CBench: Analyzing Compute Performance for Modern NVIDIA and AMD GPUs

Varun Sampath*
University of Pennsylvania

Abstract

General purpose GPU computation is a fast growing field with a variety of applications. For maximum performance, though, mapping high-level parallel algorithms to vendor hardware requires a solid grasp of both the algorithm's computational requirements and the microarchitectural limitations of the GPU. This work aims to explore the performance of high and low arithmetic intensity workloads on the latest NVIDIA and AMD GPU hardware, codenamed Fermi and Barts, respectively. A summed area table generator and a Black-Scholes option pricer were used as benchmarks to analyze performance for compute- and bandwidth-bound algorithms. It was found that the AMD Barts GPU provided a 50% performance boost on the Black-Scholes compute-bound workload, whereas Fermi excelled at the more memory-bound summed area table computation.

CR Categories: B.8.2 [Hardware]: Performance and Reliability—Performance Analysis and Design Aids;

Keywords: CUDA, OpenCL, benchmark, NVIDIA, AMD, Fermi, Barts

Links: DL PDF WEB VIDEO DATA CODE

1 Overview

Using the GPU as a coprocessor is becoming an increasingly common practice for accelerating data-parallel computations. It can be said that the GPU manufacturer NVIDIA has been spearheading such efforts with their CUDA architecture. The release of OpenCL as a platform-agnostic alternative, though, opens the market to other vendors. In this paper, the performance of NVIDIA and AMD hardware along with CUDA and OpenCL software is investigated. Two benchmarks are used to investigate both compute- bound and memory-bound performance. The specific hardware analyzed is NVIDIA's latest high-performance architecture, Fermi, and AMD's latest prosumer architecture, Barts. Section 2 describes the architecture of these two GPU families in comparison to older GPU architectures like the NVIDIA G80. Section 3 discusses related benchmarking work. Section 4 describes the benchmarks used for evaluation. Section 5 provides charts for the benchmarking data collected, and Section 6 is an analysis of that data. Section 7 describes possible future work.

*e-mail: vsampath@seas.upenn.edu

2 GPU Architectures

Both the Fermi and Barts GPU architectures follow the SIMT (single instruction, multiple thread) model of execution. A group of threads/work-items execute the same instruction at the same time on different data. The GPU scheduler tries to schedule as many of these groups as it can to saturate the hardware and hide stalls. The following is a discussion of the Fermi and Barts architectures in the scope of this model.

2.1 Fermi Architecture

2.1.1 Fermi Compute Architecture

Fermi's compute architecture is a significant enhancement from previous NVIDIA GPU generations. At the highest level, a Fermi GPU contains 14 multiprocessors, or compute units. Each of these contains 32 CUDA cores. There are also 16 load/store units in the multiprocessor.

Fermi schedules two warps (a SIMT unit of 32 threads/work-items) concurrently for each multiprocessor. Each warp can either occupy 16 CUDA cores, the 16 load/store units, or the 4 transcendental units. This is in contrast to previous NVIDIA architectures where a multiprocessor only executed one warp at a time [NVI].

2.1.2 Fermi Memory Architecture

Unlike prior generations of NVIDIA GPU architectures, Fermi provides a cache hierarchy for compute applications. Each multiprocessor contains a 64KB block partitioned between L1 cache and shared memory. The split must be 16KB/48KB, but the application developer can choose whether the L1 cache or the shared memory takes the larger portion.

All of the multiprocessors on the GPU share a 768KB L2 cache. All accesses to off-chip memory go through this L2 cache. By default, accesses to off-chip memory also go through the L1 cache. Given the fact that accesses all go through the caches, memory coalescing optimizations are now also a function of the cache. A cache line is 128B, so when a thread misses the cache on a memory read, a full 128B data chunk is pulled from off-chip memory. This implies that restrictions present in the G80 requiring consecutive reads in a half-warp are no longer necessary; all reads within this 128B line will now hit the cache instead of off-chip memory. Memory addresses are also hashed in Fermi, so partition camping is no longer an issue [2010].

Shared memory has also received upgrades in flexibility over previous generation architectures. Shared memory is now split up into 32 banks instead of 16. Also unlike previous generation architectures, if multiple threads access the same 32-bit word, that word can be multicast without any performance penalty. The G80 only supports broadcasting when all threads access the same 32-bit word [2010a].

2.2 Barts Architecture

2.2.1 Barts Compute Architecture

The Barts architecture is a derivative of the Cypress architecture behind the AMD Radeon HD 58xx series of GPUs [2010]. At the highest level, the design hierarchy is similar to Fermi and prior GPUs from both vendors. The GPU consists of compute units, and each compute unit contains many stream cores. Barts contains 14 compute units, with each compute unit containing 16 stream cores [2011].

AMD GPU architectures are different from NVIDIA architectures in that they try to extract instruction-level parallelism (ILP) alongside thread-level parallelism through a very-long-instruction-word (VLIW) architecture. Each stream core in Barts contains 5 scalar processing units. 4 of the 5 units (labeled x, y, z, and w) can perform one scalar floating-point operation per clock, while the fifth unit (t) can additionally perform transcendental operations.

The basic SIMT scheduling block is handled at the stream core level, though. This is called a wavefront in AMD terminology, and is (nearly) analogous to the NVIDIA warp. A wavefront consists of 64 work-items and one instruction clause (explained later). Since there are 16 stream cores in a compute unit, four work-items are executed on each stream core. One instruction is pipelined from each of the four work-items in the stream core, so that any memory access latency can be hidden by the three other execution cycles.

Scheduling of wavefronts is done by the dispatch processor for all compute units. The instruction stream for the kernel is broken up into a series of clauses, depending on the operation. For instance, there are ALU clauses for computation and fetch clauses for memory access. The wavefront scheduled changes at every clause change. Each instruction in a clause is a 5-wide VLIW bundle [2011].

2.2.2 Barts Memory Architecture

The Barts memory hierarchy is similar to older NVIDIA GPUs since it must maintain OpenCL compliance. The shared memory (called local data store, or LDS) has 32 32-bit wide banks that can succumb to the same bank conflict problems as NVIDIA hardware. An additional problem for AMD hardware, though, is that each stream core can make only 2 LDS accesses per cycle, whereas there are 5 ALUs that need data. AMD recommends handling this inadequacy by using more of the register file, since it is much larger and provides much higher bandwidth.

Accesses to global memory can suffer from either channel or bank conflicts. There are 8 memory controllers with 256-byte channels that access banks of even larger width. Accesses should also be coalesced, although the performance impact of weakly-coalesced writes does not impact performance terribly because of communication with the memory controller.

Barts and Cypress both have read-only L1 and L2 caches. The latest SDK lets read-only OpenCL buffers as well as OpenCL images (i.e. textures) use the caches. The L1 is 8KB and is present for each compute unit, while the L2 is built into each memory controller. With eight memory controllers, there is 512KB of L2 cache available [2011].

3 Related Work

Danalis et al. developed “The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite.” They analyzed the performance of NVIDIA GT200 and G80 class GPUs, ATI Evergreen class GPUs, and Intel and AMD multicore CPUs for a variety of compute- and bandwidth-constrained kernels using both CUDA and OpenCL. They found the Radeon HD 5870 to be a high performer, and saw OpenCL performance trail CUDA performance on NVIDIA hardware [2010].

Du et al. evaluated and tuned OpenCL kernels for level 3 BLAS routines. They benchmarked their results on the Fermi Tesla C2050 and the Radeon HD 5870. They found the ATI hardware to be much faster at their matrix multiplication task, and were able to nearly match an OpenCL implementation in speed with CUDA on Fermi [2010].

4 Approach

Two benchmarks applications were developed for both CUDA and OpenCL. The CUDA version was benchmarked on the NVIDIA Fermi GPU, while the OpenCL version was benchmarked on both the NVIDIA Fermi and the AMD Barts GPU. The specific Fermi GPU used was the Tesla C2070. The specific Barts GPU used was the Radeon HD 6870. The benchmark programs were compiled using Visual Studio 2008 and the Windows High-Performance Counter was used for timing by the CPU. NVIDIA binaries were compiled for compute model 2.0. The following subsections describe the two kernels being benchmarked.

4.1 Summed Area Tables

A benchmark program was implemented to calculate summed area tables (SAT) of float4 matrices on the GPU using a scan kernel and a transpose kernel. The scan kernel performed an inclusive scan of all rows of the input matrix in parallel. The scan algorithm used was the naive parallel scan, which requires $O(nlogn)$ additions [2007].

The scan kernel makes use of shared/local memory for coalescing reads and writes. The transpose kernel is a simple naive transpose, though, which does not coalesce writes.

To calculate a summed area table, the rows of the matrix were first scanned. The matrix was then transposed and the rows were scanned again. The matrix was transposed one more time to obtain the final result. The transposes were performed to avoid non-coalesced scans across columns of the input matrix and to provide additional performance data. Both inclusive scan and transpose operations are very bandwidth-constrained, so this program tests low arithmetic intensity performance.

4.2 Black-Scholes

The Black-Scholes model is used for the pricing of put and call European-style options. The closed-form solution for the model can be used to price these options on the GPU [2007]. Each pricing requires many floating-point calculations, and no pricings are dependent on another. This results in an embarrassingly parallel benchmark designed to test high arithmetic intensity performance. The CUDA and OpenCL kernels for this program were provided by the NVIDIA GPU Computing SDK [2007].

5 Results

5.1 Summed Area Tables

Figure 1 shows Fermi and Barts results for the SAT benchmark. Figure 2 shows the performance delta between the CUDA and OpenCL SAT kernels executing on the Fermi GPU.

5.2 Black-Scholes

Figure 3 shows Fermi and Barts results for the Black-Scholes benchmark. Figure 4 shows the performance delta between the CUDA and OpenCL kernels executing on the Fermi GPU.

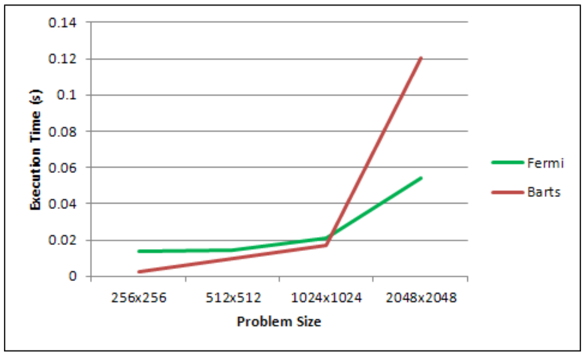

Figure 1: *SAT OpenCL Performance with work-group size of 256*

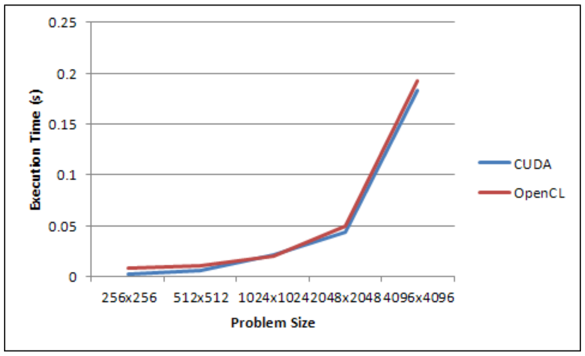

Figure 2: *SAT OpenCL and CUDA Performance with work-group size of 256*

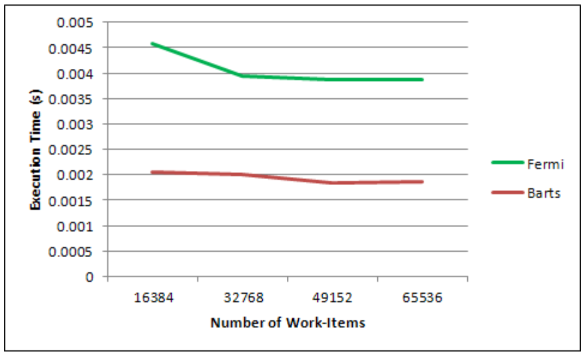

Figure 3: *Black-Scholes OpenCL Performance with work-group size of 256 and processing of 8 million options*

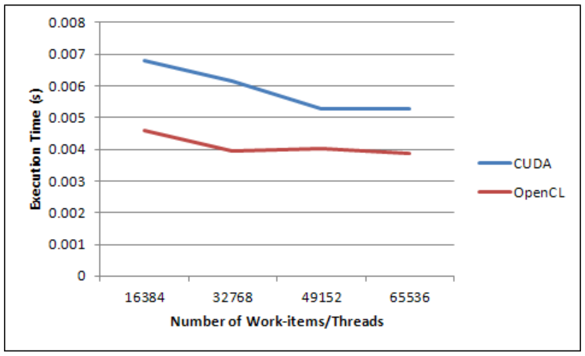

Figure 4: *Black-Scholes OpenCL and CUDA Performance with work-group size of 256 and processing of 8 million options*

6 Analysis

6.1 Black-Scholes Performance

The results of the comparison between Barts and Fermi for this program validate that this benchmark is compute-bound. The Tesla C2070 has higher theoretical bandwidth to off-chip memory (144 GB/s [2010b] vs 134 GB/s [2011]) and also has larger L1 and L2 caches. All reads and writes in this kernel are also coalesced. The performance of the Fermi card is still around 50% slower than the Barts card. We can analyze this gap by looking at the generated assembly and the compute capabilities of each card.

6.1.1 Black-Scholes Barts Performance

AMD provides the APP KernelAnalyzer to generate the AMD Intermediate Language (IL) representation of the Black-Scholes kernel. The output shows that there are 92 ALU VLIW5 instructions, compared to only 6 fetch and 2 write instructions. The kernel uses 12 registers, and since each compute unit has 16,384 available, Barts is bottlenecked to 1365 threads per compute unit, or 20 wavefronts (5 work-groups). Since there are so few memory access instructions (with the reads being cached too) , the latency of the memory accesses can be effectively hidden by these wavefronts.

Barts has a theoretical compute performance of 2016 GFLOPS [2011]. To harness this compute power, though, the 5 scalar processing units of each stream core need to be saturated with independent floating point instructions. The AMD IL shows that for the majority of ALU VLIW5 instructions, 4 to 5 of the scalar processing units are executing in parallel. If we conservatively say that compute unit ALU utilization is at 70%, then the extrapolated performance of 1411 GFLOPS is higher than Fermi's peak theoretical single precision performance [2010b]. The performance of this kernel shows that AMD hardware can perform extremely well with high numbers of independent single precision operations.

6.1.2 Black-Scholes Fermi Performance

We can use the PTX output of the NVIDIA OpenCL compiler and NVCC to understand the performance of the OpenCL and CUDA implementations. The PTX output for the Black-Scholes OpenCL kernel shows that there are two loads and 3 stores per loop iteration versus many more scalar floating point operations. Assembling via PTXAS shows that the kernel uses 30 registers. Since 32,768 registers are available per multiprocessor on Fermi, 4 work-groups can occupy a multiprocessor, or 32 warps. With the high amount of compute instructions, memory latency can also be hidden in this case.

Ideally, because the application is compute-bound, we should see near the peak compute throughput during execution. There are some reasons why we do not see these results, though. One is the use of special functions in the kernel, such as SQRT and LOG. There are only 4 units in the Fermi multiprocessor for handling these instructions (instead of 16 CUDA cores), so they take 4 times as long to execute. The advantage of the Fermi architecture is the dual warp scheduling approach that does not let the CUDA cores sit idly while the special functions are executing. In any case, the drop from peak theoretical performance for this reason plus the overhead of other instructions and scheduling can justify the Fermi performance recorded. The factor of two difference in theoretical performance between Fermi and Barts also helps in justifying the real-world factor of two difference presented here.

The comparison between CUDA and OpenCL for this application gives a surprising result, considering the several more years of maturity that CUDA has. The PTX output of NVCC and the OpenCL LLVM compiler is substantially different, so it is difficult to substantiate the performance difference with assembly analysis. Even more surprising is that there are far more instructions generated by the OpenCL compiler than NVCC. One possible reason for the performance discrepancy is the ordering of instructions, especially loads for prefetching purposes. The NVIDIA Compute Visual Profiler also confirms these results that the OpenCL implementation is faster. Understanding in more detail why may be a subject for future work.

6.2 Summed Area Table Performance

This benchmark has several variables and components, such as shared/local memory usage in the scan kernel, the uncoalesced writes in the transpose kernel, and the caching performed by both GPUs. The following sections will attempt to justify the execution times witnessed.

6.2.1 SAT Barts Performance

The Barts computations unfortunately did not scale to 4096x4096 due to runtime execution errors. Despite the limited data, we can still attempt to draw some conclusions. With two scan and two transpose calls, the 2048x2048 problem yields a total bandwidth of 26.7 GB/s. This is far lower than the peak theoretical bandwidth for a variety of reasons. For the scan kernel, reads offset by a stride are subject to channel or bank conflicts, especially since the problem sizes are all multiples of the channel width. Accesses to local/shared memory are also subject to bank conflicts because of the doubled indices used. Lastly, writes in the transpose kernel are uncoalesced. We have also ignored the effects of caching because of the small size; the 8KB L1 cache cannot supply enough data for reads of a 2048x2048 float4 matrix. The combination of these bottlenecks and the $O(nlogn)$ scaling of scan can explain the execution times found in testing.

6.3 SAT Fermi Performance

Using the same 2048x2048 problem, Fermi obtains a total bandwidth of 59.5 GB/s. There are a few reasons for this large performance delta between Fermi and Barts in this example. One is Fermi's hashing of memory addresses through the use of virtual memory. This hashing eliminates off-chip memory bank conflicts, which is a serious concern for the strided access patterns in the naive scan.

Another improvement over Barts is the distribution of shared/local memory. While they both have 32 32-bit wide banks, Fermi only needs parallelism amongst 32 threads versus Barts' 64 threads. In the scan example, where shared/local memory is indexed by 2 times the thread index, Fermi has 50% less bank conflicts.

A feature that may not aid performance is Fermi's cache hierarchy. Due to the use of different input and output buffers, the L1 and L2 caches should always miss. Data from the NVIDIA Visual Profiler confirms this.

The comparison between CUDA and OpenCL for this application shows CUDA being slightly faster than OpenCL. Understanding the differences between the PTX generated requires more knowledge of the instruction set. From this and the Black-Scholes CUDA and OpenCL comparison it shows that the performance between implementations varies substantially based on what compiler optimizations are used, so a generalized statement about CUDA versus OpenCL performance cannot be made.

7 Future Work

There are many avenues for future work. One is upgrading the SAT benchmark to use more efficient algorithms like those provided in vendor libraries. This will give a more real-world indication of the performance of the system.

Another aspect to research is NVIDIA PTX and AMD IL optimizations. By understanding the assembly generated by CUDA and OpenCL compilers, much more information can be extracted about why certain performance deltas exist.

Finally, more benchmarks that are not at the extremes of bandwidth or compute constraints can be implemented to give developers more relevant information on the performance of their own systems.

Acknowledgements

Special thanks to Aleksandar Dimitrijevic for providing the AMD Radeon HD 6870 for testing, and to Patrick Cozzi and Jon McCaffrey for their advice and help.

References

ADVANCED MICRO DEVICES, INC. 2011. *AMD Accelerated Parallel Processing OpenCL*, Apr.

DANALIS, A., MARIN, G., MCCURDY, C., MEREDITH, J. S., ROTH, P. C., SPAFFORD, K., TIPPARAJU, V., AND VETTER, J. S. 2010. The scalable heterogeneous computing (shoc) benchmark suite. In *Architectural Support for Programming Languages and Operating Systems*, 63–74.

DU, P., WEBER, R., LUSZCZEK, P., TOMOV, S., PETERSON, G., AND DONGARRA, J. 2010. From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming. Tech. rep., Department of Computer Science, UTK, Knoxville Tennessee, Sept.

MARK HARRIS AND SHUBHABRATA SENGUPTA AND JOHN D. OWENS. 2007. *Parallel Prefix Sum (Scan) with CUDA*, vol. 3 of *GPU Gems*.

NVIDIA CORPORATION. *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*.

NVIDIA CORPORATION. 2010. *NVIDIA CUDA C Programming Guide*, Oct.

NVIDIA CORPORATION. 2010. *NVIDIA Tesla Datasheet*, July.

PODLOZHNYUK, V. 2007. Black-Scholes Option Pricing. Tech. rep., June.

RUETSCH, G., AND MICIKEVICIUS, P. 2010. Optimizing Matrix Transpose in CUDA. Tech. rep., June.

SMITH, R. 2010. AMD's Radeon HD 6870 & 6850: Renewing Competition in the Mid-Range Market. *AnandTech* (Dec.).