Case Studies Using the Open Compute Language Specification

Brian Sroka, Andrew Pakalnietis, Joseph Francoeur
The MITRE Corporation
bsroka@mitre.org, pakalnietis@mitre.org, jfrancoe@mitre.org

Introduction

In this talk we will present a number of kernel and application case studies using the Open Compute Language (OpenCL) specification. Specifically we will compare and contrast the specification to the popular Common Uniform Device Architecture (CUDA) that is used to program Graphic Processing Units from NVIDIA. We will present details on the performance, scalability, and portability of the OpenCL software along with the impact to software lines of code.

Background

While modern multicore microprocessors and accelerators continue to provide increased, theoretical performance, it is more difficult to realize these gains in software and only at the cost of greatly increased development complexity. The increase in complexity is primarily caused by two factors. The first problem is that there has not existed, until the definition of the OpenCL specification, a standard language or interface for programming multicores and accelerators. Historically, it took significant efforts to develop optimized libraries for just one target architecture. An effort necessarily repeated for each new microprocessor or accelerator. A second problem is that while co-processors are intended to run concurrently with host application software, synchronization beyond the shared-memory boundary has typically been done by utilizing message passing which is tedious to develop. Interfaces, libraries, and compiler tools have varied across different accelerators, making it difficult to generalize the solutions thereby impacting software portability and maintenance.

The OpenCL standard has effectively addressed the first problem by providing utilities to query the underlying microprocessor or accelerator architecture. Architecture features such as memory and cache sizes, vector widths, and the number of processing and compute elements can now be queried through OpenCL functions. While careful and complex programming is still required, the resulting software is not only portable across various hardware architectures but also optimized.

The second synchronization or task management problem is only partly addressed by the OpenCL standard within the shared-memory boundary and is not addressed at all by OpenCL beyond the shared-memory boundary. The OpenCL programming model is presently limited to the main processing unit and directly attached processors (such as Graphics Processing Units). The main processor generally acts as a controller over attached processors. This limit is imposed by OpenCL memory model where all attached processors must be able to access the main processors memory. However, many computation-intensive applications require more processors than can be generally contained in an OpenCL unit. So while there exists support for command queues and specifying jobs dependencies, it does not extend past the shared memory limitation of OpenCL into the distributed application environment required.

In the realm of parallel processing, there have been numerous support libraries, languages and extensions, and tools developed for multiprocessor architectures based on shared memory. These facilities do not apply well or at all when there are multiple processors that do not have shared memory, as is the case with the OpenCL environment. It is left to the application developer to write discrete software units that must be distributed to all other processors using tedious software facilities for identifying available resources, dispatching and activating code, and sending and receiving results. Frameworks such as MPI, Corba, RMI, and Erlang internal actor-to-actor communications have been developed for message passing, but this only addresses part of the problem and still saddles the developer with many details to manage.

Technical Approach

Two technologies have been examined to address the multi-node-OpenCL problem. First, the Message Passing Interface (MPI) is being used to combine multi-node OpenCL environments. This approach has been used in the past to join shared memory OpenMP environments into larger computation clusters to tackle computational fluid dynamics applications such as weather and climate modeling, and/or weapon and armor design. This development environment is challenging but provides baseline metrics for the state of the practice for many HPC applications.

The second technology examined to manage the synchronization and task management challenges in a multi-node-OpenCL environment is GridGain. GridGain is a High Performance Cloud Computing software infrastructure that enables development of highly scalable distributed applications that perform on any grid or cloud infrastructure. GridGain provides state of the art implementations for computational grids, data grids, and cloud auto-scaling. The appeal of GridGain is that it provides many features that make development of highly scalable distributed applications easier and more productive than the MPI alternative.

Case Studies

To compare and contrast OpenCL to CUDA, the MITRE team developed two kernel benchmarks. One performs a covariance calculation and the second performs a Fourier transform. For the covariance benchmark the team developed both an OpenCL and a CUDA implementation. For the Fourier benchmark the team developed an OpenCL implementation and compared the results to cuFFT, an open source CUDA implementation.

To help evaluate the two multi-node-OpenCL approaches, the team is actively developed two OpenCL-optimized real-world applications. The first is a moving target indicator radar (MTI) application. The MTI software was initially written in Matlab and our analysis required the translation of the code to ANSI-C prior to the application of OpenCL. Preliminary results for the OpenCL MTI application demonstrate an order of magnitude improvement provided by the optimization.

The second application concerns fingerprint matching algorithms. This software is written using CUDA technology. CUDA shares many similarities with OpenCL and this software translation was considerable more straight-forward. As we move forward into a multi-node-OpenCL solution, the team plans to measure the overhead of incorporating both MPI and GridGain into these applications where appropriate.

References

[1] The Open Compute Language Specification URL: http://www.khronos.org/opencl/

[2] Common Uniform Device Architecture URL: http://www.nvidia.com/object/cuda_home_new.html

[3] Gridgain URL: http://www.gridgain.com/