamically enable GPU architectures for all applications, we focus on showing the feasibility of the initial ACL implementation.

The paper is divided into the following sections. Section 2 covers background related to both OpenCL and video stabilization using SIFT keypoint descriptors. Section 3 covers the motivational data of the keypoint matching algorithm and the ACL approach to direct dynamic adaptive behavior at runtime. Section 4 includes experimental results, and Section 5 concludes the paper.
2. BACKGROUND

2.1 Open Compute Language (OpenCL)

The OpenCL language is just one method of describing data-level parallel algorithms. The Nvidia Compute Unified Device Architecture (CUDA) is another model, establishing in 2007, to target the off-loading of general-purpose computations on Nvidia GPUs. While similar, OpenCL is implemented as a platform-independent framework capable of targeting heterogeneous processors. OpenCL applications consist of two components: (1) host program directing management of one or more compute devices using a series of application programming interfaces (API) calls, and (2) kernel program executing on the compute devices. For the purpose of this paper, it is only necessary to review the concept of the OpenCL kernel.

OpenCL C is a superset of a subset of C99, and OpenCL kernels include annotations to specify parallelism and explicitly control memory structures and thread interactions. Figure 1 illustrates an example OpenCL filter kernel to average three neighboring values to generate an output value. This expression can be carried out in parallel by any number of parallel hardware threads.

```c
kernel void Filter(__global float *In, __global float *Out)
{
    //Get the index of the current element to be processed
    int i = get_global_id(0);

    //Computation performed by each workitem
    Out[i] = (In[i] + In[i+1] + In[i+2])/3;
}
```

Figure 1 Example OpenCL Kernel (3-Cell Average Filter)

A programmer designates an OpenCL kernel to be launched on a compute device as an N-dimensional workspace. Each workspace is divided into smaller N-dimensional workgroups scheduled on independent compute units within the compute device. Each workgroup consists of a number of workitems that are directly mapped to hardware threads (contexts) and execute on processing elements. Each workitem has a unique ID, accessible from the kernel using OpenCL builtins such as get_global_id(0). The IDs are used to distinguish the data to be accessed (read or write) by each workitem. Workgroups are described within OpenCL to allow communication and cooperation between workitems, specifically on a designated compute unit with a dedicated physical memory (local memory.)

```
kernel void Example(__global float *In, __global float *Out)
{
    //Get the index of the current element to be processed
    int id = get_global_id(0);

    if (In[id] < 0)  // Case: Divergence
        Out[id] = -In[id]
    for (int i; i < 100; i++) // Case: No Divergence for each workitem
        ...
}
```

Figure 3 OpenCL Workgroup Example Branch Divergence

Figure 2 represents the standard OpenCL execution model of a target compute device. In this view, a compute device must include one or more compute units and each compute unit must have one or more processing elements (on which threads carry out the computation of described workitems.) The overall design of the OpenCL architecture allows a large number of workgroups to be simultaneously scheduled by the runtime system of the compute device, which for example can be a GPU. Each workgroup carries out the same overall kernel computation, but has a designated subset of the input and output space. However, there are execution issues when workitems within the same workgroup execute on different paths. These execution issues for workitems (or CUDA threads) in workgroups (or CUDA blocks) are referred to as branch divergence [7].

Branch divergence can result in significant execution overhead, as hardware that is typically highly optimized for a full set of threads during computation becomes idle. Typically, branch divergence can be the result of data-dependent conditions or when the workitem IDs are used to isolate a specific set of workitems from the rest of the workgroup. Figure 3 illustrates an example of a branch divergence as one branch condition for an individual workitem depends on independent data values. Programmers can sometimes statically restructure their code to eliminate workitem-based branches with divergence. However, true data-dependences are difficult or impossible to resolve. With respect to ACL adaptive processing, branch divergence is an important characteristic to address since the goal is to break the strict execution model in which all workgroups execute the identically composed kernel.

2.2 Example Scoring Algorithm: Video Stabilization With SIFT Keypoint Matching

Scale-invariant feature transform (SIFT) is an algorithm in computer vision to detect and describe local features in images and video frames. There are many applications of SIFT in mobile domains including object recognition, panoramic image stitching, augmented reality (image overlay), mapping and navigation, 3D modeling, gesture recognition, video tracking, and video stabilization.

Video and image stabilization is an important commercial feature for many mobile systems and for professional filmmakers. There are several approaches to image stabilization and the effectiveness of each is dependent on the magnitude of camera motion and the bandwidth of the frequency components. Hardware-assisted stabilization techniques [8] make use of inertial measurement units or accelerometers, increasing the cost of systems. Software image stabilization relies on the ability to track features in the image in order to shift the origin of the image sensor from frame to frame to counteract motion of the camera. To maintain a constant image size during shifting, this technique requires either cropping the image to a reduced size or the image sensor must collect additional pixels in the periphery of the image.

```
kernel void Example(__global float *In, __global float *Out)
{
    //Get the index of the current element to be processed
    int id = get_global_id(0);

    if (In[id] < 0)  // Case: Divergence
        Out[id] = -In[id]
    for (int i; i < 100; i++) // Case: No Divergence for each workitem
        ...
}
```
The number of keypoints is varied to evaluate low to high resolutions (respectively with 280 and 480 cores). GPUs: Nvidia GTX280 and Nvidia GTX480.

Figure 6 shows the baseline performance for evaluating full SIFT matches with varying keypoint size on two OpenCL-enabled GPUs: Nvidia GTX280 and Nvidia GTX480 (respectively with 280 and 480 cores). The execution time only includes kernel execution and not memory transfer to the graphics device. The number of keypoints is varied to evaluate low to high resolutions in frame-to-frame comparisons. Overall, execution time scales linearly with number of keypoints, and even for low resolution (1000x1000 keypoints), the match time can be as high as 1.25 seconds for matching two frames.

```c
__kernel void SiftMatch(int nframe0, __global unsigned char *Frame0, int nframe1, __global unsigned char *Frame1, __global int *d_distance, __global int *d_index, int threshold)

int gid = get_global_id(0);

// Assume defaults for each workitem
int min0, min1, diff, match = -1;
min0 = 1000000;
min1 = 1000000;

// SCAN all Frame1 keypoints looking to generate best match
for(int db_i = 0; db_i < nFrame1; db_i++){
    int sum = 0;
    for(int kp_i = 0; kp_i < 128; kp_i++){
        // MATCH descriptor width:128
        diff = (Frame0[gid*128 + kp_i] - Frame1[db_i*128 + kp_i]);
        sum = sum + diff*diff;
    }
    if(sum < min0) {
        min1 = min0;
        min0 = sum;
        match = db_i;
    }
    else if (sum < min1) {
        min1 = sum;
    }
}

d_distance[gid] = min0;
if(100 * min0 < threshold * min1)
    d_index[gid] = match;
else
    d_index[gid] = -1;  // Match doesn’t pass threshold test
```

Figure 5 OpenCL SIFT Matching

Figure 6 Full Keypoint Matching Performance Nvidia GTX280 & GTX480

3. Adaptive OpenCL (ACL) APPROACH

A preliminary design framework was constructed to direct the adaptive execution of both the SCAN and MATCH computation.

3.1 Adaptive Execution

SIFT for video stabilization doesn’t require exact matches of every keypoint to provide a stable outcome. Figure 7 shows the explicit usage of ACL pragmas to direct the modification of the SCAN and MATCH loops to use run-time limits derived between each frame-to-frame match. The programmer can identify the...
code that has a score to execution time tradeoff, while the ACL framework provides two modifications to the code:

- Grouping of workitems into workgroups based on SCAN and MATCH
- Setting of SCAN and MATCH limits.

```c
// OpenCL programmer guide
#pragma ACL_SCAN
for(int db_i = 0; db_i < nFrame1; db_i++) {
    #pragma ACL MATCH
    for(int kp_i = 0; kp_i < 128; kp_i++) {
        diff = (Frame0[db_i*128 + kp_i] - Frame1[db_i*128 + kp_i]);
        sum = sum + diff * diff;
    }
}
```

Figure 7 Adaptive OpenCL (ACL) Pragma Directives
The directives modify the code to include new per-workgroup arrays and accesses that provide each designated loop with starting conditions and limits:

- `groupScanLimit` – distinct limit on number of frame1 keypoints to scan.
- `groupMatch` – distinct limit on number of descriptor positions to use in match calculation.

Currently we only adjust `groupScanStart` to a random position in the Frame1 set of keypoints. This assures that `ScanLimit` is not used only to scan the first part of the database, which would bias the search to miss keypoints in the last positions of the array.

Figure 8 shows the updated SIFT-match implementation. The responsibility of the ACL Runtime on the CPU is to analyze current-last frame keypoint matches and apply findings to the next-current frame keypoint match.

3.1.1 Adaptive Keypoint MATCH Analysis
To evaluate the behavior for trading off descriptor match width and accuracy, we demonstrate by analyzing the original Lowe’s SIFT keypoint match example. Lowe’s demonstration used three separate image objects (Basmati, Book, and Box) that searched in the keypoint listing of a scene, which included each object (partially covered or observed a different angle). The accuracy of a match is determined by whether the same keypoints are identified with the full length (128) descriptor. Figure 9 shows the accuracy of matching when varying the descriptor width from 1 to 128. The graph clearly shows that there is a tradeoff between using the full descriptor width and accurately matching keypoints. A few interesting results emerge from the differences between in the images being analyzed. The Basmati keypoints are more difficult to match (require greater descriptor positions). For the book and box, a reduced descriptor width, about 40, results in matching 70% of the full set. There are a number of false matches for low descriptor widths, but the data is not shown due to space limitations.

Figure 9 Match Accuracy with Descriptor Width Variation
Figure 10 shows the execution tradeoff for changing the descriptor width to discrete sizes: 32, 64, 96, and 128. The results demonstrate that there is near-linear improvement in execution time based on lowering the descriptor length for the GTX 480. When analyzed in reference of Figure 9, there is a clear opportunity to adapt the SIFT match algorithm by trading off potential matches for savings in execution time.

Figure 10 GTX480 Performance versus Descriptor Width
Figure 11 shows the analysis of four distinct videos (ONDESK, STREET, ROAD, and LAB) that are described (and provided) in reference [9]. These videos have distinct characteristics and vary in number of frames from approximately 50 to 200. Figure 11 shows the percentage of incorrect keypoint matches between frames with a descriptor length of 32 (compared to the baseline.
length of 128) for the first 50 frames of each video. The results show that each video has distinct potential in using a different length of descriptor. For example, the LAB video incurs the most missed keypoints ranging between 20-25%, while the DESK video will only incur well under 10% errors. A key aspect of the video analysis shows that the error of matched keypoints does vary over time in each video when a length 32 descriptor is used. This implies that a length 32 descriptor cannot be deployed statically and thus there is important characteristic for the ACL system to use in adaptation.

Figure 11 Frame-to-Frame Keypoint Matching (ONDES,K,STREET,ROAD,LAB)

### 3.1.2 Adaptive Keypoint SCAN Analysis

To evaluate the behavior of adapting the need to scan the entire set of keypoints between frames, the current ACL system transforms the search for each workgroup of keypoints into a subset based on a random starting point and a reduced count of keypoints. The benefit of reducing the search space comes from potentially trading off unnecessary match calculations when a candidate match has already been located. The issue is always that a good candidate (low score of Euclidean distance) may not be the absolute best until all keypoints are searched.

Figure 12 shows an analysis of each keypoint comparison from Frame0 and Frame1. In this case, the execution time of each keypoint search is divided into three intervals: MATCH, AFTER_MATCH, and NOMATCH. MATCH is the execution time spent in search of a valid match. AFTER_MATCH is the additional execution that a keypoint search requires to finish scanning the remaining keypoints. As the search method of Frame1 is simply a linear scan from keypoint 0 to the last keypoint, very often the video stabilization spends 20-30% of execution time on unproductive computation. Similarly, the case of NOMATCH occurs for keypoint comparisons in which the keypoint from Frame0 is not found in Frame1. In this case, between 35%-60% of the execution time of the SIFT match algorithm is devoted to comparisons that do not result in a keypoint match. For the LAB video, 60% of the computation energy is devoted to unnecessary SCAN and MATCH execution. For the other videos this level is reduced to between 30-35%. Overall, this amount of wasted computation time and energy represents a significant opportunity to reduce or eliminate using intelligent techniques to adapt the search of keypoints at runtime.

Figure 12 Analysis of SCAN Computation in Base SIFT Match

Figure 13 shows an analysis of the 4 videos in which each keypoint workgroup of Frame1 scanned only a subset of the Frame1 keypoints. Both the Max and Min keypoints across all video frames of each video were collected. For example, the Max data points show that when a various random portions of the Frame1 keypoints are scanned, there are opportunities to match keypoints: at 10% random Frame1, between 20-25% keypoints are found in the best case. The matches are compared when 100% of the Frame1 keypoints are used. The percentage of matched keypoints increased as the largest randomly selected subset of Frame1 keypoints are scanned. However, these Max values are considered optimistic in the case of various frames across 50 to 150 frames of video. In the worst case, the Min correct data points show that when only 10% of the Frame1 keypoints are scanned, some frame to frame analysis results only in finding nearly 0% of the keypoints if the whole Frame1 was used. Even in some cases when each workgroup scans a 90% portion of the keypoints, only 65-80% of the correct keypoints are encountered. The average number of keypoints found in each frame for the SCAN setting was left off as the lines of each video overlapped closely. Overall, the results show that there is a direct trade-off between the keypoints scanned and the accuracy of the keypoint match. While random subsets of the Frame1 were selected for each workgroup’s scan, the results show there is further potential is intelligently adapting the search pattern. Currently, the ACL pragmas simply deploy a random starting point into the scanned keypoints and a percentage (70% for example) are used to reduce the match computation time.

Figure 13 Max and Min Keypoints Matched when Scanning Reduced Set of Frame1 Keypoints
3.2 ACL Design
The work presented here aims to show an initial exploration of the ACL concepts. Our initial technique simply assigns the MATCH descriptor length and SCAN percentage based on the keypoints matched. Our framework carries out the following assignment:

- Matched keypoints: descriptor length (32) & 100% SCAN.
- Non-Matched keypoints: descriptor length 128 and 70% SCAN

The overall view for the initial ACL implementation is that keypoints that are matched between frames are known quantities that persist across video frames. For these matches, it is more important to scan the entire incoming (next frame) Frame1 rather than reduce the discovery space. At the same time, matched keypoints tend to have low scores (since they are in fact a match), and using a full descriptor MATCH is unnecessary. Conversely, for discovering new matched keypoints, it is better to use the full length descriptor to gain the confidence that a trusted match has been found, and allow the SCAN space to be smaller (random 70% of search) as a keypoint that is not matched in one frame-to-frame comparison can be identified in the next comparisons.

Figure 14 shows the keypoint matching between frames for all of the videos. The percentage of matched keypoints vary with the video content and over time. However, in most cases, the keypoint matching stays at around 60-70%, indicating that from frame to frame, the ACL system will modify the descriptor length to 32 for 60-70% of keypoints. The LAB video does the descriptor length stay at 128 for the majority of keypoints over the entire video.

Figure 14 Keypoint Matching Over Video Frames

4. EXPERIMENTAL RESULTS
We evaluated the keypoint matching algorithm for performance on the Nvidia GTX480 running OpenCL. For comparison, the ACL system generated random settings for two experimental runs RANDOM MATCH and RANDOM SCAN. We compare these as the baseline to two approaches: ACL MATCH, and ACL SCAN & MATCH.

Figure 15 shows performance versus keypoint matching trade-offs sampled over all four video frames. Each point represents a trade-off taken by one of the four ways. Random techniques make their scan and descriptor length decisions using random settings, while the ACL techniques make decisions based on keypoint matching. The diagonal line indicates a uniform trade-off between execution time (X axis) and keypoint matching correctness (Y axis). Execution time is compared to the base run without any changes to the full keypoint scan and full descriptor (128). The highlight of the results of Figure 15 is that the ACL SCAN and MATCH is able to achieve a high percentage of the matched keypoint matches, while reducing execution time. In fact, 30% of execution time can be traded for just 10% loss of finding all keypoint matches.

Figure 15 ACL Execution Trade-off to Keypoint Matching

5. CONCLUSION AND FUTURE WORK
Overall, we believe exploring Adaptive OpenCL techniques hold great potential to allow programmers to direct dynamic execution by using guided code annotations. We have isolated SIFT keypoint matching in the problem set of video stabilization, but there are numerous applications to extend its utility. Our initial implementation allows the OpenCL runtime to modify execution to select between deploying a full keypoint search and comparison and a modified keypoint SCAN search of 70% and an MATCH keypoint comparison of 32 descriptors down from 128. The overall results show great potential in adaptive OpenCL execution. Compared to random selection of SCAN and MATCH settings, the system achieves a 30% performance savings with only a 10%. For future work, there are a number of ways to extend the level of adaptation of the ACL system to involve real-time constraints and more decision levels of MATCH and SCAN.

6. REFERENCES