

# Automatic Intra-Application Load Balancing for Heterogeneous Systems



Michael Boyer, Shuai Che, Kevin Skadron  
University of Virginia

Nuwan Jayasena, Jayanth Gummaraju  
AMD

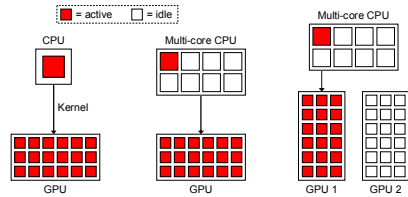
Contact: boyer@cs.virginia.edu  
SRC Tasks 1607.001 and 1972.001



## Abstract

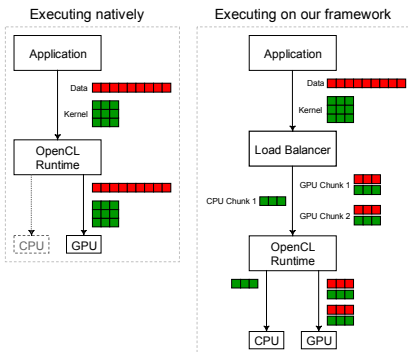
Many OpenCL applications only leverage a single device, typically a GPU, for kernel execution. The CPU and any other GPUs remain unutilized, wasting much of the system's performance potential. Unfortunately, writing applications to leverage multiple devices is challenging. To address this, we are developing a software framework that automatically divides kernel execution across multiple devices. We partition the kernel's data domain into smaller chunks, and use a history-based feedback mechanism to dynamically load balance the chunks across the CPU and GPUs. This approach improves code portability among disparate systems, and raises the level of abstraction for the programmer by eliminating the need to manually manage work distribution among multiple devices.

## Motivation



Many OpenCL applications effectively utilize only one CPU core and one GPU. On systems with multiple CPU cores or multiple GPUs, lots of computational power remains idle.

## Overview



When an application launches a kernel that would only leverage a single GPU, our framework transparently load balances the kernel across all of the available devices, whether those be a CPU and GPU (as shown above) or multiple GPUs.

## Framework Components

### (1) API Intercept Layer

Intercepts each OpenCL API call from the application and transforms it as necessary. For example, if the application requests the creation of a memory buffer in GPU memory, the intercept layer will transparently create both a GPU and a CPU version of the buffer.

### (2) Access Pattern Extractor

Analyzes the kernel source code to determine a mathematical mapping from chunk to memory region, for each memory buffer accessed by the kernel. The scheduler later uses this mapping to determine the memory regions to transfer for each chunk.

### (3) Chunk Scheduler

Dynamically schedules chunks to all available devices. Begins by sending small chunks to each device and then uses online profiling information to efficiently schedule the remaining chunks.

## Dynamic Chunking: Benefits

- No modifications required to existing OpenCL applications
  - Code is portable to different heterogeneous systems
- No framework training required
- Can respond to dynamic performance changes due to:
  - Data-dependent kernel performance
  - Changes in hardware capabilities (voltage scaling, etc.)
  - Contention from other applications
- Can overlap data transfer and kernel execution (see below)

## Dynamic Chunking: Challenges

- Effectively dealing with contention:
  - Contention from other applications
  - CPU executes both serial host code and parallel kernels
  - Memory contention between CPU and GPU operations
- Effectively dealing with increased dispatch overhead due to increase in number of data transfers, kernel launches, etc.
- Determining which data is accessed by each chunk
  - Goal: minimize amount of data transferred while still maintaining correctness

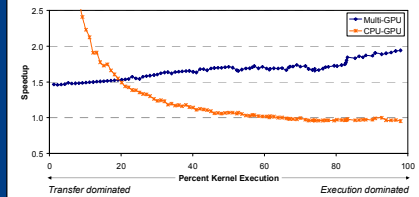
## Overlapping Operations

When a series of different data transfers and kernel executions is executed natively, the operations are serialized. By dividing the operations into independent chunks, data transfer and kernel execution can occur in parallel.



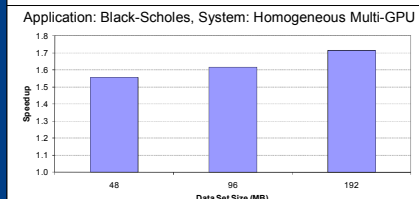
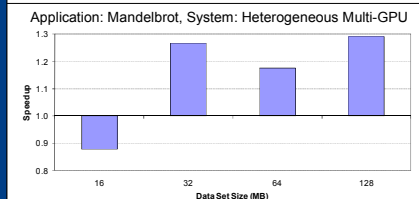
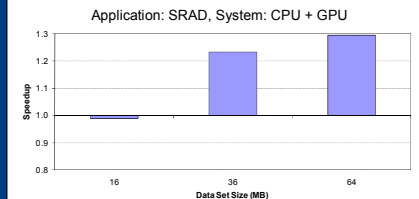
## Execution vs. Transfer Time

Different applications vary widely in their ratios of kernel execution time to data transfer time. Using a synthetic benchmark, we varied this ratio and measured the resulting performance when using an oracle-based load balancer.



**Multi-GPU systems:** Load balancing provides the most benefit to applications dominated by *execution time*.  
**CPU-GPU systems:** Load balancing provides the most benefit to applications dominated by *transfer time*.

## Application Results



## Conclusions

- Heterogeneous systems are becoming ubiquitous, but effectively using all of the available devices is challenging
- Our framework automatically load balances unmodified OpenCL applications across multiple devices by:
  - Breaking a kernel up into multiple chunks
  - Analyzing the source code to determine the data consumed and produced by each chunk
  - Using online profiling to guide scheduling decisions
- Preliminary performance results are promising

## Future Work

- Optimize the framework for:
  - Different hardware and software configurations
  - Different metrics: performance, power, energy
- Add support for multiple kernel calls; need to understand the implicit producer-consumer relationship of successive kernels
- Deployment possibilities:
  - OpenCL layer targeting standalone applications
  - Layer in the AMD Fusion software stack

## Technology Transfer

- Framework being developed in collaboration with AMD
- Weekly AMD-UVA meetings
- Presented work at 2011 AMD Fusion Developer Summit
- Currently collaborating on a paper submission

## 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.