The Amdahl Software OpenCL CodeBench Application Framework- Automating the Development and Exploration of OpenCL Solutions on Heterogeneous CPU & GPU Systems

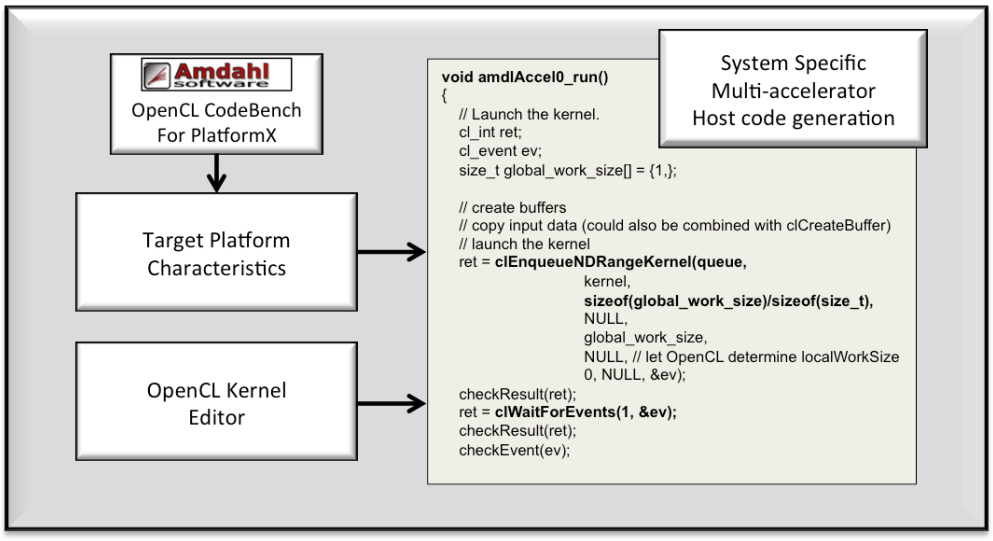
Dan Connors (dan.connors@ucdenver.edu) - Department of Electrical Engineering, University of Colorado Denver
Kurt Prunty (kurt.prunty@amdahlsoftware.com) – Amdahl Software Corporation
Erik Duymelinc (erik.duymelinck@amdahlsoftware.com) - Amdahl Software Corporation

The Open Compute Language (OpenCL) has been proposed as a platform-independent, parallel execution framework capable of scaling across hardware models (CPU, GPU, DSP, and accelerators), generations, and platforms. OpenCL is based on a virtualized parallel execution and architecture model that allows code kernels to be targeted for a specific hardware implementation using a run-time compiler. With the promise of scalability, there is significant interest in developing OpenCL-based solutions for many performance-critical applications in the mobile and embedded systems domains.

At the same time, embedded systems have many unique and conflicting requirements. The need to optimize for power, performance, cost, and real time constraints requires engineers to make best use of the available compute resources in the platform. For embedded application developers, this is especially difficult since resources in each SOC or system tend to be unique. In these embedded platforms, IP is sourced from multiple vendors, bus and memory configurations are nonstandard and operating frequencies and processor capabilities can vary widely. Adding fuel to the fire, applications are also nonstandard. There is no "standard application" that can be used as a benchmark to define the formula for acceleration on a platform.

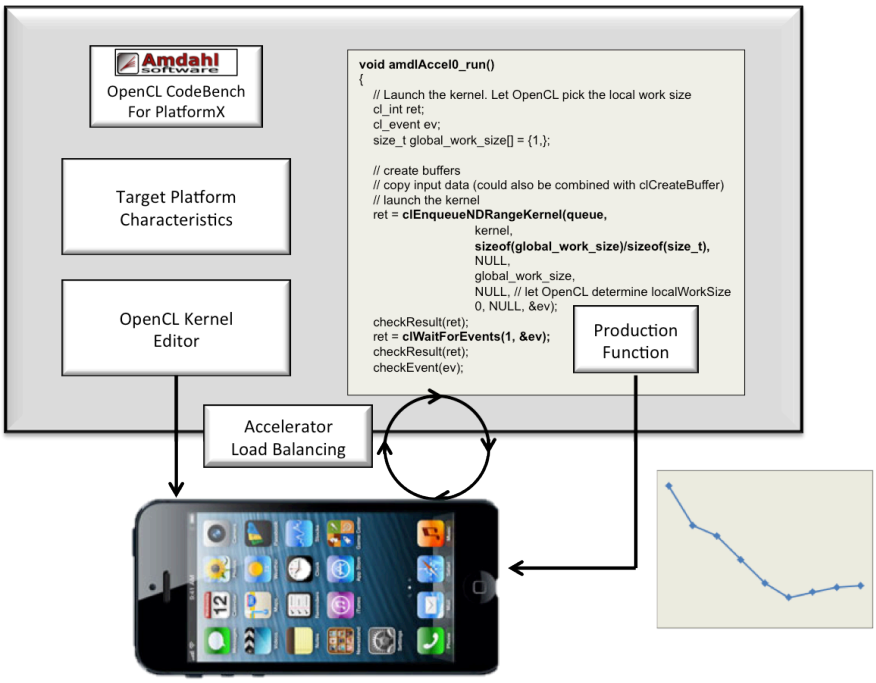
OpenCL CodeBench – Automated Development Framework

Amdahl OpenCL CodeBench guides the development and optimization process for embedded platforms. At the initial stage, CodeBench generates OpenCL host code based on target platform characteristics and high-level description of device memory objects. The automated code can be defined to target unique platforms with various numbers of OpenCL devices. A primary advantage of the systems is that a substantial amount of OpenCL development time can be reduced or eliminated, allowing the developers to focus on kernel optimization. From high-level descriptions, CodeBench automatically generates the OpenCL code interfaces for: *contexts, device command queues, memory objects, programs, kernels, and events* are automatically generated. The generated code is straightforward and easily extendible by end users. The figure below demonstrates a high-level overview of the OpenCL CodeBench development process.



OpenCL CodeBench – Platform Optimization Exploration

Even with capability to generate OpenCL code for multiple devices per platform, embedded system developers are still faced with decisions to making best use of the available CPU and GPU device resources. OpenCL CodeBench provides the additional support to automate these experiments. CodeBench has a productivity tool to enable rapid OpenCL optimization space evaluation through a series of test and analysis runs. It enables engineers to rapidly obtain feedback and determine the most appropriate accelerator allocation strategy for each unique application on each unique platform. The figure below demonstrates the high-level overview of this optimization process for workload distribution exploration.

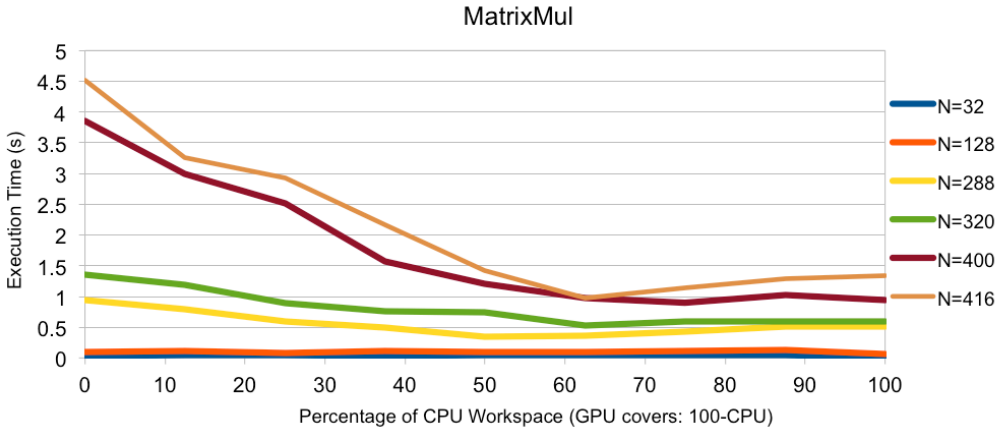


OpenCL CodeBench – Experimental Results Overview

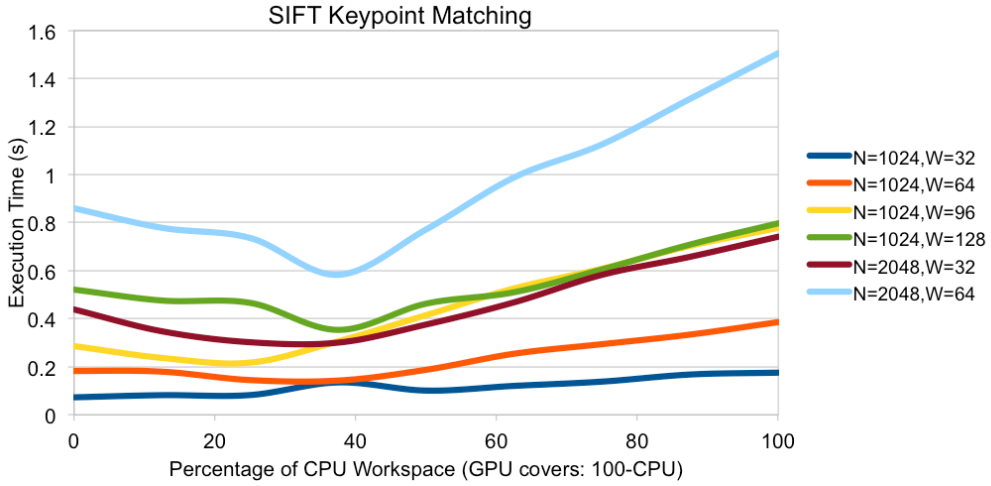
This section analyzes two applications and varies their size and running each on multiple different platforms to ascertain the best accelerator allocation strategy given the available OpenCL compute resources. The applications selects are matrix multiplication (MatrixMul) and scale invariant feature transform (SIFT) keypoint matching. This work focuses on targeting the OpenCL kernel workspace of the applications to the multiple devices of the target platform, factoring in the accelerator mix available, the desired outcome, the platform architecture and system constraints, and does not delve into kernel level optimizations. We focus the analysis on an AMD A8 Quad Core APU with Radeon Graphics, although other platforms such as Intel, NVIDIA, Freescale, Qualcomm, and Samsung have been evaluated.

Each of these algorithms was run in an automated fashion under OpenCL CodeBench, with compute acceleration swept between GPU compute, multiple shared CPU and GPU compute points, and all CPU compute. For each of these points, 100 independent iterations of the algorithm were done, generating an average and eliminating any single event abnormalities. Algorithm sizes were then incrementally increased to evaluate the impact of the larger algorithms of a concurrent type on the results. While these results are only exploring trade-offs in performance, other embedded characteristics can be applied in the process such as power consumption or device maximum latency. In these cases, the work group size may be adjusted to limit the amount of time an algorithm could be allowed to access the accelerator or balance the energy efficiency of the device. In this way, real constraints can be analyzed to provide system vendor with accurate ways to enable downloadable third party applications to be run on their platform.

As expected, results varied by Algorithm, Algorithm size and platform. Interesting results were seen when combinations of GPU and CPU were used for acceleration. The figure below shows the performance of the Matrix multiplication application as the workspace is distributed between CPU and GPU resources. Matrix multiplication is moderately compute-intensive. For this class of problems we found that for very small datasets, the most efficient performance points were seen with all CPU compute. For slightly larger data sets, we can observe that using only the GPU yields the higher performance. For even larger data sets (N), there is a point where one starts seeing benefit in adding CPU compute acceleration to supplement the GPU. From that point, this benefit increased with data size. For larger N, this was seen up until 60% of the algorithm was running on the CPU. At this point, shifting additional compute load onto the CPU reduced performance.



The figure below shows for varying sizes database (N) and varying width (W) of keypoint matching between image frames, the execution time on the AMD machine. In this case, for larger workloads, there is a characteristic break-even point for the AMD machine, in which assigning 40% work to the CPU (and remaining to the GPU/APU) achieves the best execution time.



Conclusion

In balancing conflicting requirements in constrained embedded systems, such as performance, power and limited acceleration access, multiple factors come into play that developers will need to explore. The Amdahl Software OpenCL CodeBench application framework can be used to explore the optimizations that can occur at the platform level through shared use of accelerators when accelerating applications. Results vary with Algorithm size, algorithm type and the platform characteristics of the target architecture. The choice of accelerator and optimization will also be dependent on the goals of the platform, with different mixes of power consumption, performance and latency as goals changing the optimal accelerator mix for the algorithm. OpenCL CodeBench can be used as a productivity tool to enable developers to quickly and easily develop OpenCL applications, enabling system resources to accelerate functions. It can also be used to target OpenCL applications to platforms, optimizing the application and accelerator mix to achieve the platform objectives.