On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing

Mayank Daga, Ashwin M. Aji, and Wu-chun Feng
Dept. of Computer Science
Virginia Tech
Blacksburg, USA
{mdaga, aaji, feng}@cs.vt.edu

***Abstract*—The graphics processing unit (GPU) has made significant strides as an accelerator in parallel computing. However, because the GPU has resided out on PCIe as a discrete device, the performance of GPU applications can be bottlenecked by data transfers between the CPU and GPU over PCIe. Emerging heterogeneous computing architectures that "fuse" the functionality of the CPU and GPU, e.g., AMD Fusion and Intel Knights Ferry, hold the promise of addressing the PCIe bottleneck.**

In this paper, we empirically characterize and analyze the efficacy of AMD Fusion, an architecture that combines general-purpose x86 cores and programmable accelerator cores on the same silicon die. We characterize its performance via a set of micro-benchmarks (e.g., PCIe data transfer), kernel benchmarks (e.g., reduction), and actual applications (e.g., molecular dynamics). Depending on the benchmark, our results show that Fusion produces a *1.7 to 6.0-fold* improvement in the data-transfer time, when compared to a discrete GPU. In turn, this improvement in data-transfer performance can significantly enhance application performance. For example, running a *reduction* benchmark on AMD Fusion with its mere 80 GPU cores improves performance by *3.5-fold* over the discrete AMD Radeon HD 5870 GPU with its 1600 more powerful GPU cores.

***Keywords*-AMD Fusion; graphics processing unit; GPU; GPGPU; accelerated processing unit; APU; OpenCL; performance evaluation; benchmarking; heterogeneous computing;**

I. Introduction

The widespread adoption of compute-capable graphics processing units (GPUs) in desktops and workstations has made them attractive as accelerators for high-performance parallel computing [1]. Their increased popularity has been due in part to a unique amalgamation of performance, power, and energy efficiency. In fact, three out of the top five fastest supercomputers in the world, according to the Top500, employ GPUs [2].

A closer look at the Top500 list, however, reveals that the supercomputers powered by GPUs attain only ∼50% of their theoretical peak performance, whereas non-GPU-powered supercomputers attain ∼78% of the theoretical peak [2]. This implies that there are certain aspects of GPUs that limit their performance for Linpack, the benchmark used to rank supercomputers on the Top500. Since GPUs have traditionally resided on PCI Express (PCIe), additional overhead costs are incurred for host-to-GPU data transfers and vice versa. As a consequence, GPU applications are oftentimes bottlenecked by the PCIe data transfers. Thus, GPUs are *not* a panacea [3]–[5].

With the emergence of heterogeneous computing architectures that "fuse" the functionality of the CPU and GPU onto the same die, e.g., AMD Fusion and Intel Knights Ferry, there is an expectation that the PCIe bottlenecks would be addressed. In these architectures, the x86 CPU cores and the programmable GPU cores share a common path to system memory. Also present are high-speed block transfer engines, which assist in data movement between the x86 and GPU cores. Hence, data transfers never hit the system's external bus, thereby mitigating the adverse effects of slow PCIe.

In this paper, we present an empirical characterization and analysis of the effectiveness of the AMD Fusion architecture. To the best of our knowledge, this work is the first to do so. The processor built upon the Fusion architecture is called an *accelerated processing unit* or APU. The APU combines the general-purpose x86 cores of a CPU with programmable vector-processing engines of a GPU onto a single silicon die. We then re-visit Amdahl's Law for today's era of accelerated processors. Specifically, we show that the fused CPU+GPU cores enable better performance than a discrete GPU and even traditional multi-core CPU processors by reducing the *parallel overhead* of PCIe data transfers.

To characterize the performance of the AMD Fusion architecture, we use four benchmarks from the Scalable HeterOgeneous Computing (SHOC) benchmark suite [6] as well as the OpenCL PCIe bandwidth test. Via these benchmarks, we show that the Fusion architecture can overcome the PCIe bottleneck associated with a discrete GPU, though not always.

For the first-generation AMD Fusion, the APU, which is a fused combination of CPU+GPU, delivers better performance than a discrete CPU+GPU combination when the amount of data to be transferred between the CPU and GPU exceeds a minimum threshold and when the amount of computation on the GPU cores is not high enough for the discrete GPU to amortize the PCIe data-transfer overhead.

Our empirical results indicate that the APU improves data-transfer times by *1.7 to 6.0-fold* over the discrete CPU+GPU combination. For one particular benchmark, i.e., *reduction*, the total execution time is *3.5-fold* better on the Fusion APU than on the discrete GPU despite the latter having 20 times more GPU cores and more powerful cores at that. In turn, the improvement in data-transfer times reduces the parallel overhead, thus providing more parallelism to the application.

The rest of this paper is organized as follows. Section II presents an overview of AMD GPUs and the issue of the PCIe bottleneck in discrete CPU+GPU platforms. We then introduce the AMD Fusion architecture and outline why it holds the promise of overcoming this bottleneck. In Section III, we re-visit Amdahl's Law for accelerator-based processors. In Section IV, we illustrate and discuss the results of our experiments. Section V presents related work, followed by conclusions and future work in Section VI.

II. Background

Here we present an overview of the AMD GPU and discuss the effect of the PCIe bottleneck, which oftentimes proves to be an obstacle towards achieving better overall application performance. We then describe the architecture of the first generation of *accelerated processing unit (APU)*, i.e. AMD Fusion E-Series APU, and show how it holds the promise of overcoming the PCIe bottleneck.

A. Overview of the AMD GPU

An AMD GPU follows a classic graphics design, which is highly tuned for single-precision, floating-point arithmetic and common image operations on two-dimensional and image data. Fig. 1 provides an architectural overview.

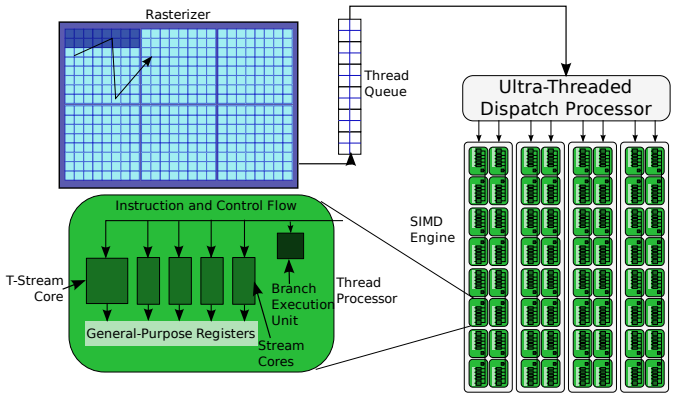


Fig. 1. Overview of an AMD/ATI Stream Processor and Thread Scheduler.

In this case, the compute unit is known as a *SIMD engine* and contains several thread processors, each containing four stream cores, along with a special-purpose core and a branch execution unit. The special-purpose (or T-Stream) core is designed to execute certain mathematical functions in hardware, e.g., transcendentals like $sin()$, $cos()$ and $tan()$. Since there is only one branch execution unit for every five processing cores, any branch in the program incurs some serialization to determine the path each thread should take. The execution of *divergent* branches is performed in lock-step manner for all the cores present in a compute unit. In addition, the processing cores are vector processors, which means that using vector types can produce material speedup on AMD GPUs.

Discrete GPUs from AMD house a large number of processing cores, ranging from 800 to 1600 cores. As a result, a humongous number of threads need to be launched in order to keep all GPU cores fully occupied. However, to run many threads, the amount of registers used per thread has to be kept to a minimum. That is, all the registers utilized per thread need to be stored in a register file, and hence, the total number of threads that can be scheduled is limited by the size of the register file, which is a generous 256 KB on the latest generation of AMD GPUs.

Another unique architectural feature of AMD GPUs is the presence of a *rasterizer* – for working with two-dimensional matrices of threads and data. Hence, accessing scalar elements stored contiguously in memory is *not* the most efficient access pattern. Accessing scalar elements can be made slightly more efficient by doing so in chunks of 128 bits due to the presence of vector cores. Loading these chunks from image memory, which uses the memory layout best matched to the memory hardware on AMD GPUs, also results in significant improvement in performance.

B. PCIe Bottlenecks with Discrete GPUs

In Fig. 2, we demonstrate the cause of PCIe bottlenecks with a discrete GPU. As shown, the x86 *host* CPU can access the system memory as well as initiate functions on the GPU. However, because the GPU resides on PCIe, a DMA is required to transfer data from the system memory of the CPU to device memory of the GPU to perform any useful work. Although the GPU can execute hundreds of billions of floating-point operations per second, the current PCIe interconnects can transfer approximately only a gigaword per second [7]. Due to this limitation, it behooves the GPU application programmer to ensure high data reuse on the GPU to be able to successfully amortize the cost of slow PCIe transfers, and in turn, achieve substantial performance benefits.

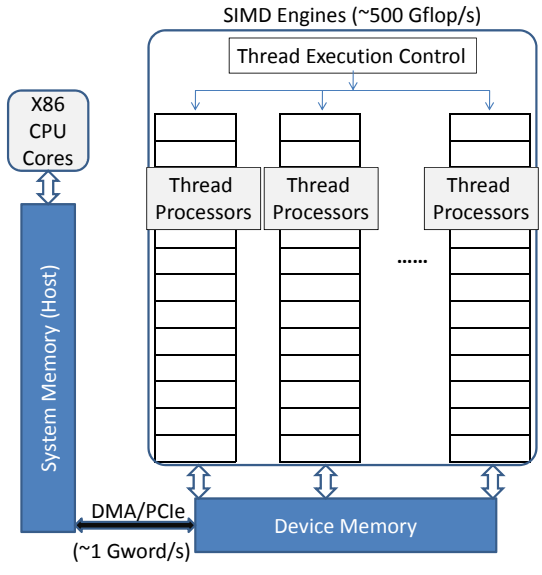


Fig. 2. Architectural Layout of a Discrete GPU.

However, ensuring high data reuse in all applications may not be possible. For example, there might be applications for

which the execution time on a GPU is less than the time it takes to get data onto the GPU or applications whose execution profiles consist of iterating over DMA transfers and GPU execution. For such applications, discrete GPUs may not be the appropriate to accelerate performance.

Emerging architectures like AMD Fusion seek to address these issues by eliminating PCIe access to the GPU by "fusing" CPU and GPU functionality onto a single silicon die.

C. AMD Fusion Architecture

At the most basic level, the Fusion architecture combines general-purpose scalar and vector processor cores onto a single silicon die, thereby forming a heterogeneous computing processor. It aims to provide the "best of both worlds" scenario in the sense that scalar workloads, like word processing and web browsing, use the x86 cores whereas vector workloads, like parallel data processing, use the GPU cores.

Fig. 3 depicts a block diagram of this novel architecture. The key aspect to note is that the x86 CPU cores and the vector (SIMD) engines are attached to the system memory via the same high speed bus and memory controller. This architectural artifact allows the AMD Fusion architecture to alleviate the fundamental PCIe constraint that has traditionally limited performance on a discrete GPU. Apart from the processing cores, Fusion also consists of the following system elements: memory controller, I/O controller, video decoder, display output, and bus interfaces, all on the same die.

Although Fusion's x86 cores and SIMD engines share a common bus to the system memory, the first-generation implementation of Fusion divides system memory into two parts — one that is visible to and managed by the operating system running on x86 cores and one that is managed by the software running on the SIMD engines. Therefore, even on the Fusion architecture, data has to be moved from the operating system's portion of system memory to that portion that is visible to the SIMD engines. However, unlike discrete GPUs, where these data transfers from system memory to device memory hit PCIe, the data transfers on Fusion are expected to amount to a `memcpy`, as logically captured in Fig. 4. Moreover, AMD currently provides high-speed block transfer engines that move data between the x86 and SIMD memory partitions. Therefore, the Fusion architecture holds the promise of improving performance for all applications that were previously bottlenecked by PCIe transfers. Future APU architectures are expected to have these memories seamlessly merged [7], which means that there will not be a need to transfer data to and from the GPU memory at all.

Programming on Fusion is facilitated by the emerging OpenCL standard. Therefore, existing applications written in OpenCL for the discrete CPU and GPU combination can be run without modification on the fused CPU+GPU of Fusion.

III. Revisiting Amdahl's Law

We first briefly review Amdahl's Law [8], followed by a theoretical discussion on Hill and Marty's work on applying it to symmetric and asymmetric multi-core chips [9]. We then re-visit Amdahl's Law, specifically for accelerators, and show that fused asymmetric CPU+GPU cores for an APU enable more parallelism in the code than discrete GPUs or traditional multi-core symmetric processors.

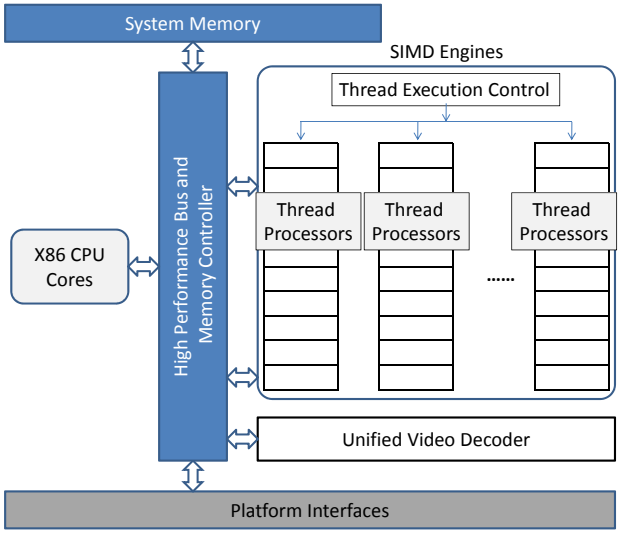


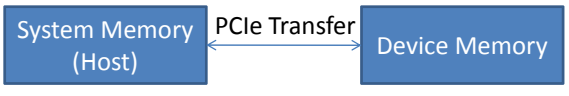
Fig. 3. Architectural Layout of the AMD Fusion Architecture.

The speedup of parallel applications on multi-processor architectures is limited by Amdahl's Law, which implies that the speedup obtained by implementing an application in parallel is dependent upon the fraction of the workload that can be parallelized [8]. Hence, the speedup, S, for a parallel application is given by (1).

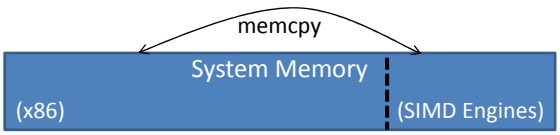
$$S = \frac{1}{s + p/N} \tag{1}$$

where p = parallel fraction of the application
s = serial fraction of the application, i.e., $(1 - p)$
and N = number of processors

Amdahl's law holds true in the ideal scenario for any multi-processor system if we assume the workload to be constant, i.e., strong scaling. This also makes the assumption that all the processors have the same overall computational capabilities.



(a) Discrete GPU.



(b) AMD Fusion (First Generation).

Fig. 4. Description of Data Transfers.

If $N \to \infty$ then,

$$S = \frac{1}{s} \tag{2}$$

Informally, this means that even when the serial fraction of the work is small, the maximum speedup obtainable from an infinite number of parallel processors is limited by $1/s$.

Hill and Marty [9] first categorize multi-core chips into three groups based on how the on-chip unit resources, or *base core equivalents (BCEs)*, are combined to form larger processing cores. Specifically, they classify chips into symmetric, asymmetric, and dynamic multi-core chips and then theoretically analyze the attainable speedups on each platform.

Symmetric chips are the traditional multi-core chips, where every processor has the same computational capability. Equation (1) can be directly applied to symmetric chips with the implication that it is critical for the programmer to extract parallelism from their code.

Dynamic chips are idealistic chips, where the cores can be dynamically combined to boost the performance of the serial fraction of the program, thereby providing maximum efficiency even if the code has a fairly large serial fraction. We do not study dynamic chips in this paper. However, we will revisit them in the future to study the next-generation AMD A-Series APUs, which promise to improve power efficiency by dynamically turning on and off the CPU and GPU resources depending on the application load (AMD Power Gating) [10].

On the other hand, asymmetric chips are those that have one large complex core for sequential programming and several other simpler cores that help the larger core in parallel processing. Hill and Marty show that asymmetric multi-cores offer more potential speedup than their symmetric counterparts even for lower values of p [9]. For example, they show that for p = 0.975 and N = 256, the best asymmetric speedup is 125.0, whereas the best symmetric speedup is 51.2. However, they make an idealistic assumption that the parallel fraction of the program utilizes all the available cores completely. This is only possible if there is a perfect co-scheduling mechanism that enables complete utilization of the on-chip resources. Nevertheless, it is evident that asymmetric multi-core chips are more efficient than the symmetric ones [9].

We now study Amdahl's Law for accelerator-based systems, which can be considered to be a special type of asymmetric multi-cores, where the accelerator cores and the serial processor may be separated by PCIe. In general, Amdahl's Law ignores the overhead incurred due to parallelizing a workload. On any multi-core processor, this overhead is largely due to setup of parallel threads, interprocessor communication, and thread rejoining. Therefore, the speedup obtained is always less than the ideal. Furthermore, the overhead incurred by parallelizing an application on an accelerator-based system, especially the GPU, is even higher because data has to be transferred over the slow PCIe. This fact is corroborated by one of our micro-benchmark results, as shown in Fig. 5.

This particular micro-benchmark performs a `fmad` operation between each element of two `float-type` arrays of size 96 MB each. It is executed on three different platforms, i.e., a modern four-core CPU, a discrete CPU and GPU (AMD Radeon HD 5870) and a fused CPU+GPU (AMD E-Series Zacate APU). We use OpenMP as the parallel programming platform for the four-core processor and OpenCL for the Radeon GPU and the Zacate APU. The figure shows the total execution time as the sum of (i) the execution time of the serial part, (ii) the execution time of the parallel part, and (iii) the overhead incurred due to parallelization, i.e., device buffer creation, destruction, and buffer transfer. (We have not included the constant OpenCL setup time, i.e. kernel compilation, program and platform initialization.)

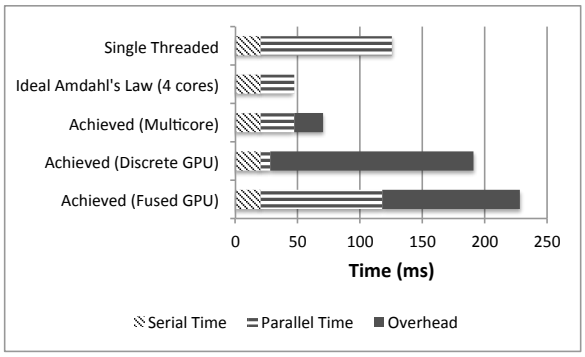


Fig. 5. Characterization of Parallel Overhead.

The execution time of the 'parallel part' in the case of the discrete GPU and APU is the kernel execution time. The single-thread implementation depicts the serial and parallel fractions of the code, while in case of the *ideal* Amdahl's law, the parallel part is sped up by four-fold (on a four-core CPU) with zero overhead. While the actual multi-core implementation, parallelized using OpenMP, does contain parallel overhead, it is negligible when compared to the overhead incurred due to parallelization on the accelerated platforms.

For the discrete GPU, however, the parallel overhead is so significant that it is *more than the sum of execution times of the serial and parallel parts*. So, while the execution time of the parallel part on the discrete GPU is substantially better than that on the multi-core CPU, the overhead is so much more that it does *not* make the micro-benchmark amenable for GPU processing. It also demonstrates the bottleneck caused by communication over PCIe.

Lastly, the APU (or fused CPU+GPU) does assist in reducing the parallel overhead. However, due to the presence of computationally less powerful SIMD cores, the execution time of the parallel part is longer than on the discrete GPU.

To apply Amdahl's Law to accelerator-based platforms, we model the following two factors:

- **Accelerated Parallel Fraction (p'):** For traditional symmetric multi-core processors, the parallel fraction p would ideally be optimized to p/N. However, accelerator cores have very different computational capabilities from the serial cores; so, using the terms p and N in the equation will be meaningless. The actual performance on the accelerators can vary depending on how the algorithm

is mapped to its cores. So, we change the performance of the parallel fraction p to p', as shown in the Fig. 6. Note that p' can be greater than p for a poor mapping.

- **Parallel Overhead (o):** For traditional symmetric multi-core processors, the parallel overhead (o) is typically negligible given that the inter-processor communication is generally small. For accelerator-based platforms, however, the parallel overhead o forms a significant part of the program because of the data transfer overhead between CPU memory and device (GPU) memory.

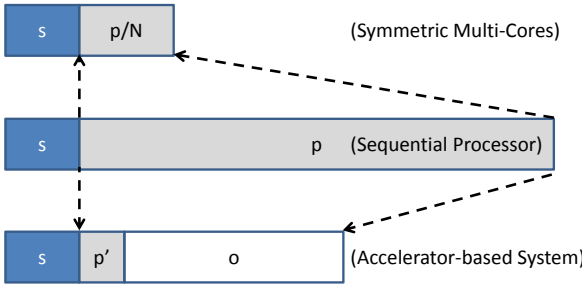


Fig. 6. Code Partitioning for Different Parallel Programming Platforms.

Therefore, the new speedup, S', for a parallel application can be represented as:

$$S' = \frac{1}{s + p' + o} \quad (3)$$

where, p' = accelerated parallel fraction

o = parallel overhead

It is still important to find enough parallelism in the code to minimize s, so that the parallel fraction in the program can be maximized. However, because discrete GPUs have large o, the benefit of the extracted parallelism p' is substantially reduced as a result. On the other hand, the parallel overhead o for AMD Fusion is much smaller and will therefore benefit more from a larger parallel fraction in the code, i.e., p'. Architectures like AMD Fusion resemble true asymmetric multi-core processors with much higher resource utilization and efficiency, as argued by Hill and Marty [9].

In summary, AMD Fusion should perform better than a discrete GPU with equivalent SIMD compute units if the extracted parallelism p' remains the same, because the parallel overhead, o, is relatively smaller when compared to its discrete GPU counterpart. However, in practice, the parallel fraction p' can be larger for memory-bound applications running on the APU platform because the memory bandwidth on a traditional CPU is lower than that of the discrete GPU. As a consequence, we can expect to achieve better speedup S' for AMD Fusion provided that the memory interface to the SIMD cores is as fast as the one on a discrete GPU.

However, are GPUs the *panacea* for all problems? Will the GPUs always out-perform multi-core CPUs? The answer depends on the application, the type of GPU, and the co-scheduling strategy (p'). Architectures such as AMD Fusion are a step in the right direction, as the results presented in Fig. 5 make a strong point in favor of this novel architecture. If the APU comes with more powerful SIMD engines and memory interfaces that are as fast as those in a discrete GPU, then the performance of the accelerated applications will only get better on these fused platforms.

IV. Experiments and Discussions

Here we present and discuss the results of our experiments that help to demonstrate the efficacy of a fused CPU+GPU processor like the AMD E-series Zacate APU, an instantiation of the AMD Fusion architecture.

A. Experimental Setup

Table I provides information about our three test platforms: AMD Zacate APU, AMD Radeon HD 5870, and a discrete version of the GPU on the Zacate APU, namely the AMD Radeon HD 5450. It is important to note that the AMD Zacate APU is only an experimental system in a laptop form factor.

We selected the PCIeBandwidth test from the AMD Stream SDK version 2.3 as well as four benchmarks from the Scalable HeterOgeneous Computing (SHOC) benchmark suite to evaluate our test platforms. Table II provides a brief description of each benchmark test. The platforms were programmed using OpenCL v1.1 with AMD Stream SDK v2.3 and AMD Catalyst graphics driver v10.7.

B. PCIe Bandwidth Tests

We used the OpenCL PCIeBandwidth test to understand the cost of data transfers on discrete GPUs as well as within the AMD Fusion architecture and to identify when and where the Fusion architecture performs better. Fig. 7 presents the data transfer bandwidth for varying data sizes, ranging from 1 KB to 64 MB on the three test platforms. (We ran 10,000 iterations of the test for each data transfer size.)

Fig. 7a shows the bandwidths obtained when data is transferred from the 'host' (CPU) to 'device' (GPU). Though the APU eliminates PCIe access, the bandwidth that it achieves for smaller data sizes is less than that obtained on the discrete GPU platform. Only after a certain threshold does the benefit of eliminating PCIe become evident and the AMD Zacate APU starts to outperform its discrete brethren. Fig. 7b shows similar results for the reverse case, i.e., when data is transferred from the 'device' (GPU) to 'host' (CPU). However, for 'device to host' transfers, the threshold beyond which the Zacate APU achieves greater bandwidth is lower than that for the 'host to device' transfers. Further, the 'host to device' bandwidth is higher than 'device to host' bandwidth for the discrete GPU, while the opposite is true for Zacate APU.

Thus, the ideology that the Fusion architecture would overcome the PCIe bottleneck in a discrete GPU platform is not always true. The bandwidth obtained on the APU increases

TABLE I
TEST PLATFORMS

Platform	AMD Zacate APU	AMD Radeon HD 5870	AMD Radeon HD 5450
Stream Processors	80	1600	80
Compute Units	2	20	2
Memory Bus Type	NA	GDDR5	DDR3
Device Memory	192 MB	1024 MB	512 MB
Local Memory	32 KB	32 KB	32 KB
Max. Workgroup Size	256 Threads	256 Threads	128 Threads
Peak Core Clock Freq.	492 MHz	850 MHz	675 MHz
Peak FLOPS	80 GFlop/s	2720 GFlop/s	104 GFlop/s
Host:			
Processor	AMD Engg. Sample @1.6 GHz	Intel Xeon E5405 @2.0 GHz	Intel Celeron 430 @1.8 GHz
System Memory	2 GB (NA)	2 GB DDR2	2 GB DDR2
Cache	L1: 32K, L2: 512K	L1: 32K, L2: 6M	L1: 32K, L2: 512K
Kernel	Ubuntu 2.6.35.22	Ubuntu 2.6.28.19	Ubuntu 2.6.32.24

TABLE II
BENCHMARK SUITE

Benchmark	Description
Bandwidth Test	Measures the bandwidth of PCIe interconnect between host processor and the discrete GPU as well as bandwidth of transfers between the two divisions of system memory for the APU. It does so by repeatedly transferring data of various sizes (1KB to 64 MB) to and from the device.
FFT	Measures the performance of a two dimensional Fast Fourier Transform. The benchmark computes multiple FFTs of size 512 in parallel. The FFT implementation is based on the algorithm described by Volkov and Kazian [11].
MD	Measures the performance of pairwise calculation of the Lennard-Jones potential from molecular dynamics. Each thread computes the acceleration for one particle based on the potential field generated by all particles into a cutoff area. It uses a neighbor-list algorithm as LAMMPS [12].
Scan	Measures the performance of the parallel prefix sum algorithm (also known as Scan) on a large array of floating point data.
Reduction	Measures the performance of a sum reduction operation using floating point data. The kernel first performs a partial reduction on the global data, storing the results in local memory. Next, a reduction is computed over the local data array and the result is stored in global memory.

with the increase in data-transfer size, which suggests that there is some cost incurred to carry out the data transfer, e.g., DMA set-up and buffer pinning, and this cost of transferring data gets amortized with the increase in data-transfer size. From our results, one can infer that the APU does not provide the promised benefit of overcoming PCIe costs for small data transfers, but it begins to show promise for larger data sizes. Moreover, it is difficult to know this crossover point since it may vary from application to application (and since the Zacate APU is largely a "black box" to the end user). However, this problem of deducing the threshold data size is only temporary as the next generation of APUs from AMD are expected to merge the two memory partitions.

C. Kernel Benchmarks

We demonstrate the efficacy of the Fusion architecture via four application benchmarks (MD, FFT, Scan, and Reduction) from the SHOC benchmark suite.

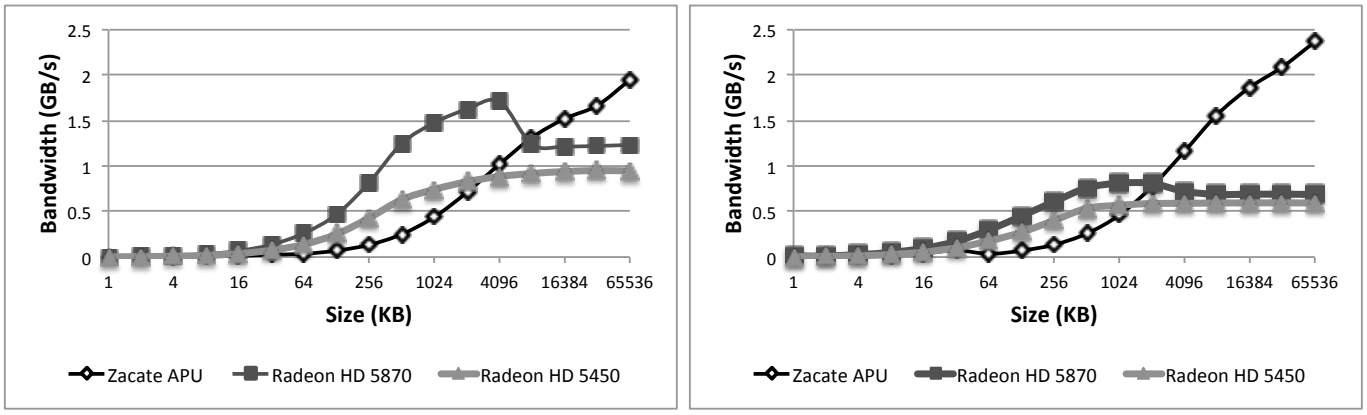
Fig. 8 and Fig. 9 present the data transfer and kernel execution times for each application benchmark with varying problem sizes on the three accelerator platforms: (i) AMD Zacate APU, (ii) discrete AMD Radeon HD 5450, and (iii) discrete AMD Radeon HD 5870. (We ran each test 200 times and report the average.)

Since the GPU present in the Zacate APU is similar to the discrete AMD Radeon HD 5450, one would expect similar kernel execution times on these two platforms, however, it is not the case. It was not possible to run two applications, i.e., scan and reduction, on the low-power discrete 5450 GPU because of its upper limit of 128 work-items in a work-group while the SHOC benchmark suite requires at least 256 work-items to run. Therefore, we do not present results for these benchmarks on the discrete AMD Radeon HD 5450 GPU.

Molecular Dynamics (MD): Fig. 8a shows the data-transfer times and kernel execution times for a molecular dynamics (MD) benchmark. We see that the APU reduces the data transfer times for all problem sizes, and the effectiveness of the APU increases with an increase in problem size, which is in-line with the bandwidth numbers obtained by the OpenCL PCIeBandwidth test, shown in Fig. 7.

As expected, the kernel executes fastest on the discrete AMD 5870 because of the presence of substantially faster and greater number of GPU processing cores. However, between the AMD Radeon HD 5450 and the APU, the kernel surprisngly executes faster on the APU than on the Radeon HD 5450.

By analyzing the execution profile of MD, we found out that it is a *compute-bound* application. The current core-clock frequencies of the APU and the AMD Radeon HD 5450 were found to be 278 MHz and 157 MHz, respectively. Hence, the APU is clocked *1.77-fold* faster than the discrete GPU, which is the likely reason for faster kernel execution on the APU. To



(a) Host to Device

(b) Device to Host

Fig. 7. PCIe Bandwidth Test.

corroborate this fact, we report the Gflop/s obtained for MD on these two platforms in Table III. Although the difference in the Gflops/s obtained is not 1.77-fold (difference in core-clock frequency), the presence of slower processing cores on the discrete AMD 5450 indicates that it is the reason for the slower kernel execution on the AMD Radeon HD 5450.

TABLE III
PERFORMANCE OF MD IN GFLOP/S.

Problem Size	MD		
	APU	5450	Difference Factor
1	1.63	1.39	1.17×
2	1.90	1.44	1.32×
3	1.90	1.45	1.31×
4	1.91	1.44	1.33×

Fast Fourier Transform (FFT): Fig. 8b shows the data-transfer times and kernel-execution times for a 1-D FFT transformation on all three accelerator platforms. Like the previous MD benchmark, the APU assists in reducing data-transfer times for FFT and enables a greater amount of data to be transferred, resulting in a larger perceived benefit for the APU. As expected, the kernel execution time is minimal on the powerful discrete AMD 5870, when compared to other platforms. However, the kernel execution time on the APU is worse than on the AMD Radeon HD 5450. As FFT transformations are known to be *memory bound* [13], [14], analyzing the device memory bandwidth on the APU as well as the AMD Radeon HD 5450 would provide us with a concrete reason for slower kernel execution on the APU.

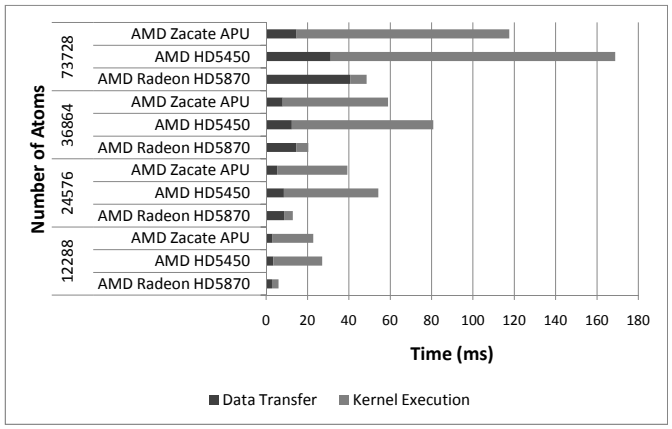
To this end, using the DeviceMemory benchmark from the SHOC Benchmark Suite, we computed the global memory bandwidth on these two platforms and found that global memory bandwidth on the AMD Radeon HD 5450 is roughly *two times better* than on the Zacate APU. We also measured the performance (GFlop/s) of FFT on the two platforms, which are presented in Table IV. This table shows that the performance achieved on the AMD Radeon HD 5450 is twice as much as that on the APU, and hence, we infer that the better global memory bandwidth leads to faster kernel execution on the discrete AMD Radeon HD 5450.

TABLE IV
PERFORMANCE OF FFT IN GFLOP/S.

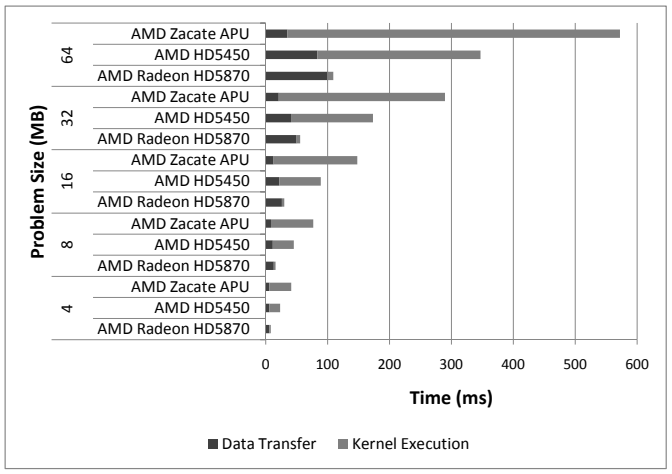
Problem Size	FFT		
	APU	5450	Difference Factor
1	0.68	1.36	2.00×
2	0.69	1.40	2.03×
3	0.70	1.40	2.00×
4	0.70	1.42	2.03×
5	0.70	1.43	2.04×

Scan and Reduction: Fig. 9a and Fig. 9b present the data-transfer times and kernel execution times for *scan* and *reduce*, respectively. The total execution time for scan is approximately the same for the high-powered AMD Radeon HD 5870 GPU and laptop-powered AMD Zacate APU. This empirical result is a stunning one given that the AMD Radeon HD 5870 not only has faster GPU cores, but it also has 20 times more of them, when compared to the APU. Though kernel execution is slower on the APU, data transfers are faster, thus offsetting the effect of the slower APU processing cores. For the reduce benchmark, the total execution time is actually better on the 'less powerful' AMD Zacate APU than on the discrete and 'more powerful' AMD Radeon HD 5870. The elimination of PCIe in data transfer allows the fused GPUs in the Zacate APU, with less than $\frac{1}{20}$ of the computational capability of an AMD Radeon HD 5870, to deliver *3x better performance*.

As presented in Section III, the speedup S', due to parallelization of an application on a GPU with infinite processing cores, is a function of the parallel overhead, which is the time spent in transferring data between the CPU and the GPU. Table V summarizes the savings in data transfer times due to the elimination of PCIe. The benefit is computed as the ratio between the data transfer time on a discrete GPU and the APU. For each application, the *median* of savings for all problem sizes is computed. From the table, it is evident that the AMD Fusion architecture is certainly successful in overcoming the

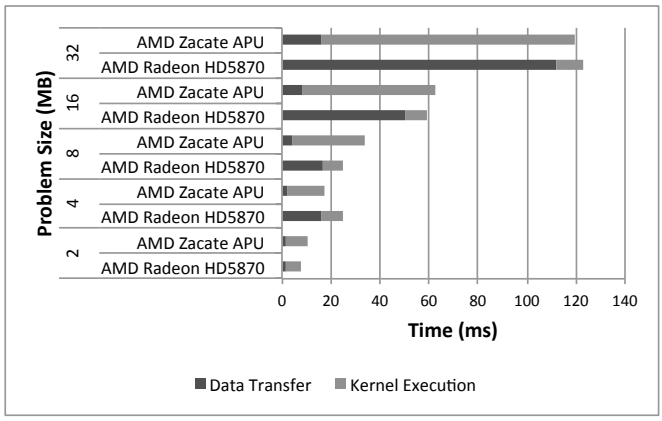


(a) MD

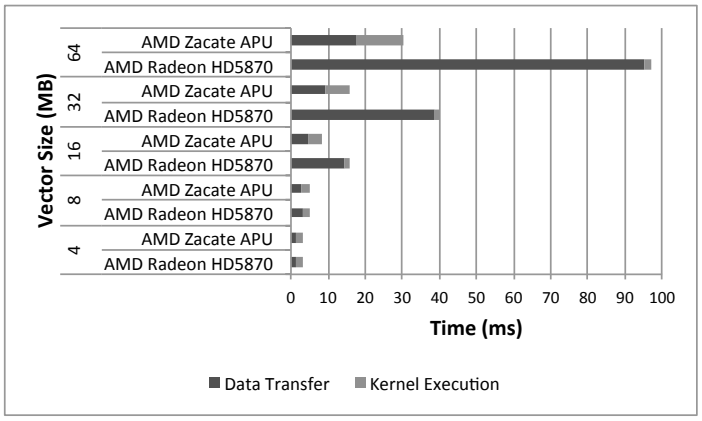


(b) FFT

Fig. 8. Application Performance (MD and FFT).



(a) Scan



(b) Reduction

Fig. 9. Application Performance (Scan and Reduction).

TABLE V
BENEFIT DUE TO ELIMINATION OF THE PCIE.

Application	Benefit Over AMD 5870	Benefit Over AMD 5450
MD	1.7×	1.6×
FFT	2.0×	1.7×
Scan	6.1×	N/A
Reduction	3.1×	N/A

PCIe bottleneck that the discrete GPU suffers from.

However, we must note that the speedup, S', is also a function of p', the accelerated parallel fraction of the program. Moreover, p' can vary significantly depending on the number of compute units and available memory bandwidth on the GPU, as well as the arithmetic intensity and memory access pattern of the algorithm. While o can be reduced greatly by using AMD Fusion to alleviate the data transfer costs between CPU and GPU memory, there is no guarantee that p' will remain unchanged from the discrete GPU. It is an important and challenging problem to understand the performance variation of p' among the fused CPU+GPU and discrete GPU systems, which we have identified as our future work.

V. RELATED WORK

Due to the newness of the AMD Fusion architecture, not much has been discussed in literature about this particular architecture. However, substantial research has been conducted to evaluate the performance of GPUs. For example, Ryoo et al. have carried out an optimization study and have evaluated the performance of discrete NVIDIA GPUs [15]–[17]. The authors have demonstrated that while GPUs hold the promise of delivering substantial benefits, one needs to be diligent in optimizing GPU applications. Jang et al. and Xudong et al. have proposed optimization strategies for AMD GPUs using the Brook+ programming language and have evaluated them by implementing a matrix multiplication kernel [18] and a multi-grid application [19] for solving PDEs, respectively.

Micro-benchmarks have been extensively used to reveal the architectural details of the discrete GPUs. For example, Volkov et al. have exploited GPUs to tune dense linear algebra and have developed detailed benchmarks to reveal kernel bottlenecks like shared memory access patterns and kernel launch overhead [20]. Wong et al. have also used micro-benchmarks to understand the micro-architecture of the NVIDIA GT200 GPU [21]. Both of them used *decuda*, which is a disassembler

for NVIDIA's machine level instructions, to understand the mapping of various instructions on the GPU [22].

Recently, a technology called *GPUDirect* [23] has been developed, which aims to minimize the cost of data transfers between GPUs in a multi-GPU environment. However, it plays no role in addressing the PCIe bottlenecks that arise due to data transfers between CPU and the GPU.

VI. Conclusions and Future Work

GPUs have proven to be quite beneficial in accelerating many scientific applications. However, quite often, the performance of GPU applications is thwarted by slow PCIe transfers between the CPU and GPU. Novel architectures like AMD Fusion, which eliminate PCIe accesses to and from the GPU, hold the promise of overcoming these bottlenecks, and in turn, improving application performance.

In this paper, we have empirically demonstrated the effectiveness of the Fusion architecture, where PCIe data transfer costs are replaced by simple fast memory block transfers between the x86 and SIMD memory partitions. We show that for some applications, these costs could be reduced by as much as *six-fold*. However, in the case of smaller data-transfer sizes, we found that the data-transfer cost for AMD Fusion was actually worse than its discrete brethren. This is a problem that points to the likelihood that expensive underlying protocol mechanisms and policies create overhead costs that cannot be amortized until a certain data-size threshold is reached. Future APU architectures are expected to have these memories seamlessly merged, which means that there will not be a need to transfer data to and from the GPU memory at all, with the overall application performance only getting better.

Finally, we showed that the application performance on AMD Fusion is quite robust. In particular, for one benchmark, i.e., reduction, Fusion delivers a *three-fold* improvement in application performance when compared to a discrete GPU that is 20 times computationally more powerful.

Acknowledgment

This work is supported in part by NSF grants IIP-0804155 and CNS-0916719 and an AMD Research Faculty Fellowship.

References

[1] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream Computing on Graphics Hardware," in *International Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 2004, pp. 777–786.

[2] "The Top500 Supercomputer Sites," http://www.top500.org.

[3] R. Vuduc, A. Chandramowlishwaran, J. W. Choi, M. E. Guney, and A. Shringarpure, "On the Limits of GPU Acceleration," in *Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar)*, Berkeley, CA, USA, June 2010.

[4] R. Bordawekar, U. Bondhugula, and R. Rao, "Can CPUs Match GPUs on Performance with Productivity?: Experiences with Optimizing a FLOP-intensive Application on CPUs and GPU," in *Research Report RC25033, IBM T.J. Watson Research Center*.

[5] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 451–460. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816021

[6] 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*, ser. GPGPU '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1735688.1735702

[7] AMD, "AMD Fusion Family of APUs: Enabling a Superior, Immersive PC Experience," 2010, http://www.amd.com/us/Documents/48423_fusion_whitepaper_WEB.pdf.

[8] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[9] M. Hill and M. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33 –38, 2008.

[10] AMD, "APU 101: All about AMD Fusion Accelerated Processing Units," 2011, http://developer.amd.com/assets/apu101.pdf.

[11] V. Volkov and B. Kazian, "Fitting FFT onto the G80 Architecture," *University of California Berkeley*, p. 6, 2008. [Online]. Available: http://www.cs.berkeley.edu/~kubitron/courses/cs258-S08/projects/reports/project6_report.pdf

[12] "LAMMPS Molecular Dynamics Simulator," http://lammps.sandia.gov/index.html.

[13] A. Nukada and S. Matsuoka, "Auto-tuning 3-D FFT Library for CUDA GPUs," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 30:1–30:10. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654090

[14] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354?origin=crossref

[15] S. Ryoo, C. I. Rodrigues, S. S. Stone, J. A. Stratton, S.-Z. Ueng, S. S. Baghsorkhi, and W. mei W. Hwu, "Program Optimization Carving for GPU Computing," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1389 – 1401, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/B6WKJ-4SVC5M7-2/2/55e6c6806c3260e1e89aea930c915d19

[16] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, and W.-m. W. Hwu, "Program Optimization Space Pruning for a Multithreaded GPU," in *CGO '08: Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*. New York, NY, USA: ACM, 2008, pp. 195–204.

[17] S. Ryoo, C. Rodrigues, S. Stone, S. Baghsorkhi, S.-Z. Ueng, and W. mei Hwu, "Program Optimization Study on a 128-Core GPU," in *Workshop on General Purpose Processing on Graphics Processing*, 2008.

[18] B. Jang, S. Do, H. Pien, and D. Kaeli, "Architecture-aware Optimization Targeting Multithreaded Stream Computing," in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, ser. GPGPU-2. New York, NY, USA: ACM, 2009, pp. 62–70. [Online]. Available: http://doi.acm.org/10.1145/1513895.1513903

[19] F. Xudong, T. Yuhua, W. Guibin, T. Tao, and Z. Ying, "Optimizing Stencil Application on Multi-thread GPU Architecture Using Stream Programming Model," in *Architecture of Computing Systems - ARCS 2010*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 5974, pp. 234–245. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11950-7_21

[20] V. Volkov and J. Demmel, "Benchmarking GPUs to Tune Dense Linear Algebra," in *Proc. of the 2008 ACM/IEEE Conference on Supercomputing*, November 2008.

[21] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU Microarchitecture through Microbenchmarking," in *ISPASS '10: Proceedings of the 37th IEEE International Symposium on Performance Analysis of Systems and Software*, 2010.

[22] W.J. van der Laan, "Decuda," 2008, http://wiki.github.com/laanwj/decuda.

[23] Mellanox and NVIDIA, "NVIDIA GPUDirect Technology – Accelerating GPU-based Systems," 2010, http://www.mellanox.com/pdf/whitepapers/TB_GPU_Direct.pdf.