A Framework for Profiling and Performance Monitoring of Heterogeneous Applications

Perhaad Mistry, Yash Ukidave, Dana Schaa, David Kaeli

Department of Electrical and Computer Engineering
Northeastern University, Boston, MA, U.S.A.
{pmistry, yuKidave, dschaa, kaeli}@ece.neu.edu

Abstract. Heterogeneous computing has become prevalent due to the computing power and low cost of Graphics Processing Units (GPUs). OpenCL provides a programming model where the CPU is the master or host, and compute-intensive portions of an algorithm are offloaded to the GPU. However, the host-device model is very limiting. In this model, data-dependent, run-time optimizations that could benefit many applications cannot be easily realized as the cost of transferring intermediate results between devices is high. Besides, implementing run-time optimizations for OpenCL devices would require additional synchronization, profiling and value monitoring code on the part of the application developer.

To overcome this challenge, we present the open-source Haptic (Heterogeneous Application Profiling Tools) framework. Haptic provides extensions to OpenCL that create a closer coupling between the host and device, which allows them to work cooperatively on challenging compute problems. This work discusses the architecture of the Haptic system. The effectiveness of Haptic is demonstrated using signal processing and computer vision applications where performance improvements up-to 40% were seen. The worst case overhead witnessed was 17.5%.

1 Introduction

Parallel programming models such as OpenCL have enabled straightforward development of parallel applications that can run on commodity GPUs. To obtain peak performance for an application the programmer has to be acutely aware of application-specific factors, such as the data layout. An example would be sparse matrix multiplication where data-set details such as the distribution of non-zero values can improve performance by orders of magnitude [1]. Modern algorithms in domains such as computer vision and signal processing have performance that is highly dependent on data characteristics such as noise-level [2, 3].

Due to the complexity of parallel programming, the OpenCL kernel and the compute device are being abstracted away using domain specific languages (DSL) [4] and libraries [1]. Compilation of a DSL or linking with external libraries allows application developers to invoke code optimized for the compute device. However, modularization makes tuning applications harder [5]. This issue is especially of concern for cross-platform specifications such as OpenCL, which target different classes of devices using the same program. When code is interfaced through an API call, it is more difficult for the calling application to specialize the underlying library by tuning parameters or modifying functionality of a library. Also, due to device specific optimizations, specialization usually requires deep understanding of hardware and source code access [1].
In this work, we focus on the interaction between the OpenCL implementation of an algorithm and its use within a high-level application. We propose an architectural resource known as an analysis device whose main role is to carry out computation which can improve performance of an algorithm. Multiple analysis devices can be defined for an OpenCL computation pipeline and a high level application developer can utilize his/her domain expertise to specialize the heterogeneous application. Our work complements existing multidevice research in OpenCL [6, 7], since it allows developers to leverage performance monitoring built on top of OpenCL and also leverage multiple OpenCL devices in scenarios where data cannot be easily partitioned across devices due to irregular memory accesses. Thus, our contributions include:

- We propose a new OpenCL resource, called an analysis device, for inserting optimization and profiling into a compute pipeline.
- We present Haptic, a software framework that implements analysis devices as an extension to OpenCL to specialize algorithms and utilize multiple devices.
- Using discrete and fused platforms, we study the performance of analysis devices when integrated into two applications where host-device behavior is coupled.

2 Limitations of OpenCL Performance Tools

OpenCL is an open standard maintained by the Khronos group, and has received the backing of a number of major graphics hardware vendors. An OpenCL program has host code that executes on a CPU and is responsible for setting up data and scheduling execution on a compute device (such as a GPU). This architecture is shown in Figure 1. The code that executes on a compute device is called a kernel. In-depth information on implementing heterogeneous applications in OpenCL is provided in [8].

The popular vendor-provided performance tools such as the CUDA Visual Profiler and the AMD APP Profiler [8] can help a programmer understand the performance of an OpenCL application. These tools allow a developer to observe which architectural parameters (e.g., branch divergence, number of memory accesses, etc.) are the bottleneck in a kernel. However, vendor-provided profiling tools require a low-level understanding of an architecture to develop an intuition about how profile-based statistics relate to source code. Even for advanced developers, the ability to optimize performance is limited to the data sets evaluated during development, which may not be sufficient for kernels where performance is heavily data dependent. Moreover, GPU performance counters cannot be queried during program execution and therefore cannot be used to help make runtime optimization decisions.
Currently, the only mechanisms for runtime profiling provided by OpenCL are *events*, which record timestamps at different stages of kernel execution and data transfer commands (enqueued, submitted etc.) The programmer can query event objects to make decisions on the execution of the program. However, as events only provide timestamps [8], their utility for profiling is limited.

We target the development of tools that can monitor and adjust algorithms whose performance is data-driven and non-trivial to predict in advance. This class of algorithms are commonly specialized and used as part of a pipeline within larger applications. There is a growing need for performance monitoring and specialization using domain-specific information.

3 OpenCL Analysis Devices

3.1 Architecture and Implementation

A library writer implementing a computational pipeline targeting heterogeneous devices has to write generalized code, leading to non-optimal performance. Developers using such libraries could improve performance using application-specific knowledge (e.g., preprocessing, data or program transformations). Such specializations improve a computational pipeline’s performance, but are challenging to implement since, they require additional platform-specific data management and synchronization (e.g., optimal data placement may change in an APU versus a discrete GPU) [6, 8].

![Fig. 2. Architecture of Haptic’s software stack based on OpenCL. The compute pipeline is called by an High Level Application. The application can provide domain specific information to the pipeline which will be used by the analysis device interface. OpenCL kernel execution on analysis devices is managed by Topology and Profiler modules. Table 1 discusses the role of the modules.](image_url)

To address this problem, Haptic’s software stack defines an architectural resource known as an *analysis device* which interacts with a computation pipeline. For example, an analysis device could be a device that preprocesses data before a stage in the computational pipeline. The potential benefit of such specializations cannot be easily
predicted during library development and thus such specializations cannot be hardcoded into a pipeline. However, application developers can specialize pipelines by leveraging Haptic’s performance monitoring and choosing analysis devices to launch preprocessing functions, change data layout and track values.

Haptic implements an analysis device framework using OpenCL as shown in Figure 2. Two modules, the profiler module and the topology module, maintain the state necessary to introduce specialization into a pipeline. The profiler module provides performance monitoring and maintains a platform-independent performance picture of the OpenCL computation pipeline. Within the profiler module, the event profiler keeps track of executed OpenCL commands and the value tracker is used to monitor data that may provide optimization information. The topology module manages the OpenCL devices available on the platform. The topology module also enqueues analysis kernels for execution on the analysis device and ensures proper coordination with compute devices. Because the topology module exposes the underlying OpenCL command queue (cl_command_queue) [8] as part of its interface, existing OpenCL applications can leverage Haptic with minimal modification. Haptic’s modules are described in more detail in Table 1.

The functionality of the modules in Table 1 is exposed to library developers via the analysis device interface. The API (discussed in Section 3.2) allows developers to specify: value checking and performance monitoring frequency, analysis kernels to launch, and topology of compute and analysis devices which is mapped to the available resources (e.g., CPU and GPU).

<table>
<thead>
<tr>
<th>Component</th>
<th>Role in HAPTIC</th>
<th>Component</th>
<th>Role in HAPTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Profiler</td>
<td>Maintain a vector of cl_event objects. Records command execution timestamps and status provided by OpenCL runtime. Minimize overhead by query events (on the critical path).</td>
<td>Topology</td>
<td>Manage “cl_command_queue” and context interfaces to OpenCL applications. Enables existing OpenCL compute pipelines to leverage HAPTIC modules</td>
</tr>
<tr>
<td>Value Profiler</td>
<td>A value profiler instance records data in OpenCL buffers specified by a user. Implement using calls to OpenCL API functions e.g. clEnqueueMapBuffer</td>
<td>Analysis - Topology</td>
<td>Manage command queue mapping to analysis device. Launches analysis kernels based on Sampling and synchronization with topology</td>
</tr>
<tr>
<td>Sampling Information</td>
<td>Sampling granularity information is required when the analysis device and the value profiler is initialized. To achieve between sampling interval (fidelity) and performance overhead.</td>
<td>Kernel Manager</td>
<td>Manages compilation of the OpenCL kernels to be executed on the Analysis-Topology</td>
</tr>
</tbody>
</table>

a) Main components of profile module  
b) Main components of topology module

<table>
<thead>
<tr>
<th>Component</th>
<th>Role in the HAPTIC system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Kernel</td>
<td>Configure kernel executed on analysis device</td>
</tr>
<tr>
<td>Inject-Analysis</td>
<td>Function that launches analysis kernels based on “Analysis-Topology” defined and the profiling granularity chosen</td>
</tr>
<tr>
<td>Create Buffers</td>
<td>OpenCL cl_nmem buffer which serve as input to the analysis kernels</td>
</tr>
</tbody>
</table>

c) Interface exposed to library developers to build analysis devices that can be integrated into their pipelines and leverage the profiling and topology capabilities provided by HAPTIC

**Table 1.** The main components of Haptic and their role in heterogeneous applications.

The low-level nature of OpenCL provides correctness portability, but not performance portability, due to differences across device architectures. OpenCL-runtime calls
have varying performance on different devices. We leverage the capabilities of architectures such as AMD Fusion and a discrete GPU by studying vendor SDK microbenchmarks for different data locations and using the right runtime calls for data management (e.g., `clEnqueueMapBuffer` vs `clEnqueueReadBuffer`). Interaction between analysis devices and computation pipelines could be improved through driver support [9], but our focus is to present an architecture-neutral interface using OpenCL.

### 3.2 Use of Analysis Devices

Library developers can use Haptic’s API within a compute pipeline to enable possible specializations at runtime. A usage example of Haptic’s API is shown in Figure 3. The example shows the source modifications required to add a data-preprocessing step before a stage in a compute pipeline.

---

**Fig. 3.** Preprocessing device implemented by inheriting from the `analysis_device` class. The “Preprocessing” kernel and topology details are provided to Haptic by the library developer. HAPTIC shows API utilized by developer. The application developer can choose preprocessing.
The analysis_device base class in Figure 3 provides an interface to a user to build analysis devices. The base class interface maintains the necessary OpenCL state within the respective modules (Figure 1). To define a new device as shown in Figure 3 (p\_device), the library developer creates a new class by inheriting from the analysis_device base class. The declaration of p\_device contains the relevant OpenCL buffers and invocations to Haptic’s APIs. The Haptic-provided function inject\_analysis() (Figure 3) synchronizes the compute and analysis command queues and launches the analysis kernel. The function init\_device() is necessary to configure the topology used by p\_device and the profiler’s sampling intervals. Simpler APIs would result in fixed sets of buffers or fixed OpenCL topologies. This would reduce the applicability of Haptic. The API we present does not require detailed OpenCL knowledge or the implementation of threads or multi-device synchronization.

The programming paradigm followed by Haptic has gained acceptance in well-known open-source projects such as Ceres (a non-linear optimization toolchain) [10], where users implement their non-linear optimization problems by reusing and extending base-class interfaces. This methodology allows usage of architecture-agnostic performance information such as required Quality Of Service (QOS) and acceptable error rate to optimize computation.

To summarize, the effort of specialization is divided between the library developer and the application developer. The library developer can build and expose multiple analysis devices (using Haptic’s interfaces), while application developers can choose to use these devices based on domain experience. Due to the object-oriented (C++) nature of Haptic, library developers can also implement custom synchronization schemes (e.g., waiting on non-OpenCL applications or subsets of command queues) by over-riding the base class. Usage scenarios of analysis devices and their implications on application performance are covered in Sections 4 and 5 respectively.

4 Evaluation Methodology

To evaluate the potential benefits of Haptic, we have used analysis devices to improve the performance of two heterogeneous applications. The first application is an adaptive Finite Impulse Response (FIR) filter. The second is a more complex computer vision application, called Speeded Up Robust Features (SURF).

4.1 Finite Impulse Response (FIR) Filter

FIR filters provide linear-phase filtering and are widely used in digital signal processing. The input signal to an FIR filter is split into chunks, called blocks, which are processed together by a kernel. Each element in the block is multiplied by a number of coefficients, called taps, producing an output block (i.e., an output signal). The number of taps for the filter decides the sharpness and stopband attenuation characteristics. For the OpenCL kernel, this affects the memory usage and amount of computation per block.

Adaptive filtering is a filtering technique implemented by changing the weight of the taps for the FIR filter based on signal characteristics. Adaptive filters are used in audio filtering, speech recognition, and pulse detection applications. We define an analysis device using Haptic that interacts with the FIR filter and updates the tap weights to perform adaptive filtering. The Haptic implementation of the FIR benchmark is shown in Figure 4. The performance improvements obtained when using analysis devices for adaptive FIR filtering is discussed in Section 5.1.
Fig. 4. The FIR OpenCL kernel is executed for each block of the input signal. The FIR pipeline is modified to add an analysis device that performs adaptive filtering by modifying filter tap weights.

4.2 Speeded Up Robust Features (SURF)

The SURF (summarized in Figure 5) algorithm generates features from images that are strongly invariant to changes in scale, orientation and intensity in the image. SURF represents a large class of algorithms where profiling and profile-guided optimization can play a role in tuning performance. We modified a publicly available, optimized OpenCL implementation [11] of SURF to serve as the computational pipeline used by Haptic. SURF is commonly used as a component in computer vision applications, such as image search and video stabilization [2].

Applications that use SURF have specific needs, and the algorithm may perform sub-optimally if the requirements of the application are not taken into account. For example, using application-specific knowledge, data transformations can be used to improve spatial locality. Also, in some cases, portions of the computation pipeline can be disabled entirely. To demonstrate Haptic and improve the performance of SURF, we have implemented analysis devices that take advantage of application-specific information to perform useful optimizations. We describe below, the implementation of analysis devices. The performance impact on applications is discussed in Section 5.2.

Fig. 5. The SURF Algorithm and its usage as a computation pipeline in three applications.
Disabling Unnecessary Kernels: Image processing pipelines commonly contain preprocessing stages to eliminate redundant data. Similarly, for a video consisting of static image scenes across frames, the features produced by SURF will not change. To take advantage of this fact, we implement an analysis device that performs a preprocessing step of comparison between frames, and disables the entire feature generation pipeline if frames that do not change significantly (Figure 5a). The benefits of throttling execution of the SURF pipeline include power savings and increased frame-rate.

For a video feed where the camera’s focus and view does not change, the orientation of features do not change. This allows us to implement an analysis device that monitors the change in orientation across features (Figure 5b). If the analysis device does not see a variation in orientation then, corresponding compute stages can be disabled.

Data Monitoring and Transformations: By counting the number of features generated by SURF across a window of frames, we can detect scene changes. We evaluate the performance of an analysis device that uses the value tracker module for OpenCL buffers. Using the value profiler module, we monitor the buffer that stores the feature count (Figure 5b). This device has also be used to modify the threshold input parameter if a consistent number of features are required across frames. The number of features detected affects execution time, this technique can be used to provide a consistent frame-rate regardless of data.

5 Performance Results

We evaluate the performance benefits of Haptic using the FIR filter and applications based on SURF. The OpenCL-capable heterogeneous architectures consisting of multiple device categories are 1) Discrete GPUs connected to a multi-core CPU device using a PCI-Express bus 2) AMD Fusion platforms where the CPU and GPU device share a single system memory [12]. The evaluated architectures are summarized in Table 2.

<table>
<thead>
<tr>
<th>Haptic Supported OpenCL Platforms</th>
<th>Available OpenCL Devices on Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon W3565 - GPU</td>
<td>No. Com. Unit</td>
</tr>
<tr>
<td>AMD Radeon 5870 - GPU</td>
<td>8</td>
</tr>
<tr>
<td>A3850 - CPU</td>
<td>4</td>
</tr>
<tr>
<td>AMD Radeon 5850 - GPU</td>
<td>4</td>
</tr>
<tr>
<td>AMD Radeon 5870 -GPU</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 2. Platforms tested with Haptic. Any OpenCL device within the platform can be used as the analysis or the compute device

5.1 Results - Haptic with Adaptive FIR Filter

In this section, we discuss how the adaptive FIR filter benefits from an analysis device implemented using Haptic.

Three cases of the tap weight modification kernel (100, 200 and 300 iterations) have been executed to view the effects of increasing amount of time spent on the analysis kernel tap-weight-modify with respect to the FIR kernel. The baseline, two-command-queue implementation uses one command queue for the FIR filter and one to provide the tap changes. A single command queue for both the FIR and the tap
a) FIR execution on a APU, Tap weight modification on CPU. (without Haptic)

b) FIR execution on a APU, Tap weight modification on CPU. (with Haptic)

Fig. 6. Execution time of the FIR compute and tap weight modification application (Figure 4) on a APU device with and without Haptic. FIR kernel executes 1000 filter blocks.

5.2 Results - Haptic in SURF Applications

Frame Preprocessing Analysis Device: The preprocessing analysis device (Figure 5a) executes a simple comparison kernel on incoming video frames to disable the feature generation pipeline for similar images to improve the frame rate. Figure 7a shows improvements in execution time.

As expected, the performance of the APU platform improves with larger thresholds. A larger threshold denotes that the difference between two consecutive frames must be larger in order to launch feature generation. To maintain QOS, we use realistic thresholds ensuring that no object visible in the source video is omitted when frames are dropped. The improvement shown includes the time spent in the comparison of video frames. The improvement is proportional to the size of the video and the number of
frames executed after thresholding. Similar performance improvements are seen for a discrete GPU system as well.

**Comparing Orientation:** The orientation device (Figure 5b) allows us to study the impact of changing the sampling frequency of the profiler. This device uses the orientation data produced by the SURF pipeline and disables the orientation stage for a fixed number of incoming frames if the orientation between features did not change. Figure 7b shows the results as the sampling interval is reduced.

Intuitively, disabling the orientation stage should improve performance. However, the improvement effect is small in this case since the kernels that would have been skipped in the orientation stage only make up 15% of the execution time. This scenario, where profiling is active but the resultant directed optimization does not provide substantial benefit, allows us to measure overhead. The challenging threshold disables the orientation stage only a few times in each video. From Figure 7b we observe that the worst-case overhead (smallest video, highest sampling frequency) a 17.5% slowdown.

**Feature Count Tracking Application:** The SURF algorithm's “threshold” input parameter affects the number of features generated and thus performance. We use the analysis device implemented in Figure 5b to track feature counts and the threshold parameter. This device has been applied in two disjoint scenarios. The first use case is to track variation in the number of features generated to automatically detect scene changes. The results for a single video sequence are shown in Figure 8a. By maintaining a window of 10 frames where the average feature count is tracked by the value tracker, changes in the scene can be detected by comparing the window’s average to the feature count of the processed frame. The second use case of feature tracking is a form of feedback parallelism to maintain a constant number of features per frame. Based on the number of features being generated, the threshold can be changed (Fig 8b) to reduce variation in the number of detected features.

Performance in FPS is shown in Figure 8c. Substantial variation in FPS occurs natively without Haptic and while detecting scene changes (due to varying feature counts). Varying the feature threshold parameter stabilizes performance. As seen, Feature count
tracking for scene change adds no significant overhead since very little data needs to be moved and only a running sum and comparison is required on the CPU.

6 Related Work

Domain-specific languages are gaining popularity for Heterogeneous Computing [4]. Petabricks [14] allow us to express a range of different algorithms to solve a problem. Petabricks provides an API that allows a programmer to write the same problem using different implementations. The auto-tuning component of Petabricks shows excellent promise for multi-core CPU performance but does not discuss generating code for GPUs. Teleport Messaging [13] is a language construct in StreamIT (a stream programming framework) that enables sending control messages in a distributed memory environment. By using a synchronous data flow model, StreamIT programs benefit from powerful compiler optimizations. Our work has a similar aim of injecting control information but caters to a challenging environment of heterogeneous computing.

Related work in runtime data transformations on GPUs is specific to molecular dynamics [3]. Some programming frameworks simplify writing scalable multi-device code [6, 7] by presenting high level abstractions which simplify programming. Profiling and specialization extensions discussed here are complimentary to such frameworks. Our work is motivated by architectural exploration such as architectural support for profiling [15] and continuous optimization using profiling [5].

7 Conclusion

In this work, we demonstrate the feasibility of greater interaction between the OpenCL host and device by defining a compute resource known as an analysis device. This methodology is suited for heterogeneous applications where irregular data level parallelism does not permit splitting of a single kernel’s computation across multiple devices. The interface shown can be used by non-experts in parallel programming to improve application performance. The emergence of massively parallel programming and popularity of GPUs has encouraged the use of brute force computation as opposed to more
fine-tuned and specialized solutions. Using Haptic we show that a coupled heterogeneous parallel system with tighter control on the offloaded computation can be a path to more effective and extensible applications in the future.

Future work will include leveraging device partitioning provided in OpenCL 1.2. We also plan to investigate the architectural and language support for interaction between compute devices announced in the HSAIL specification [16].

8 Acknowledgements

This work was supported in part by NSF Award EEC-0946463, and through the support and donations from AMD and NVIDIA.

References