

Asynchronous computing using CUDA on a Tesla C2050 GPU

Eduardo Ponce Mojica
Polytechnic University of Puerto Rico
Research Alliance in Math and Science
<https://sites.google.com/site/eduardoponce2011>

Mentor: Jacob Barhen
Oak Ridge National Laboratory
Center for Engineering Science Advanced Research
Collaborators: Charlotte Kotas

MOTIVATION

Synchronous methods for future exascale systems, containing million of components with mean time to failure in the order of seconds to minutes, will unavoidably result in performance degradation. An alternative is concurrently-asynchronous algorithms implemented in massively parallel multithreaded and many-core general-purpose graphics processing units (GPU), which relaxes synchronization points produced by hardware jitter and inherent latency.

RESOURCES

Intel Xeon X5677 CPU

Tesla C2050 GPU



Host



Device

Specifications	Intel Xeon X5677	Tesla C2050
Core count	4	448
Cores clock	3.46 GHz	1.147 GHz
Memory size	24 GB	3 GB
Memory bandwidth	32 GB/s	144 GB/s

- Compute Unified Device Architecture (CUDA)
- Fortran 95 / PVF compiler
- Microsoft Visual Studio
- NVIDIA Compute Visual Profiler

Figure 1. Tesla C2050 / Fermi architecture (14 streaming multiprocessors each with 32 cores)

OBJECTIVES

1. Implement algorithms in sequential, synchronous, and asynchronous paradigms on CPU and GPU
2. Evaluate performance of synchronous and asynchronous schemes
3. Investigate methods for concurrently-asynchronous computing in a CPU / GPU heterogeneous system
4. Derive techniques to optimize asynchronous computing

ASYNCHRONOUS COMPUTING

- Events that occur independently from main program flow
- Reduces idle times between processors, exploit cores
- Requires pinned (page-locked) memory on host
- Uses CUDA streams: sequence of operations in device
- Allows overlap of memory transfers, device executions, and host computations

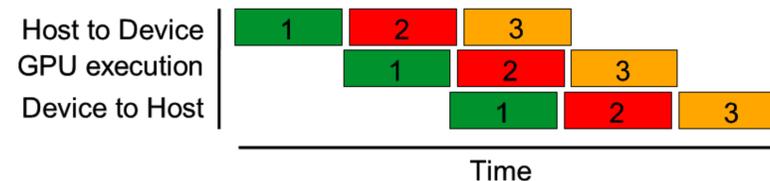


Figure 2. Timeline for concurrent memory transfer between CPU/GPU and kernel execution

METHODOLOGY

- Matrix – matrix multiplication algorithm
- Complexity for square matrices: $O(n^3)$
- Streams use different tiles in B, same tile in A

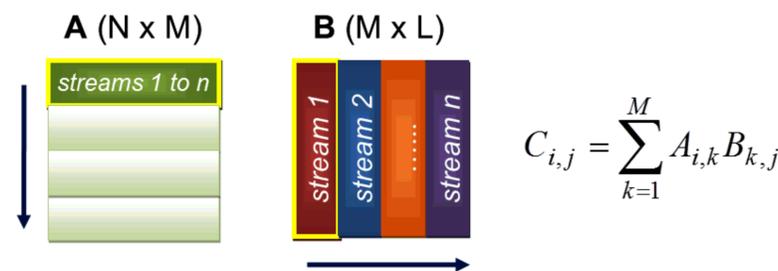


Figure 3. Asynchronous scheme using streams for matrix – matrix multiplication

- Loop begins transferring data for streams, host to device
- Each stream computes a portion of C
- Complete a C tile, transfer GPU to host, start loop

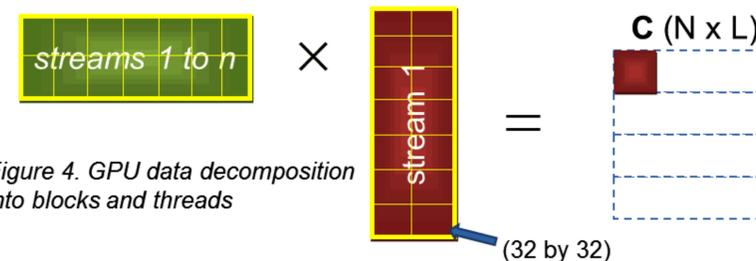


Figure 4. GPU data decomposition into blocks and threads

- Tiles from A and B are subdivided into a grid of blocks
- Blocks are further divided into a grid of 32 x 32 threads

RESULTS AND CONCLUSIONS

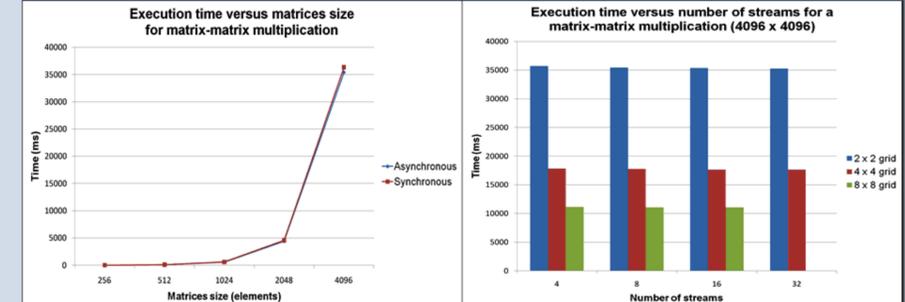


Figure 5. Execution time graphs for synchronous and asynchronous matrix-matrix multiplication schemes using tiles

- Synchronous version execution time (4096): 36,332 ms
- Asynchronous version execution time (4096): 35,423 ms

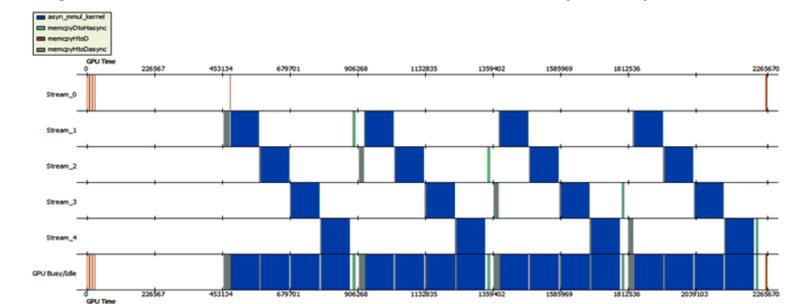


Figure 6. Asynchronous computing pipeline using 4 streams shown in NVIDIA Compute Visual Profiler, GPU idle time is reduced

- CUDA streams usage serves as a first stage to enable concurrently-asynchronous computing

FUTURE RESEARCH

- Benchmark additional parameters, such as GPU occupancy and memory bandwidth
- Implement common algorithms using the asynchronous transfer-execute-transfer pipeline
- Derive conditions for convergence in an asynchronous regime

REFERENCES

1. NVIDIA Developer Zone: <http://developer.nvidia.com/category/zone/cuda-zone>
2. Kirk, D. and Hwu, W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach*. USA: Elsevier, Inc.
3. Sanders, J. and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Michigan, USA: Edwards Brothers.