Architecture Comparisons between Nvidia and ATI GPUs: Computation Parallelism and Data Communications

Ying Zhang[1], Lu Peng[1], Bin Li[2]
[1] Department of Electrical and Computer Engineering
[2] Department of Experimental Statistics
Louisiana State University, Baton Rouge, LA, USA
{yzhan29, lpeng, bli}@lsu.edu

Jih-Kwon Peir[3], Jianmin Chen[3]
[3] Department of Computer & Information Science and Engineering
University of Florida
Gainesville, Florida, USA
{peir, jichen}@cise.ufl.edu

***Abstract* — In recent years, modern graphics processing units have been widely adopted in high performance computing areas to solve large scale computation problems. The leading GPU manufacturers Nvidia and ATI have introduced series of products to the market. While sharing many similar design concepts, GPUs from these two manufacturers differ in several aspects on processor cores and the memory subsystem. In this paper, we conduct a comprehensive study to characterize the architectural differences between Nvidia's Fermi and ATI's Cypress and demonstrate their impact on performance. Our results indicate that these two products have diverse advantages that are reflected in their performance for different sets of applications. In addition, we also compare the energy efficiencies of these two platforms since power/energy consumption is a major concern in the high performance computing.**

Keywords - GPU; clustering; performance; energy efficiency

I. INTRODUCTION

With the emergence of extreme scale computing, modern graphics processing units (GPUs) have been widely used to build powerful supercomputers and data centers. With large number of processing cores and high-performance memory subsystem, modern GPU is a perfect candidate to facilitate high performance computing (HPC). As the leading manufacturers in the GPU industry, Nvidia and ATI have introduced series of products that are currently used in several preeminent supercomputers. For example, in the Top500 list released in Jun. 2011, the world's second fastest supercomputer Tianhe-1A installed in China employs 7168 Nvidia Tesla M2050 general purpose GPUs [11]. LOEWE-CSC, which is located in Germany and ranked at 22nd in the Top500 list [11], includes 768 ATI Radeon HD 5870 GPUs for parallel computations.

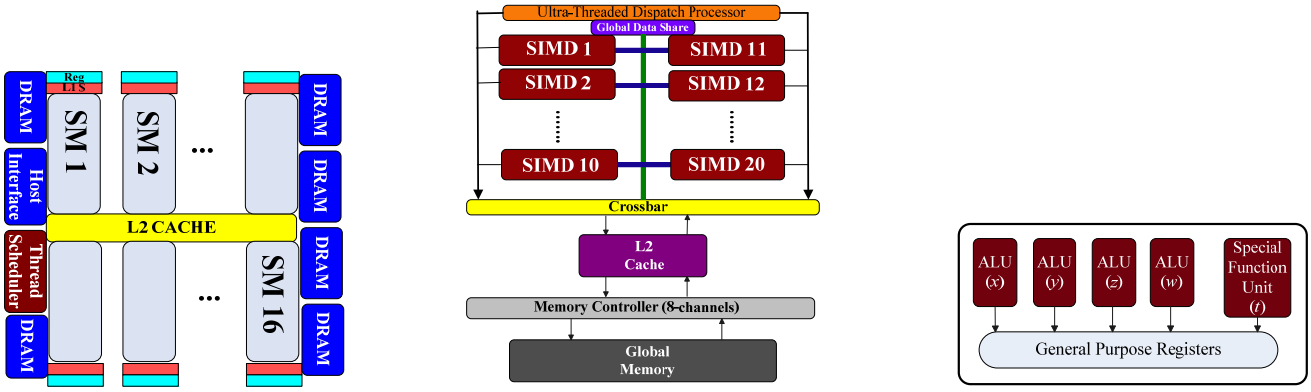
Although typical Nvidia and ATI GPUs are close to each other on several design specifications; they deviate in many architecture aspects from processor cores to the memory hierarchy. In this paper, we measure and compare the performance and power consumption of two recently released GPUs: Nvidia GeForce GTX 580 (Fermi) [7] and ATI Radeon HD 5870 (Cypress) [4]. By running a set of representative general-purpose GPU (GPGPU) programs, we demonstrate the key design difference between the two platforms and illustrate their impact on the performance.

The first architectural deviation between the target GPUs is that the ATI product adopts very long instruction word (VLIW) processors to carry out computations in a vector-like fashion. Typically, in an *n*-way VLIW processor, up to *n* data-independent instructions can be assigned to the slots and be executed simultaneously. Obviously, if the *n* slots can always be filled with valid instructions, the VLIW architecture will outperform the traditional design. Unfortunately, this is not the case in practice because the compiler may fail to find sufficient independent instructions to generate compact VLIW instructions. On average, if *m* out of *n* slots are filled during an execution, we say the achieved *packing ratio* is *m*/*n*. The actual performance of a program running on a VLIW processor largely depends on the packing ratio. On the other hand, the Nvidia GPU uses multi-threading execution to execute code in a Single-Instruction-Multiple-Thread (SIMT) fashion and explores thread-level parallelism to achieve high performance.

The second difference between two GPUs exists in the memory subsystem. Both GPUs involve a hierarchical organization consisting of the L1 cache, L2 cache, and the global memory. On the GTX 580 GPU, the L1 cache is configurable to different sizes and can be disabled by setting a compiler flag. The L1 cache on the HD 5870 is less flexible and can only be used to cache image objects and constants. The L2 caches on both GPUs are shared among all hardware multiprocessor units. All global memory accesses go through the L2 in GTX 580, while only image objects and constants use the L2 in HD 5870. Given these differences, we will investigate and compare the memory system of the target GPUs.

Thirdly, power consumption and energy efficiency is becoming a major concern in high performance computing areas. Due to the large amount of transistors integrated on chip, a modern GPU is likely to consume more power than a typical CPU. The resultant high power consumption tends to generate substantial heat and increase the cost on the system cooling, thus mitigating the benefits gained from the performance boost. Both Nvidia and ATI are well aware of this issue and have introduced effective techniques to trim the power budget of their products. For instance, the PowerPlay technology [1] is implemented on ATI Radeon graphics cards, which significantly drops the GPU idle power. Similarly, Nvidia use the PowerMizer technique [8] to reduce the power consumption of its mobile GPUs. In this paper, we measure and compare energy efficiencies of these two GPUs for further assessment.

One critical task in comparing diverse architectures is to select a common set of benchmarks. Currently, the programming languages used to develop applications for Nvidia and ATI GPUs are different. The Compute Unified Device Architecture (CUDA) language is majorly used by Nvidia GPU developers, whereas the ATI community has introduced the Accelerated Parallel Processing technology to encourage engineers to focus on the OpenCL standard. To select a common set of workloads, we employ a statistical clustering tech-



(a) GTX 580 (b) Radeon HD 5870 (c) VLIW processor in HD 5870

Figure 1. Architecture of target GPUs

nique to select a group of representative GPU applications from the Nvidia and ATI developer's SDK, and then conduct our studies with the chosen programs. According to the experiment results, we can make several interesting observations:

- For programs that involve significant data dependency and are difficult to generate compact VLIW bundles, the GTX 580 (Fermi) is more preferable from the standpoint of high performance. In contrast, the ATI Radeon HD 5870 (Cypress) is a better option to run programs where sufficient instructions can be found to compact the VLIW slots.
- The GTX 580 GPU outperforms its competitor on double precision computations. The Fermi architecture is delicately optimized to deliver high performance in double precision, making it more suitable in solving problems with high precision requirement.
- Memory transfer speed between the CPU and GPU is another important performance metrics which impact the kernel initiation and completion. Our results show that Nvidia generally has higher transfer speed. Besides the lower frequency of the device memory on the ATI HD 5870 GPU [4][7], another reason is that the memory copy in CUDA has smaller launch overhead compared to the ATI OpenCL counterpart.
- According to our experiments, the ATI Radeon HD 5870 consumes less power in comparison with the GTX 580. If a problem can be solved on these two GPUs in similar time, the ATI GPU will be more energy efficient.

The remainder of this paper is organized as follows. In section II, we introduce the architecture of these two graphics processing units. We describe our experiment methodology including the statistical clustering technique in section III. After that, we analyze and compare the different characteristics of the target GPUs and their impact on performance and energy efficiency by testing the selected benchmarks in section IV. We review the related work in section V and finally draw our conclusion in section VI.

II. BACKGROUND

In this section, we describe the architecture organizations of Nvidia GTX 580 and ATI Radeon HD 5870. We also briefly introduce the programming languages that are used on these GPUs. A summary of manufacturing parameters of these two GPUs along with a description of the host system is listed in Table I [4][7].

TABLE I. SYSTEM INFORMATION

GPU information		
Parameter	GTX 580	Radeon HD 5870
Technology	40nm	40nm
#Transistors	3.0 billion	2.15 billion
Processor clock	1544 MHz	850 MHz
#Execution units	512	1600
GDDR5 clock rate	2004 MHZ	1200 MHz
GDDR5 bandwidth	192.4 GB/s	153.6 GB/s
Host system information		
CPU	Intel Xeon E5530	AMD Opteron 6172
Main memory type	PC3-8500	PC3-8500
Memory size	6GB	6GB

A. Fermi Architecture

Fermi is the latest generation of CUDA-capable GPU architecture introduced by Nvidia [13]. Derived from prior families such as G80 and GT200, the Fermi architecture has been improved to satisfy the requirements of large scale computing problems. The GeForce GTX 580 used in this study is a Fermi-generation GPU [7]. Figure 1(a) illustrates its architectural organization [13]. As can be seen, the major component of this device is an array of streaming multiprocessors (SMs), each of which contains 32 CUDA cores. There are 16 SMs on the chip with a total of 512 cores integrated in the GPU. Within a CUDA core, there exist a fully pipelined integer ALU and a floating point unit (FPU). In addition to these regular processor cores, each SM is also equipped with four special function units (SFU) which are capable of executing transcendental operations such as sine, cosine, and square root.

The design of the fast on-chip memory is an important feature on the Fermi GPU. In specific, this memory region is now configurable to be either 16KB/48KB L1 cache/shared memory or vice versa. Such a flexible design provides performance improvement opportunities to programs with different resource requirement. Another subtle design is that the L1 cache can be disabled by setting the corresponding compiler flag. By doing that, all global memory requests will be bypassed to the 768KB L2 cache shared by all SMs directly. Note that in the following sections, we may use the term Fermi, GTX 580, and Nvidia GPU interchangeably.

The CUDA programming language is usually used to develop programs on Nvidia GPUs. A CUDA application

launches at least one *kernel* running on the GPU. To improve the parallelism, a typical kernel includes several *blocks*, each of which is further composed of many *threads*. During a kernel execution, multiple *blocks* are assigned to an SM according to the resource requirement. Each *block* has multiple threads and 32 threads form a *warp*. A warp is the smallest scheduling unit to be run on the hardware function units in an SIMT fashion for optimal performance.

B. Cypress Architecture

Cypress is the codename of the ATI Radeon HD 5800 series GPU [12]. The architecture organization is shown in Figure 1(b). In general, it is composed of 20 Single-Instruction-Multiple-Data (SIMD) computation engines and the underlying memory hierarchy. Inside an SIMD engine, there are 16 thread processors (TP) and 32KB local data share. Basically, an SIMD engine is similar to a stream multiprocessor (SM) on an Nvidia GPU while the local data share is equivalent to the shared memory on an SM. Each SIMD also includes a texture unit with 8KB L1 cache.

Unlike the CUDA cores in an SM, a thread processor within an SIMD is a five-way VLIW processor. We demonstrate this in the Figure 1(c) by visualizing the internal architecture of a thread processor. As shown in the figure, each TP includes five processing elements, four of which are ALUs while the remaining one is a special function unit. In each cycle, data-independent operations assigned to these processing elements constitute a VLIW bundle and are simultaneously executed. In this paper, we use the term HD 5870, Cypress GPU, and ATI GPU to represent the same device.

For software developers working on ATI GPUs, the Open Computing Language (OpenCL) is the most popular programming tool. OpenCL is similar to CUDA in many design concepts. For example, an OpenCL kernel may include a large amount of *work-groups* which can be decomposed of many *work-items*. This relation is comparable to that between CUDA blocks and threads. In addition, 64 work-items constitute a *wavefront*, similar to a warp in CUDA.

III. METHODOLOGY

A. Experimental Setup

Our studies are conducted on two separate computers, equipped with an Nvidia Geforce GTX 580 and an ATI Radeon HD 5870 GPU respectively. The CUDA toolkit version 3.2 [5] is installed on the Nvidia system while the ATI Stream SDK version 2.1 [3] is used on the ATI computer. Both development kits provide visual profilers [1][5] for the performance analysis.

For power analysis, the power consumption of a GPU can be decoupled into the idle power P_{i_gpu} and the runtime power P_{r_gpu}. To estimate the GPU idle power, we first use a YOKOGAWA WT210 Digital Power Meter to measure the overall system power consumption P_{idle_sys} when the GPU is added on. We then record the power $P_{idle\_sys\_ng}$ by removing the GPU from the system. No application is running during these two measurements; therefore, the difference between them (i.e., $P_{idle\_sys} - P_{idle\_sys\_ng}$) denotes the GPU idle power. When the GPU is executing a CUDA or OpenCL kernel, we measure the system power P_{run_sys} and calculate the GPU runtime power as $P_{run\_sys} - P_{idle\_sys}$. By summing up P_{i_gpu} and P_{r_gpu}, we obtain the power consumption of the target GPU under stress. Note that P_{i_gpu} is a constant while P_{r_gpu} is varying across different measurements. For the sake of high accuracy, we measure the power consumption of each program multiple times and use their average for the analysis.

B. Statistical Clustering

As described in section I, modern GPUs have been delicately designed to better execute large scale computing programs from different domains. Therefore, we choose the general purpose applications from SDKs to carry out our investigation. In total, the Nvidia application suite contains 53 such applications while the ATI set including 32 different benchmarks. Considering that both SDKs include tens of programs, it will be fairly time consuming to understand and study each of the problems in detail. Previous studies show that it is effective to use a small set of applications to represent the entire benchmark suite, in order to investigate the underlying CPU hardware [31]. We believe that this approach can be also applied to the GPU study. In this work, we employ a statistical clustering technique to choose the most representative programs from the SDKs.

Cluster analysis is often used to group or segment a collection of objects into subsets or "clusters", so that the ones assigned to the same cluster tend to be closer to each other than those in different clusters. Most of the proposed clustering algorithms are mainly heuristically motivated (e.g., *k*-means), while the issue of determining the "optimal" number of clusters and choosing a "good" clustering algorithm are not yet rigorously solved [20]. Clustering algorithms based on probability models offer an alternative to heuristic-based algorithms. Namely, the model-based approach assumes that the data are generated by a finite mixture of underlying probability distribution such as multivariate normal distributions. Studies have shown that the finite normal mixture model is a powerful tool for many clustering applications [15][16][28].

In this study, we assume that the data are generated from a finite normal mixture model and apply the model-based clustering. In order to select the optimal number of clusters, we compute the Bayesian Information Criterion (BIC) [34] given the maximized log-likelihood for a model. The BIC is the value of the maximized log-likelihood plus a penalty for the number of parameters in the model, allowing comparison of models with differing parameterizations and/or differing numbers of clusters. In general, the larger the value of the BIC, the stronger the evidence for the model and number of clusters is [21]. This means that the clustering which yields the largest BIC value is the optimal. In this paper, model-based clustering is run by using the *mclust*, which is contributed by Fraley and Raftery [21].

C. Procedure Overview

Our approach consists of three steps. First, we use the visual profilers to collect the execution behaviors of all general purpose applications included in the SDKs. Some applications provide more than one kernel implementations with different optimization degrees. For example, the *matrix multiplication* benchmark from the ATI SDK contains three versions: computation without using the local data share, using the local data share to store data from one input matrix, and using the

TABLE II. CLUSTERING RESULT FOR NVIDIA BENCHMARKS

	Benchmarks
Cluster 1	clock, convolutionSeparable, DwtHarr, **FastWalshTransform**, ptxjit, ScalarProd, SimpleAtomicsIntrincs, SimpleZeroCopy, Transpose_coarsegrain, Transpose_coalesed, Transpose_diagonal, Transpose_finegrain, Transpose_optimized, Transpose_sharedmemory, Transpose_simplecopy, VectorAdd, BinomialOption, QuasiRandomGenerator, Scan, Reduction_k0, Reduction_k1, Reduction_k2, Reduction_k3
Cluster 2	ConjugateGradient, FDTD3D, **Histogram**, SimpleCUFFT, RadixSort
Cluster 3	ConvolutionFFT2D_builtin, ConvolutionFFT2D_custom, ConvolutionFFT2d_optimized, dxtc, SortingNetworks, Transpose_naive, **BlackScholes**, Reduction_k4, Reduction_k5, Reduction_k6
Cluster 4	EstimatePiInlineP, EstimatePiInlineQ, EstimatePiP, EstimatePiQ, **MatrixMul_2_smem**, MatrixMulDrv, MatrixDylinkJIT, MonteCarlo, SimpleVoteIntrincs, SingleAsianOptionP, threadFenceReduction, dct8×8, MersenneTwister
Cluster 5	**EigenValue**, Mergesort

TABLE III. CLUSTERING RESULT FOR ATI BENCHMARKS

	Benchmarks
Cluster 1	AESEncryptDecrypt, **BlackScholes**, DwtHarr, MonteCarloAsian, MersenneTwister, LDSBandwidth,
Cluster 2	HistogramAtomics, MatrixMulImage, **MatrixMul_no_smem**, ConstantBandwidth, ImageBandwidth
Cluster 3	**BinomialOption**
Cluster 4	BitonicSort, **FastWalshTransform**
Cluster 5	BinarySearch, DCT, FFT, **Histogram**, MatrixTranspose, PrefixSum, Reduction, SimpleConvolution, QuasiRandomSequence, ScanLargeArray
Cluster 6	**EigenValue**
Cluster 7	**FloydWarshall**
Cluster 8	MatrixMul_1_smem, **MatrixMul_2_smem**
Cluster 9	**MonteCarloAsianDP**, GlobalMemoryBandwidth
Cluster 10	**RadixSort**

local data share to store data from both input matrices. Each of the three versions can be invoked individually. In this work, we treat these kernels as different programs since they have distinct execution behaviors on the GPU. Another issue is that several benchmarks from two SDKs correspond to the same application scenario. For such programs, we explore the code and ensure that the Nvidia and ATI implementations have identical input and output size. Second, by employing the BIC based statistical clustering method, we classify all applications into a number of categories according to their performance profiles. We then choose a program from each cluster for our analysis. For fair comparisons, each selected application based on clustering in one SDK is used to find an "equivalent" application in the other SDK. We made the best effort including minor code modifications to ensure the selected kernels to perform the same tasks when running on both systems. Third, we use the selected set of applications to compare the architectural differences and energy efficiency of two GPUs.

IV. RESULT ANALYSIS

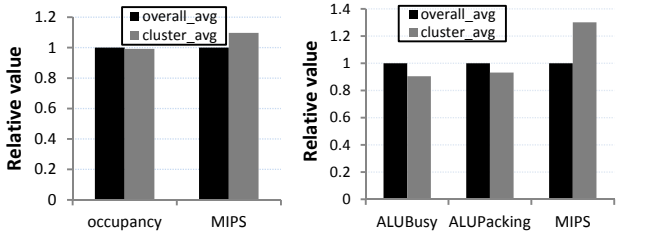
A. Benchmark Clustering

The clustering results for Nvidia and ATI benchmark suites are respectively listed in Table II and Table III. As can be seen, the optimal number of categories for Nvidia applications is five. The ATI programs have a larger number of clusters, although this set has even fewer applications than the Nvidia suite. Actually, our clustering analysis shows that the global optimal cluster number for ATI programs is 31, while 10 is a suboptimal choice. Considering that the goal of this study is to investigate and compare the architectural features of two GPUs using a manageable set of representative applications, we decide to classify all ATI programs into 10 groups from a suboptimal classification.

The common set of applications used for this work should cover all clusters from both benchmark suites. To achieve this goal, we select 10 programs including *BinomialOptions*, *BlackScholes*, *EigenValue*, *FastWalshTransform*, *FloydWarshall*, *Histogram*, *Matrixmul_2_smem*, *Matrixmul_no_smem*, *MontecarloDP*, and *RadixSort*. By doing this, all the 5 clusters in the Nvidia SDK and the 10 clusters in the ATI SDK application set are fully covered. Note that the Nvidia benchmark suite does not provide CUDA implementations for applications including *FloydWarshall*, *Matrixmul_no_smem*, and *MontecarloDP*; so we need to implement them manually. A brief description of these 10 applications is given in Table IV.

TABLE IV. OVERLAPPED APPLICATIONS

Workload	Description
BinomialOption	Binomial option pricing for European options
BlackScholes	Option pricing with the Black-Scholes model
EigenValue	Eigenvalue calculation of a tridiagonal symmetric matrix
FastWalsh	Hadamard ordered Fast Walsh Transform
FloydWarshall	Shortest path searching in a graph
Histogram	Calculation of pixel intensities distribution of an image
Matmul_2_smem	Matrix multiplication, using the shared memory to store data from both input matrices
Matmul_no_smem	Matrix multiplication, without using shared memory
MonteCarloDP	Monte Carlo simulation for Asian Option, using double precision
RadixSort	Radix-based sorting



(a) Nvidia (b) ATI

Figure 2. Validation results of benchmark clustering

For each benchmark suite, we validate the effectiveness of clustering by comparing the average of selected programs and that of all applications for important metrics. The metrics used for validations on two GPUs are slightly different. For the execution rate, we employ the widely used *millions of instructions per second* (MIPS) as the criteria for each set individually. For the Nvidia applications, we also compare the SM *occupancy*, which is defined as the ratio of actual resident warps on an SM to the maximal allowable warps on a streaming multiprocessor. This metric can reflect the overall parallelism of an execution and is fairly important in the general purpose GPU computing. For the ATI programs, we choose the *ALUBusy* and *ALUPacking* as additional validation metrics. This is because that in the VLIW architecture, the packing ratio is one of the dominant factors that determine the throughput. Moreover, the *ALUBusy* indicates the average ALU activity during an execution, which is also critical to the overall performance.

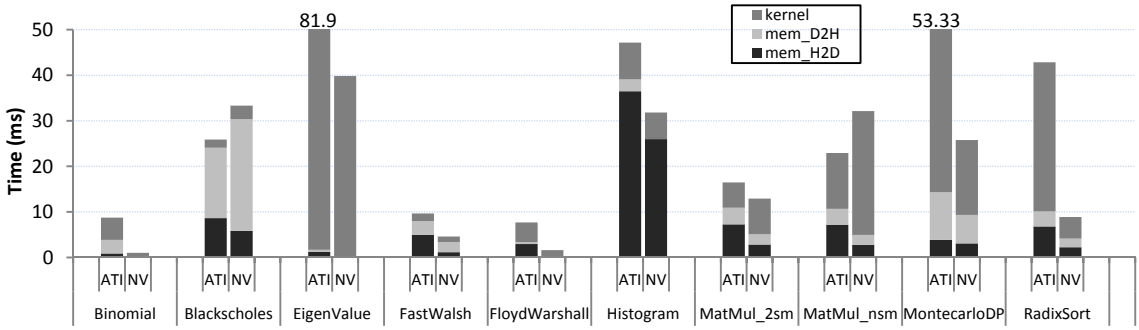
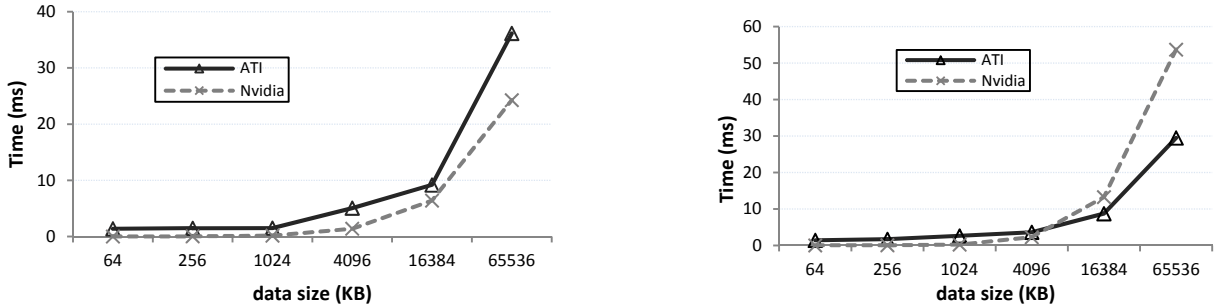


Figure 3. Execution time breakdown of selected applications



(a) Host to device transfer

(b) Device to host transfer

Figure 4. Memory transfer performance comparison

The validation results are demonstrated in Figure 2. As observed, the average *occupancy* and *MIPS* for all Nvidia applications can be well approximated by the selected programs. For the ATI programs set, both *ALUBusy* and *ALUPacking* can be estimated reasonably well; however, we notice that the metric *MIPS* leads to around 30% discrepancy when using the subset of programs. As we described previously, the global optimal cluster number for the ATI programs is 31, meaning that almost each application stands as an individual cluster. This indicates that the execution patterns of ATI programs are not sufficiently close to each other compared to the Nvidia programs. As a consequence, the chosen 10 programs are not able to accurately represent the characteristics of all applications. Nevertheless, considering that the number of applications has been largely reduced, we believe that the validation result is still acceptable to reduce the benchmarking efforts. In general, the validation results indicate that our benchmark clustering is reasonable and the selected programs are representative of the entire suite.

B. Overall Execution Time Comparison

In general purpose GPU computing realm, the CPU side is usually referred as the *host* while the GPU is termed as the *device*. Previous studies have demonstrated that the data transfer between the host and the device costs even more time than the GPU computation does in some problems [23]. Given this consideration, we collect the time spent on different stages during execution and demonstrate the overall breakdown in Figure 3. As shown in the figure, the execution of each application is decoupled into three stages: memory copy from the host to device (*mem_H2D*), kernel execution (*kernel*), and the data transfer from the device back to the host (*mem_D2H*). Obviously, the selected applications have distinct characteristics on the execution time distribution. For applications such as *Histogram*, the time spent on communication between the CPU and the GPU dominates the total execution. On the contrary, the GPU computation takes most portion of the time in benchmarks including *EigenValue*. Several interesting findings can be observed from the figure.

First, for all 10 applications, the Nvidia computer system outperforms the ATI competitor from the standpoint of host-to-device data transfer. In addition, the time spent on the memory copy from the GPU to the CPU is also shorter on the Nvidia machine, except for *BlackScholes*. This indicates that the Nvidia system is able to transfer data more efficiently than the ATI computer. To further understand this issue, we conduct a group of experiments to test the memory transfer performance on both computer systems. Figure 4(a) illustrates the communication time when copying different sizes of data from the host to the device. Similarly, the time for *mem_D2H* is shown in Figure 4(b). In general, the results support our inference. However, when copying a large amount of data from the GPU to the CPU, ATI performs better.

In a CUDA application, the API *cudamemcpy* is called for data communication, whereas an OpenCL program uses the *CLEnqueueWritebuffer* function to transfer data to the GPU and then invokes the *CLEnqueuReadbuffer* routine to copy the computation result back to the host side. As can be observed, the *cudamemcpy* takes fairly short time (i.e., tens of microseconds) when the data size is small (e.g., < 1024KB); in contrast, the OpenCL API needs at least 1 millisecond (i.e., 1000 μs) regardless of the data size. Note that in both systems, the time hardly changes when the data size varies between 64KB and 1024KB. It is thereby reasonable to infer that the time should be majorly taken by the configuration overhead such as source and destination setup in this case. Therefore, the gap demonstrates that the OpenCL API for memory copies has a larger launch overhead than the corres-

TABLE V. EXECUTION INFORMATION ON THE ATI GPU

Workload	ALUBusy (%)	Packing ratio (%)
BinomialOption	62.51	31.1
Blackscholes	58.58	95.75
Eigenvalue	18.32	54.44
Fastwalsh	56.94	30.83
FloydWarshall	20.35	32.3
Histogram	21.03	33.5
Matmul_2_smem	54.4	81.04
Matmul_no_smem	15.4	73.5
MonteCarloDP	49.29	71.9
Radixsort	3.12	30.9

ponding CUDA routine. On the other hand, the OpenCL function *CLEnqueueReadbuffer* takes shorter transfer time when the data size is relatively large. This indicates that the ATI OpenCL implementation has specific advantages on transferring large chunk of data from the GPU to the CPU. The *BlackScholes* benchmark has the largest size of data that need to be read back to the host side, making the ATI system to be a faster device.

The kernel execution on the GPU is always considered as the most important part in studying GPU performance. In these 10 pairs of applications, seven of them run faster on the Nvidia GPU, while ATI performing better on *Blackscholes*, *MatMul_2_smem*, and *MatMul_no_smem* benchmarks. The kernel computation time of *EigenValue*, *FloydWarshall*, and *RadixSort* on Radeon HD 5870 is substantially longer than those on GTX 580. Table V lists the *ALUBusy* rate and packing ratios of these ten programs when executed on the HD 5870. As shown in the table, the three programs running faster on the ATI GPU have a common point that the VLIW packing ratio is fairly high (highlighted in light gray). Recall that Radeon HD 5870 includes 320 five-way VLIW processors working at 850MHz. Therefore, provided that the packing ratio is α, the theoretical peak performance can be calculated as [4]: $320 \times 5 \times \alpha \times 850\text{MHz} \times 2 = 2.72\ \alpha$ TFLOPS. Note that in this equation, the factor 2 is included because that the fused multiply-add (FMA) operation, which includes two floating point operations, is usually used while deriving peak throughput of a GPU in convention. Similarly, the maximal performance of the GTX 580 GPU is 512×1544MHz×2 = 1.581 TFLOPS. In comparison, the packing ratio α should be no less than 58% (i.e., 1.581/2.72) to make the ATI GPU run faster. Since the packing ratios of *BlackScholes*, *Matmul_2_smem*, and *Matmul_no_smem* are all greater than this threshold, these programs run faster. On the other aspect, *Eigenvalue*, *FloydWarshall*, and *RadixSort* have fairly low packing ratios; even worse, their *ALUBusy* rate are low during the execution (highlighted in dark grey). These two factors result in the poor performance of these three programs.

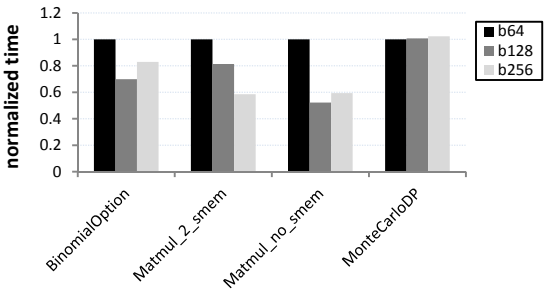
The third point that deserves detailed analysis is the double precision performance because of its importance in solving HPC problems. We use the *MonteCarloDP* application from financial engineering to compare the double precision computing capability of these two GPUs. This benchmark approximately achieves 70% packing ratio and 50% ALU utilization when running on the ATI GPU, which are adequately high for outstanding performance. However, its kernel execution time is remarkably longer compared to that on the Nvidia GPU. Unlike native benchmarks selected from the SDK, the CUDA version of *MonteCarloDP* is directly transformed from the OpenCL implementation. This indicates that the two programs are identical on both the algorithm design and the implementation details. We can conclude that the performance gap is from the hardware difference. Each SM on the GTX 580 is able to execute up to 16 double precision FMA operations per clock [13] with a peak throughput of 16×16×1544MHz×2 = 790.5 GFLOPS. In the Radeon HD 5870, however, the four ALUs within a VLIW processor cooperate to perform a double precision FMA per clock. Therefore, the maximal processing power is no more than 320×1×850MHz×2 = 544 GFLOPS. Obviously, the GTX 580 is more preferable for double precision computations.

C. Parallelism

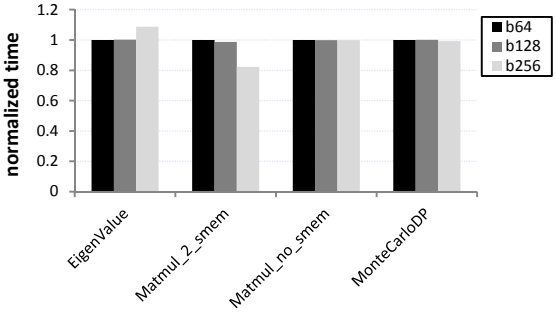
Execution parallelism stands as the heart of general purpose GPU computing. A typical GPGPU application usually launches a large amount of warps/wavefronts to hide long latencies encountered during the execution. In this section, we will investigate that how execution parallelism impacts the overall performance on these two GPUs.

We first observe the performance variations for changing the thread block size in Nvidia programs (work-group size for ATI programs). When the block size is changed, the number of blocks/work-groups resided on an SM/SIMD may vary accordingly. This in turn changes the execution parallelism. Clearly, the parallelism will be greatly reduced if there are too few warps/wavefronts on an SM or SIMD. In this case, the performance is likely to be degraded. Figure 5 shows the normalized execution time of selected benchmarks when the block size is set to 64, 128, and 256 respectively. Note that only a fraction of 10 applications are tested. The reason is that the group size is tightly fixed in the program implementation for some benchmarks. As a result, changing the configuration will violate the correctness of these applications. Therefore, we do not include such programs in this experiment.

As shown in Figure 5 (a), on the Nvidia platform, the execution time tends to become shorter when the block size is enlarged since the occupancy keeps rising in this circumstance except for *BinomialOption* and *Matmul_no_smem*, where the performance gets slightly worse if the block size is increased from 128 to 256. This is due to the fact that the number of global memory accesses is significantly increased when the group size becomes larger. In this case improving parallelism may degrade the overall performance. The other exception is that the performance of *MonteCarloDP* is hardly changed regardless of the thread block size. This is because that each thread of the kernel requires substantial registers, resulting in extremely few resident warps on an SM due to the resource constraint. Actually, the occupancy remains fairly low regardless of the block size while executing *MonteCarloDP*. Figure 5(b) demonstrates that the performance of these applications do not change much with varying work-group sizes on the ATI GPU. As described previously, the ATI GPU adopts the VLIW architecture; therefore, other factors including the ALU packing ratio are also playing significant roles in determining the execution performance.



(a) Nvidia benchmarks



(b) ATI benchmarks

Figure 5. Performance variation with changing the block size

Next, our second study concentrates on the impact of working size. The working size denotes the number of output elements calculated by each thread/work-item. By setting the working size to different values, it is conveniently to adjust the packing ratio on the ATI GPU. While executing on the Nvidia GPU, an appropriate working size can lead to efficient usage of the data fetched from the global memory and reduce the unnecessary memory accesses. This may improve the overall performance. In order to simplify the packing ratio tuning, we choose the *Matmul_no_smem* benchmark to conduct the study. Figure 6 illustrates the change of performance when the working size increases from 1 to 8 on both GPUs. As can be observed, the HD 5870 GPU greatly benefits from larger working sizes while the Nvidia GPU is not notably impacted by the variation of working sizes.

To further understand this issue, we record the occupancy and ALU packing ratio corresponding to each working size and show them in Figure 7. Both occupancies on two GPUs are reducing with the increase of working sizes. This is due to the resources constraint on an SM/SIMD. As each thread computes more elements, the number of registers which are allocated to store intermediate variables is inevitably increased. Therefore, fewer threads are allowed to reside on the same SM, resulting in a decreased occupancy. On the GTX 580 GPU, such decreased parallelism counteracts the advantage of improving efficiencies of single threads, making the overall performance slightly changed. However on the ATI GPU, since the calculation of each matrix element is independent, the compiler is able to assign the extra computations to the unoccupied slots within a VLIW processor, thus increasing the packing ratio. When the working size varies within a reasonable range, the high packing ratio is the dominant factor to the performance. Consequently, the HD 5870 GPU shows a performance boost when working size increases.

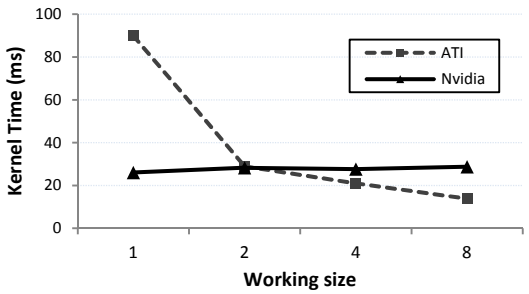


Figure 6. Performance variation with changing the working size

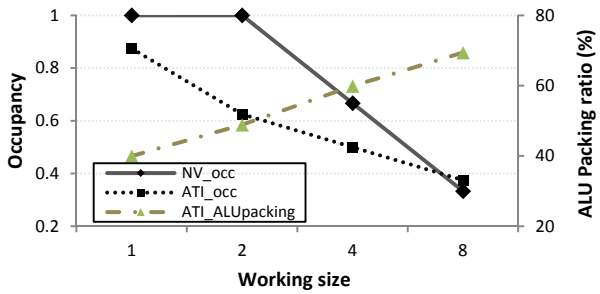


Figure 7. Occupancy and VLIW packing variations with changing the working size

Putting all of these together, we can conclude that the extraction of the optimal parallelism on two GPUs follows different patterns. On Nvidia GPU, we shall aim at increasing the SM occupancy in general, while paying attention to other factors such as the resource usage and memory access behavior. On the ATI GPU, improving the VLIW packing ratio is of great importance for higher performance.

D. Cache Hierarchy

In general purpose GPU programming, long latency events including global memory accesses can be hidden by switching among the available warps or wavefronts on an SM or SIMD. However, due to limited available warps and wavefronts, frequently global memory accesses tend to be the bottleneck for many GPU applications, especially when the parallelisms are not sufficiently high. Therefore, similar to CPU design, both Nvidia and ATI GPUs employ a cache hierarchy to shorten memory latency. In this section, we will investigate the architectural features of caches on these two GPUs.

We first focus on the GTX 580 GPU with new designs of on-chip fast memory. Our study starts from the performance comparison of selected benchmarks with the L1 cache enabled or disabled. The results are shown in Figure 8. As can be observed, eight out of ten applications show little impact on the inclusion of the L1 cache, except for *FloydWarshall* and *Matrixmul_ no_smem*. This indicates that those eight applications are running with superb parallelism, thus long latencies due to global memory operations can be hidden. On the contrary, the execution of *FloydWarshall* suffers from memory access latencies, therefore, the L1 cache is able to capture data locality and effectively improve the performance. The result of *MatrixMul_no_smem* is surprising since the execution time is getting even longer when the L1 cache is enabled. We thereby conduct a case study based on this benchmark to reveal the underlying reasons.

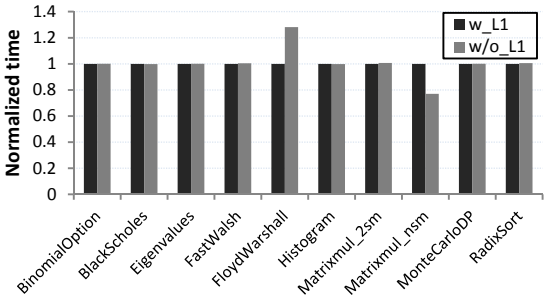


Figure 8. Execution time on GTX 580 when the L1 is enabled/disabled

In *MatrixMul_no_smem*, each thread is responsible for calculating four adjacent elements in a column of the output matrix. This is illustrated in Figure 9 (labeled as *vertical* in Matrix *C*). When a thread is calculating the first element, it will load a block of consecutive data from the corresponding line in matrix A. According to [6], the L1 cache line size in the GTX 580 is 128 bytes, while the L2 cache line size is 32B. Therefore, when an L1 cache miss is encountered, a 128B segment transaction will be always issued. As the thread continues to calculate the second element, a global memory read request is issued again to load the data from the following line in matrix A. Note that all threads within the same SM shares the L1 cache. This implies that a previously cached block might be evicted in order to accommodate the new fetched data requested by a more recent L1 miss. In this program, the memory access pattern is quite scattered. Only a small fraction of the 128-byte cached data is utilized and the resultant global memory transactions tend to waste the memory bandwidth. However, when the L1 cache is disabled, all global memory requests directly go through the L2 cache which issues 32 byte transactions. Therefore, the global memory bandwidth is more efficiently used, leading to better performance.

Based on this analysis, we modify the kernel and make each thread calculate four adjacent elements in the same line of matrix *C* (labeled as *horizontal* in Figure 9) for better reuse of L1 cache data. To validate these two cases (i.e., *vertical* and *horizontal*), we carry out a group of experiments by setting the input matrix to different sizes. The result is demonstrated in Figure 10. As we expect, in the *horizontal* implementation, the computation throughput is much higher when the L1 cache is enabled. In contrast, disabling the L1 cache can yield better performance for the *vertical* program.

The caches involved in the Radeon HD 5870 GPU have different design specifications from that on the Nvidia GPU. In specific, both the L1 and L2 caches on the HD 5870 are only able to store images and read-only constants. Many data structures used in GPGPU application kernels such as *float* type arrays are uncacheable. In the OpenCL programming, this can be worked around by defining the target structures as *image objects* and use the corresponding routines for data accesses. In order to understand the effect of the caches on the HD 5870, we compare the performance of two matrix multiplication programs, one of which is designed to use the caches. In Figure 11, the curve labeled by "image object" corresponds to the version using caches. Note that these two programs are built on identical algorithms and neither of them uses the local data share; hence the performance gap comes directly

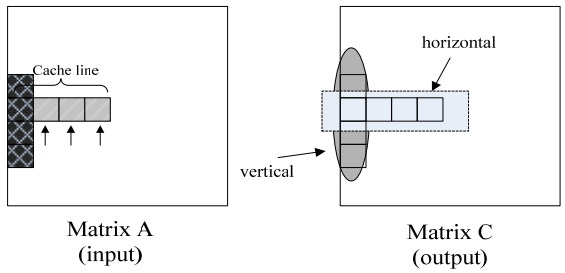
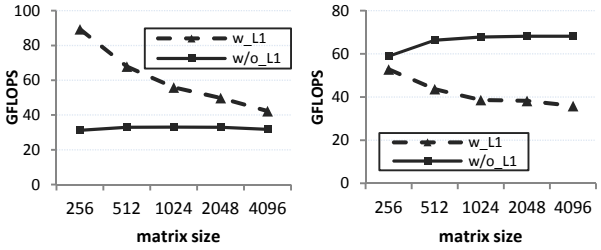


Figure 9. Two versions of Matrix multiplication implementations



(a) Horizontal (b) Vertical

Figure 10. Performance comparison of two versions of matrix multiplications executed on GTX 580

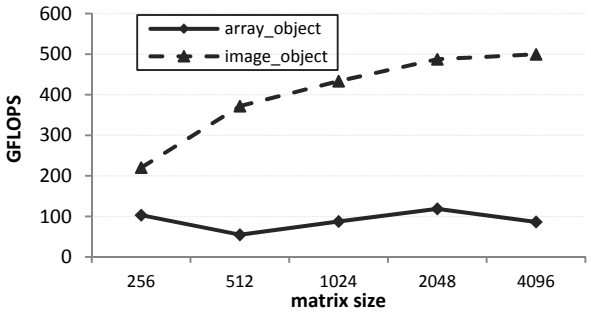


Figure 11. Performance of matrix multiplication on HD 5870

from caches. Obviously, when setting the data array type to image object, the performance is boosted tremendously.

In summary, there are several architectural differences between the caches on the GTX 580 and Radeon HD 5870 GPUs. While programming cache-sensitive applications on Fermi GPUs, the data access patterns and kernel workflows should be carefully designed, in order to effectively and efficiently use the L1 cache. The caches on the HD 5870 are less flexible compared to that on the GTX 580. To take the advantage of caches on the ATI GPU, cacheable data structures such as image objects should be appropriately used in the programs.

E. Energy Efficiency

As power-consuming GPUs are widely used in supercomputers, high energy efficiency is becoming an increasingly important design goal. As we described in section I, both Nvidia and ATI pay substantial attention to trimming the power budget of their products while improving the performance. Therefore, evaluating energy efficiencies of the target GPUs is of great importance.

Figure 12 shows the power consumptions of selected benchmarks running on two GPUs. Obviously, the Fermi

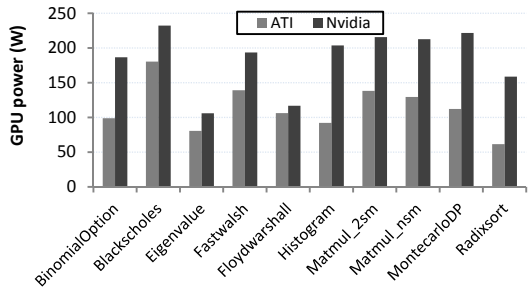


Figure 12. Power consumption comparison of two GPUs

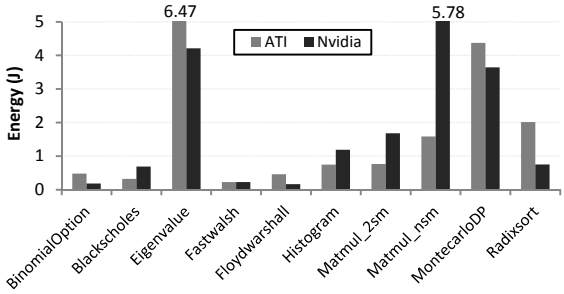


Figure 13. Energy consumption comparison of two GPUs

GPU consumes more power than the ATI counterpart. Recall the manufacture parameters listed in Table I. The GTX 580 integrates more transistors and its processor cores are running on a higher frequency compared to the HD 5870. Therefore, the Nvidia GPU tends to consume more power during program execution. The energy consumption of these benchmarks is shown in Figure 13. We observe four of those selected applications consume less energy on the ATI GPU. Because of the relative low power consumption, the HD 5870 consumes less energy to solve a problem when its execution time is not significantly longer than that on the GTX 580.

The energy efficiency can be interpreted by the metric Energy-delay product (EDP). We demonstrate the normalized EDP for these applications in Figure 14. As shown in the figure, the HD 5870 GPU wins on four of them: *BlackScholes*, *Histogram*, *MatrixMul_2sm*, and *MatrixMul_nsm*. Note that three benchmarks from these four contain efficient OpenCL kernels with fairly high VLIW packing ratios. This indicates that the VLIW packing is also critical to the energy efficiency of the HD 5870 GPU. In case where a compact packing is easy to explore, the Radeon HD 5870 is more preferable from the standpoint of high energy efficiency. In general, we can summarize a principle that the ATI GPU can deliver better energy efficiency when the program can perfectly fit the VLIW processors; otherwise the GTX 580 card is more preferable.

V. Related Work

In recent years, several researchers have authored outstanding studies on modern GPU architecture. On the performance analysis aspect, Hong et al. [25] introduce an analytical model with memory-level and thread-level parallelism awareness to investigate the GPU performance. In [36], Wong et al. explore the internal architecture of a widely used Nvidia

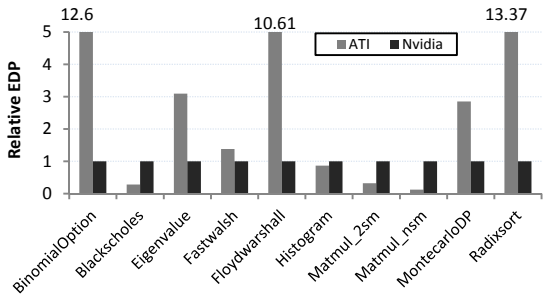


Figure 14. Energy efficiency comparison of two GPUs

GPU using a set of micro-benchmarks. More recently, Zhang and Owens [37] use a similar micro-benchmark based approach to quantitatively analyze the GPU performance. Studies on typical ATI GPUs are even fewer. Taylor and Li [35] develop a micro-benchmark suite for ATI GPUs. By running the micro-benchmarks on different series of ATI products, they discover the major performance bottlenecks on those devices. In [38], Zhang et al. adopt a statistical approach to investigate characteristics of the VLIW structure in ATI Cypress GPU.

Literature on the GPU power/energy analysis can also be found in prior studies. Hong and Kim [26] propose an integrated GPU power and performance analysis model which can be applied without performance measurements. Zhang [38] and Chen [17] use similar strategies to statistically correlate the GPU power consumption and its execution behaviors. The established model is able to identify important factors to the GPU power consumption, while providing accurate prediction for the runtime power from observed execution events. Huang et al. [27] evaluate the performance, energy consumption and energy efficiency of commercial GPUs running scientific computing benchmarks. They demonstrate that the energy consumption of a hybrid CPU+GPU environment is significantly less than that of traditional CPU implementations. In [33], Rofouei et al. draw a similar conclusion that a GPU is more energy efficient compared to a CPU when the performance improvement is above a certain bound. Ren et al. [32] consider even more complicated scenarios in their study. The authors implement different versions of matrix multiplication kernels, running them on different platforms (i.e., CPU, CPU+GPU, CPU+GPUs) and comparing the respective performance and energy consumptions. Their experiment results show that when the CPU is given an appropriate share of workload, the best energy efficiency can be delivered.

Efforts are also made to evaluate comparable architectures in Prior works. Peng et al. [29][30] analyze the memory hierarchy of early dual-core processors from Intel and AMD and demonstrate their respective characteristics. In [24], Hackenberg et al. conduct a comprehensive investigation on the cache structures on advanced quad-core multiprocessors. In recent years, comparison between general purpose GPUs is becoming a promising topic. Danalis et al. [18] introduce a heterogeneous computing benchmark suite and investigate the Nvidia GT200 and G80 series GPU, ATI Evergreen GPUs, and recent multicore CPUs from Intel and AMD by running the developed benchmarks. In [19], Du et al. compare the performance between an Nvidia Tesla C2050 and an ATI HD

5870. However, their work emphasizes more on the comparison between OpenCL and CUDA as programming tools. Recently, Ahmed and Haridy [14] conduct a similar study by using an FFT benchmark to compare the performance of an Nvidia GTX 480 and an ATI HD 5870. However, power and energy issues are not considered in their work.

On the other hand, benchmark clustering has been proved to be useful for computer architecture study. Phansalkar et al. [31] demonstrate that the widely used SPEC CPU benchmark suite can be classified into a number of clusters based on the program characteristics. In [22], Goswami et al. collect a large amount of CUDA applications and show that they can also be grouped into a few subsets according to their execution behaviors.

Our work adopts the benchmark clustering approach. We believe that the applications in the SDKs provide the most typical GPU programming patterns that reflect the characteristics of these two devices. Therefore, we can extract and compare the important architectural features by running the selected applications.

VI. CONCLUSION

In this paper, we use a systematic approach to compare two recent GPUs from Nvidia and ATI. While sharing many similar design concepts, Nvidia and ATI GPUs differ in several aspects from processor cores to the memory subsystem. Therefore, we conduct a comprehensive study to investigate their architectural characteristics by running a set of representative applications. Our study shows that these two products have distinct advantages and favor different applications for better performance and energy efficiency.

ACKNOWLEDGMENT

This work is supported in part by an NSF grant CCF-1017961, the Louisiana Board of Regents grant NASA / LEQSF (2005-2010)-LaSPACE and NASA grant number NNG05GH22H, NASA(2011)-DART-46, LQESF(2011)-PFUND-238 and the Chevron Innovative Research Support (CIRS) Fund. Ying Zhang is holding a Flagship Graduate Fellowship from the LSU graduate school. We acknowledge the computing resources provided by the Louisiana Optical Network Initiative (LONI) HPC team. Finally, we appreciate invaluable comments from anonymous reviewers which help us finalize the paper.

REFERENCES

[1] AMD Corporation. AMD PowerPlay Technology. http://www.amd.com/us/products/technologies/ati-power-play/Pages/ati-power-play.aspx.

[2] AMD Corporation. AMD Stream Profiler. http://developer.amd.com/gpu/amdappprofiler/pages/default.aspx.

[3] AMD Corporation. AMD Stream SDK. http://developer.amd.com/gpu/amdappsdk/pages/default.aspx.

[4] AMD Corporation. ATI Radeon HD 5870 Graphics. http:// www.amd.com/us/products/desktop/graphics/ati-radeon-hd-5000/hd-5870/Pages/ati-radeon-hd-5870-overview.aspx#2.

[5] Nvidia Corporation. CUDA Toolkit 3.2. http://developer.nvidia.com/cuda-toolkit-32-downloads.

[6] Nvidia Corporation. Fermi Optimization and advice.pdf.

[7] Nvidia Corporation. GeForce GTX 580. http://www.nvidia.com/object/product-geforce-gtx-580-us.html.

[8] Nvidia Corporation. Nvidia PowerMizer Technology. http://www.nvidia.com/object/feature_powermizer.html.

[9] Nvidia Corporation. What is CUDA? http:// www.nvidia.com/object/what_is_cuda_new.html.

[10] OpenCL – The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl.

[11] Top 500 Supercomputer sites. http://www.top500.org.

[12] AMD Corparation. ATI Radeon HD5000 Series: In inside view. June 2010.

[13] Nvidia Corporation. Nvidia's Next Generation CUDA Compute Architecture: Fermi. September 2009.

[14] M. F. Ahmed and O. Haridy, "A comparative benchmarking of the FFT on Fermi and Evergreen GPUs", in Poster session of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), April 2011.

[15] J. D. Banfield and A. E. Raftery, "Model-based Gaussian and non-Gaussian clustering. Biometrics", Biometrics, vol. 49, September 1993, pp. 803-821.

[16] G. Celeux and G. Govaert, "Comparison of the mixture and the classification maximum likelihood in cluster analysis", The Journal of Statistical Computation and Simulation. vol. 47, September 1991, pp. 127-146.

[17] J. Chen, B. Li, Y. Zhang, L. Peng, and J.-K. Peir, "Tree structured analysis on GPU power study", in Proceedings of the 29th IEEE international Conference on Computer Design (ICCD), Amherst, MA, Oct. 2011.

[18] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite", in Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU), March 2010.

[19] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From CUDA to OpenCL: towards a performance-portable solution for multi-platform GPU programming", Technical report. Department of Computer Science, UTK, September 2010.

[20] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns", in Proceedings of the National Academy of Sciences of the USA, vol. 95, October 1998, pp. 14863-14868.

[21] C. Fraley and A. E. Raftery, "Model-based clustering, discriminant analysis and density estimation", Journal of the American Statistical Association, vol. 97, June 2002, pp. 611–631.

[22] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring GPGPU workloads: characterization methodology, analysis and microarchitecture evaluation implication", in Proceedings of IEEE International Symposium on Workload Characterization (IISWC), December 2010.

[23] C. Gregg and K. Hazelwood, "Where is the data? Why you cannot debate CPU vs. GPU without the answer", in Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), April 2011.

[24] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems", in Proceedings of 42nd International Symposium on Microarchitecture (MICRO), New York, December 2009.

[25] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in Proceedings of 36th Annual International Symposium on Computer Architecture (ISCA), June 2009.

[26] S. Hong and H. Kim, "An integrated gpu power and performance model," in Proceedings of 37th Annual International Symposium on Computer Architecture (ISCA), June 2010.

[27] S. Huang, S. Xiao and W. Feng, "On the energy efficiency of graphics processing units for scientific computing," in Proceedings of 5th IEEE Workshop on High-Performance, Power-Aware Computing (in

conjunction with the 23rd International Parallel & Distributed Processing Symposium), June 2009.

[28] G. J. McLachlan, and K. E. Basford, "Mixture Models: Inference and Applications to Clustering. Dekker" , New York, 1998.

[29] L. Peng, J.-K. Peir, T. K. Prakash, C. Staelin, Y-K. Chen, and D. Koppelman, "Memory hierarchy performance measurement of commercial dual-core desktop processors", in Journal of Systems Architecture, vol. 54, August 2008, pp. 816-828.

[30] L. Peng, J.-K. Peir, T. K. Prakash, Y-K. Chen, and D. Koppelman, "Memory performance and scalability of Intel's and AMD's dual-core processors: a case study", in Proceedings of 26th IEEE International Performance Computing and Communications Conference (IPCCC), April 2007.

[31] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Measuring program similarity: experiments with SPEC CPU Benchmark Suites", in Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), March 2005.

[32] D. Ren and R. Suda, "Investigation on the power efficiency of multi-core and gpu processing element in large scale SIMD computation with CUDA", in Proceeding of 1st Green Computing Conference, August 2010.

[33] M. Rofouei, T. Stathopulous, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-aware high performance computing with graphics processing units", in Workshop on Power-Aware Computing and Systems (HotPower), December 2008.

[34] G. Schwarz, "Estimating the dimension of a model", The Annals of Statistics, vol. 6, March 1978, pp. 461–464.

[35] R. Taylor and X. Li, "A micro-benchmark suite for AMD GPUs", in Proceedings of 39th International Conference on Parallel Processing Workshops, September 2010.

[36] H. Wong, M. Papadopoulou, M, Alvandi, and A. Moshovos, "Demistifying GPU microarchitecture through microbenchmarking", in Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS), March 2010.

[37] Y. Zhang and J. Owens, "A quantitative performance analysis model for GPU architectures," in Proceedings of 17th IEEE Symposium on High Performance Computer Architecture (HPCA), February 2011.

[38] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and Power Analysis of ATI GPU: A statistical approach", in Proceedings of the 6th IEEE International Conference on Networking, Architecture, and Storage (NAS), Dalian, China, July 2011.