Motivation
• How to reduce the burden of multi-GPU programming?
• How to guarantee functional correctness on multi-GPU?
• How to achieve the optimal performance on multi-GPU?

Summary
• We analyze the burden of multi-GPU programming.
• We are working on static code analysis & architectural supports to achieve functional correctness & optimal performance.

Background
• Higher throughput with multiple GPUs
• Non-optimal performance with multi-GPU
• Burden of multi-GPU programming

Background
• Higher throughput with multiple GPUs
• Non-optimal performance with multi-GPU
• Burden of multi-GPU programming

Problem:
> Keeping data consistency across GPUs degrades performance.
> Data and thread assignment cannot be changed during program execution.
> Lack of atomic operations forces programmers to modify their codes.

Figure 3. Execution time analysis [1]

Figure 1. Speedup versus a single GPU [1]

Figure 2. Speedup versus a single GPU [1]

Not all applications take advantage of multiple GPUs!

Experimental Results
• Existing multi-CPU programming model

Figure 4. Inter-GPU communications through a host

1) Programmers write their program as if there is a single large GPU.
(with many cores & large memory)

2) Static code-analysis for core-data mapping to minimize inter-core communications

“Logical” large GPU

3) Locality-aware run-time thread & memory migration for optimal performance

4) Guarantee atomic operations from cores to be “atomic”

Current Work
We are currently implementing static code analysis (front-end data and thread assignment) and architectural supports (inter-GPU synchronization, atomics, data replication) to guarantee functional correctness and optimal performance based on OpenCL [2].

References