

Task Sharing: Collaborative CPU-GPU Execution



Michael Boyer, Kevin Skadron
 University of Virginia Department of Computer Science
 boyer@cs.virginia.edu
 SRC Tasks 1607.001 and 1972.001

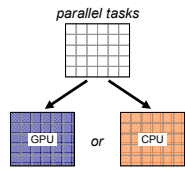


Abstract

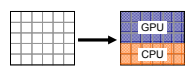
In recent years, there has been growing interest in leveraging the enormous computational power of graphics processors for accelerating general-purpose applications. This generally requires decomposing an application into serial and data-parallel phases, with the parallel phases expressed as *kernels*. Typically the GPU is idle during the serial phases of the program while the CPU is idle during the data-parallel kernels. For kernels in which the two devices provide comparable performance, performance can be improved by allowing the two devices to work collaboratively. We are currently developing a software framework, based on OpenCL, to automatically divide a kernel's execution across the CPU and GPU with minimal programmer effort.

Overview

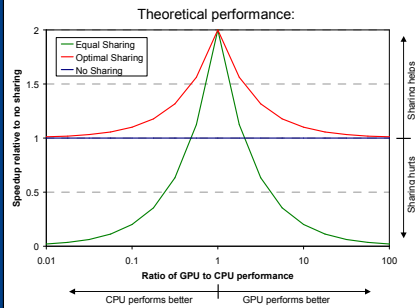
Old approach: execute kernel on CPU or GPU exclusively



New approach: divide kernel across CPU and GPU



Motivation



Lesson: performance improvement is possible, but so is performance degradation!

OpenCL

OpenCL is a programming model for heterogeneous systems that is supported by more than 30 companies, including SRC member companies AMD, Freescale, IBM, Intel, and TI.

Serial code executes on the host device (CPU) while data-parallel kernels can be executed on other devices in the system (GPU, DSP, or even on the CPU). The programmer only needs to write one version of the kernel, rather than a different version for each device.

Executing an OpenCL kernel involves the following steps:

- 1) Choosing a device.
- 2) Compiling the kernel for that device.
- 3) Transferring input data from the host to the device.
- 4) Executing the compiled kernel on the device.
- 5) Transferring output data from the device to the host.

Task Sharing in OpenCL

Task sharing can be accomplished manually, but requires significant programmer effort. Our goal is enable task sharing automatically with minimal programmer involvement.

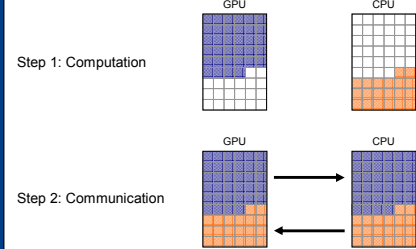
We propose extending the five steps above as follows:

- 1) We extend the OpenCL device model to include a virtual device which represents a collection of heterogeneous devices (a CPU and a GPU).
- 2) The kernel is compiled for each physical device separately and is analyzed to determine which input and output data is accessed by each thread.
- 3) Based on the analysis above, the necessary input data is transferred to each device. The transfers may be deferred until after the kernel division is determined in the next step.
- 4) The kernel is divided across the two physical devices. An appropriate division can be determined in three ways:
 - Dynamically, using a training run (executing the kernel on both devices) and computing the optimal division.
 - Statically, based on analyzing the kernel source code.
 - Using a combination of the two techniques, with the static analysis providing an initial division that is improved dynamically based on run-time feedback.
- 5) Data is transferred between devices as necessary. For non-iterative applications, this typically involves combining the CPU and GPU output into a single host-side array. For iterative applications, this is more complicated; see "Data Access Patterns" for more information.

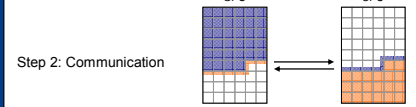
The goal is for the programmer to only need to make a single modification to the source code: selecting the new virtual device instead of one of the physical devices. The rest of the code remains unchanged, interacting with the collection of heterogeneous devices as though they were a single device.

Data Access Patterns

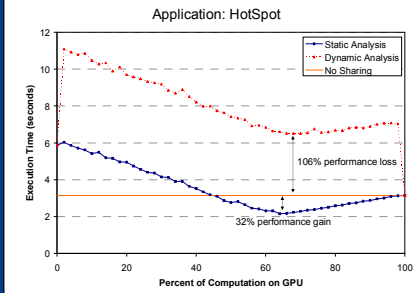
For iterative applications, if we cannot determine data access patterns statically, then the CPU and the GPU must have a consistent view of memory before each kernel launch.



If we can determine data access patterns statically, then we can minimize the communication overhead



Preliminary Results



Future Work

- Finish implementing task sharing framework.
- Measure performance impact on a diverse set of applications.
- Compare different algorithms for kernel division.
- Extend framework to support more than two devices.

Related Work

Qilin [1] and Merge [2] allow a programmer to distribute computations across multiple heterogeneous devices. Both systems require the programmer to write different code for each device. Merge is restricted to map-reduce style applications.

Same Program for All Processors (SPAP) [3] is a programming language that can be mapped to multiple devices in a heterogeneous system. SPAP requires the programmer to either use a small set of language constructs that have already been implemented on the supported devices, or implement a new operation separately for each device.

CUDASA [4] extends NVIDIA's CUDA programming language to allow a kernel to be executed across multiple GPUs. CUDASA requires the programmer to explicitly divide the work among the multiple devices.

- [1] C.-K. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.
- [2] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: a programming model for heterogeneous multi-core systems," *SIGOPS Operating Systems Review*, vol. 42, no. 2, pp. 287-296, 2008.
- [3] Q. Hou, K. Zhou, and B. Guo, "SPAP: A programming language for heterogeneous many-core systems," Zhejiang University Graphics and Parallel Systems Lab, Technical Report, January 2010.
- [4] C. Muller, S. Frey, M. Strengert, C. Dachsbacher, and T. Ertl, "A compute unified system architecture for graphics clusters incorporating data locality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 605-617, 2009.

Technology Transfer

Liaison: Sriatha Manne (AMD)

Acknowledgments

This work is supported by an SRC GRC Fellowship, SRC grants 2007-HJ-1607 and 2009-HJ-1972, and equipment donated by AMD and NVIDIA. We would also like to thank Chris Gregg and Marisabel Guevara for their helpful feedback.

Biography



Michael Boyer was awarded the Bachelor of Science Degree in Computer Engineering from Union College in 2006. As an undergraduate he received several awards, including the Loughry Prize for best senior project in Computer Engineering. He participated in the pilot for the SRC Undergraduate Research Assistants Program, presenting his research at the Graduate Fellows Conference in 2004. He has also completed three internships with Intel. He is currently a Ph.D. student at the University of Virginia, where he is researching software support for heterogeneous architectures under the guidance of Kevin Skadron. He is supported by an AMD/Mahboob Khan Fellowship from SRC.