Exploiting Concurrent Kernel Execution on Graphic Processing Units

High Performance Computing and Simulation (HPCS), 2011 International Conference

Author’s:
Lingyuan Wang:
The George Washington University
Miaoqing Huang:
The University of Arkansas
Motivation

• Graphic Processing Units (GPUs) have massive parallel computing capability.
• Achieved tremendous momentum in high-performance computing domain.
• Following Moore’s law as the number of GPU cores keeps growing.
• This has shifted the balance of computation capability beyond parallelism.

How to improve performance??

• *It has become increasingly important to support effective shared access to a single GPU for parallel applications.*
• Traditionally, data parallel kernels can only run sequentially on GPUs
• Concurrent multi-kernel execution is not supported.
Research Significance

• Motivated by this current paper propose **CONTEXT FUNNELING**: which uses a shared GPU context to synchronize the accesses from multiple host threads for each process (same application).

• **Advantages**
  - Removes the overhead of context switching.
  - Extends capability of concurrent kernel execution across all the kernels of an application.
  - Better overall utilization of modern hybrid multicore /GPU systems.
Approach

Traditional GPU Architecture: Fig 1 (a)
- A kernel is a function executed on a GPU device
- CUDA thread blocks CTA are scheduled automatically on Streaming Multiprocessors (SMs).
- Thread blocks also do not migrate to other SMs during their execution
- Single-Program-Multiple-Data (SPMD)

New Approach: Fig 1(b)
- Kernels can execute concurrently,
- Better utilization of GPU resources.

Figure 1. Serial vs. Concurrent Kernel Execution
Approach 1

Context Switching

• Different Application threads will take turn on the GPU cores allowing multiple kernel active at the same time.

• **Limitations**
  - Kind of virtual concurrency
  - Context switching overhead
  - Synchronization overhead.
Approach 2

Manual Context Funneling

• Programmer manually combines the kernels of a multithreaded program into a master host thread

• **Master thread:**
  - Initializes data
  - Communicates with GPU
  - Responsible for streaming data back and forth to the GPU for the entire group of threads.

• **Advantages:**
  - Concurrent kernel execution
  - Efficient sharing and memory usage
  - No context switch overheads
Approach 3

Auto Context Funneling

• Automatic shared context model in essence leads to the virtualization of GPU resources.

• Advantages:
  - Concurrent execution is extended across all host threads.
  - Manual synchronization/funneling: Not required
  - No context switch overheads
Setup

- Two quad-core Intel Xeon X5570 (Nehalem) processors clocked at 2.93GHz with 24GB memory.

- NVIDIA Tesla C2070 GPU is attached via the PCIe x16 bus.

- Tesla C2070 (Fermi) GPU has 14 SMP with 32 superscalar CUDA cores per SM.

- Tesla GPU: Running at 1.15 GHz,

Figure 3. The Hybrid Multicore/GPU Platform
Evaluations

• **SSCA SAR SENSOR PROCESSING**

• Context switching
  - Slowest
  - Decreased occupancy and performance.

• Automatic & manual funneling
  - Same Performance
  - CPU processing is well overlapped with GPU execution for Auto funneling
Evaluations...

Overall speedup

Figure 12. SSCA #3 Strong Scaling Performance
Thank You!

Questions?