The Tradeoffs of Fused Memory Hierarchies in Heterogeneous Computing Architectures

Kyle Spafford Jeremy S. Meredith Seyong Lee
Dong Li Philip C. Roth Jeffrey S. Vetter

Future Technologies Group
Computer Science and Mathematics Division
Oak Ridge National Laboratory
1 Bethel Valley Road-MS6173
Oak Ridge, TN 37831
{*spaffordkl, jsmeredith, lees2, lid1, rothpc, vetter*}*@ornl.gov*

ABSTRACT

With the rise of general purpose computing on graphics processing units (GPGPU), the influence from consumer markets can now be seen across the spectrum of computer architectures. In fact, many of the high-ranking Top500 HPC systems now include these accelerators. Traditionally, GPUs have connected to the CPU via the PCIe bus, which has proved to be a significant bottleneck for scalable scientific applications. Now, a trend toward tighter integration between CPU and GPU has removed this bottleneck and unified the memory hierarchy for both CPU and GPU cores. We examine the impact of this trend for high performance scientific computing by investigating AMD's new Fusion Accelerated Processing Unit (APU) as a testbed. In particular, we evaluate the tradeoffs in performance, power consumption, and programmability when comparing this unified memory hierarchy with similar, but discrete GPUs.

Categories and Subject Descriptors

B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids; C.1.3 [**Processor Architectures**]: Other Architecture Syltes—*heterogeneous (hybrid) systems*

General Terms

Performance, Measurement

1. INTRODUCTION

1.1 GPUs and Heterogeneity

The demand for flexibility in advanced computer graphics has caused the GPU to evolve from a highly specialized, fixed-function pipeline to a more general processor. However, in its current form, there are still substantial differences between the GPU and a traditional multi-core CPU.

Perhaps the most salient difference is in the memory hierarchy: the GPU shuns high capacity, coherent caches in favor of a much larger number of functional units. This lack of coherent caches in the GPU is not surprising, given the low reuse of graphics data flowing through the frame buffer. Instead, GPUs have used wide memory busses and specialized texturing hardware (that provides a limited set of addressing and interpolation operations) for a high bandwidth, high latency connection to off-chip RAM. Because of these architectural differences, early GPGPU adopters observed that many data-parallel problems in scientific computing exhibited substantial performance improvements when run on a GPU.

Initially, achieving such speedups usually required that all operations be cast as graphics operations, comprising a "heroic" programming effort. This made the initial costs of GPGPU too high for mainstream scientific computing. However, with the advent of programming models like OpenCL and CUDA, which are very similar to standard C/C++, the barriers to entry have decreased and GPGPU now enjoys much wider adoption. Indeed, the low cost of the GPU hardware itself is one of GPGPU's main advantages.

Despite these improvements, most GPUs suffer from performance limitations due to the PCIe bus, and limited productivity to due an increasingly complex memory model. Both of these problems are direct consequences of the hardware architecture. Simply put, any data that moves between the CPU and a discrete GPU must traverse the PCIe bus, which has limited bandwidth (usually at least an order of magnitude less than GPU memory bandwidth). This architecture results in relatively slow transfers, and high latency synchronization between devices that applications should avoid when possible.

The complexity of the memory model for programming accelerated applications has also increased. The OpenCL memory model for a single GPU is already more complicated than the cache hierarchy of a traditional multicore. In fact, the OpenCL memory model contains five distinct memory spaces (global, local, constant, image, and private), each with its own coherency policy and optimal access pattern. Moreover, many of the address spaces only implement relaxed consistency, requiring the programmer to perform explicit synchronization. This model is further complicated by the PCIe bus, since the programmer is required to keep

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CF'12, May 15–17, 2012, Cagliari, Italy.
Copyright 2012 ACM 978-1-4503-1215-8/12/05 ...$10.00.

CPU and GPU memory consistent via explicit DMA transfers.

In an effort to address these difficulties, system architects and vendors are now focusing on designs which feature much tighter integration between the CPU and GPU, such as AMD's Fusion [4] (studied in this paper) and NVIDIA's Project Denver.

1.2 SoC and Tighter Integration

Another consumer trend that has motivated the design of the APU is the shift towards a system-on-a-chip (SoC). SoC design largely came about in the mobile and embedded spaces due to the desire for reuse of silicon designs, specifically reusing basic system blocks for wireless technologies, specialized media processing units, etc. Tighter integration also offers advantages in energy efficiency by enabling fine-grained dynamic voltage and frequency scaling (DVFS) across multiple system components. In DVFS, the clock speed and voltage of a processing element are raised or lowered by the operating system based on processor utilization and workload, resulting in higher performance under load and lower power consumption when cores are idle. This improved efficiency is increasingly appealing for HPC systems, due to the projections for the power requirements of an exascale computer [12], and the possibility of leveraging design trends in in the mobile and embedded markets, where the focus is on longer battery life.

1.3 Tradeoffs

Heterogeneity and SoC-like integration are both evident in the design of AMD's Fusion APU, shown in Figure 1; it replaces the PCIe connection between the CPU and GPU cores with a unified north bridge and two new busses, the Radeon Memory Bus (RMB) and the Fusion Compute Link (FCL), discussed further in Section 2.1. While integrated GPUs have existed for some time in the mobile market, AMD's fused GPU (fGPU) can snoop CPU cache transactions using the FCL, making Fusion the first mass-market architecture to support cache coherency between the CPU and GPU. This capability for cache coherency is the hallmark of a fused heterogeneous architecture.

Fusing these two distinct memory hierarchies results in numerous tradeoffs. For example, traditional GPU architectures support much higher memory bandwidth due to dedicated GDDR memory (see Section 4) than the Llano Fusion architecture. This paper explores five such tradeoffs using Llano as a forward-looking example of tightly-integrated CPU and GPU architectures:

1. In the multi-level cache hierarchy of fused designs, what is the set of caches that should be kept coherent in order to allow scalability to large core counts?

2. With fixed die space and transistor count, how should resources be allocated to improve serial performance (CPU cores) as opposed to parallel performance (GPU cores)?

3. Given a fixed power budget, should the memory be configured for higher capacity (e.g. DDR3) or higher bandwidth (e.g. GDDR5)?

4. Are fused designs more power efficient? Would power be better spent using discrete, specialized components?

5. Given a limited amount of programmer effort, what is the correct level of abstraction to use when programming APUs? Simple abstractions will require advanced runtime systems and middleware, while lower-level abstractions require more application programming effort.

2. CACHE COHERENCY VS. SCALABILITY

The specialization of the GPU memory hierarchy is also evident in its cache configuration, which has traditionally relied on a combination of simple, incoherent SRAM scratchpad memories with a specialized texturing unit. This texturing unit typically contains its own separate cache, targeted to regular access patterns on two dimensional data. It also implements a limited set of functions in hardware including several addressing computations and interpolation operations that are ubiquitous in graphics applications. Aside from the texture unit, the cache hierarchy of the GPU is remarkably flat compared to the CPU, largely due to the data parallelism of graphics operations. One of the benefits of this flat hierarchy is that it scales very well with the number of processor tiles (e.g. to sixteen hundred cores in Cypress, with only thirty-two kilobytes of scratchpad memory per tile). The disadvantage is that this hierarchy can only support relaxed consistency models for small groups of cores, which places a burden on the programmer which will be discussed more in Section 6.

On the other hand, CPUs have traditionally used multi-level, high capacity caches which are kept consistent using protocols like MESI. Such protocols provide strong consistency models for the programmer, but are much less scalable due to the rapid growth of coherency-related traffic. The key tradeoff, then, is in determining how the CPU and fGPU caches can be coupled to enforce coherency while preserving scalability to a large number of cores.

2.1 Fusion Memory Hierarchy

In Fusion, this coupling is accomplished via the addition of two new busses, the Fusion Compute Link (FCL) and the Radeon Memory Bus (RMB), depicted in Figure 1 as well as a unified north bridge which coordinates access to the different logical regions of physical memory. One of the goals of the Fusion memory hierarchy is to allow the CPU and GPU to share data while preserving performance for each processing element's predominant access pattern—the CPU should still support low latency access (optimized with caches) and the GPU should still have high bandwidth access (optimized via the RMB) to contiguous regions of memory. In order to understand the design at a high level, it is important to understand each of the components:

- **Physical Memory**. In Fusion, the same physical memory is shared between the CPU and GPU. The operating system maintains the partition between system memory (normal, pageable memory) and "local" memory, which is reserved for the GPU. Local memory is conceptually similar to the frame buffer or the onboard RAM on a discrete GPU.

- **Traditional CPU Cache Hierarchy**. The CPU cores are supplied with a standard cache hierarchy including private L1 data and instruction caches, and a 1MB private L2 cache (4MB total L2 capacity).

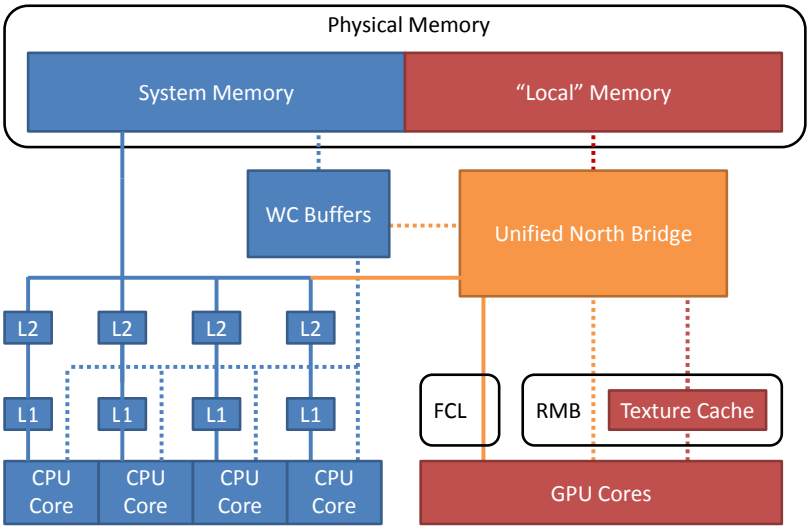

Figure 1: The Fusion Memory Hierarchy. The solid lines in this figure indicate cache coherent connections, and the dashed lines show lack of coherence. Blue indicates components of a traditional CPU memory hierarchy and red shows components of a traditional GPU hierarchy. For example, the CPU usually accesses System Memory through the L2 cache and the write-combining buffers. Orange indicates novel features and paths in Fusion. The familiar cache hierarchy of the CPUs is connected to the GPU cores by the FCL. The RMB preserves high bandwidth access from the GPU cores to the "Local" memory (optionally storing data in the texture cache). The CPU cores can access this same "Local" memory via the write-combining buffers through the Unified North Bridge.

- **Write Combining Buffers**. Each CPU core in Llano has four uncached, write-combining buffers. Ideally, these buffers can provide relatively high write bandwidth (by merging memory transactions) but are typically avoided due to a lack of strict ordering and extremely high read latency. However, in Fusion, the WCBs are utilized when the CPU needs to write into "local" memory. In this case, they exploit the higher write bandwidth and avoid polluting the CPU cache with data that will be primarily used on the GPU cores.

- **Fusion Compute Link**. The FCL provides a high latency, low bandwidth from the GPU cores to the CPU cache. It's arbitrated by the UNB, and has the capability to snoop CPU cache transactions, providing for full coherency between the CPU and GPU. Due to its low bandwidth (compared to other memory paths in the system), it should primarily be used for fine-grained data sharing between the CPU and GPU.

- **Radeon Memory Bus**. The RMB is a much wider bus that connects the GPU cores to the "local" partition of physical memory and mimics the performance of RAM access in a discrete GPU–high latency and high bandwidth. It bypasses all cache (except L1 and texture), and can saturate DRAM bandwidth.

Simply put, CPU-like accesses are supported by traditional caches, GPU access patterns are handled by the RMB, and the FCL enables cache coherency only when it is needed, with most of the pathways enabled through the address translation capabilities of the UNB.

3. LATENCY OPTIMIZED VS. THROUGHPUT OPTIMIZED CORES

Another distinction between the traditional CPU core and the GPU core is the marked difference in the allocation of transistors and die space. In CPUs, a substantial portion of these resources have been devoted to optimizing latency in single-threaded programs using caches, complex techniques for instruction-level parallelism including out-of-order execution, and other specialized units like branch predictors. In comparison to CPUs, GPUs have throughput-optimized cores that are much simpler, but GPUs tend to have far more cores. In addition, GPUs can schedule thousands of application threads onto these cores in order to hide memory access latency. For example, contemporary GPUs (e.g. Cypress and Llano's core counts) can have ten to one hundred times as many cores as CPUs. In these throughput-oriented designs, performance is achieved through massive, fine-grained parallelism, and, particularly for HPC, a large number of floating-point units.

The question then, is how many resources should be devoted to serial performance (latency-optimized) compared to parallel performance (throughput-optimized). For instance, are wide SIMD units on the CPU cores still beneficial? Would those resources be better spent on more GPU cores, which would handle the majority of the floating point workload? This complex tradeoff has serious implications for sustained performance and depends on application characteristics like the fraction of parallel work, the instruction mix, and the requirements for synchronization.

It is difficult to characterize this tradeoff in a sense that will generalize well to any application. However, the amount of serial performance sacrificed to obtain parallel perfor-

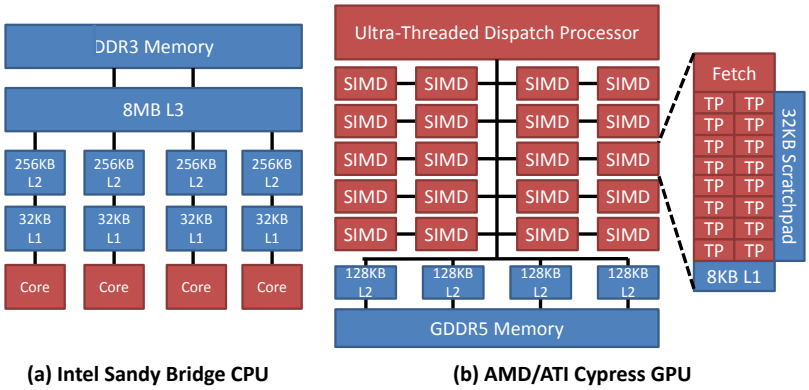

Figure 2: Block Diagram of Core i7-2600 (Sandy Bridge) and Cypress Architectures. Blue elements indicate memory or cache while red indicates processing elements. The dotted lines show that each SIMD engine in Cypress contains sixteen thread processors, each with five ALUs, a shared L1, and a shared scratchpad memory.

	Cores	Clock	Peak FLOPS	RAM	Bus Width	Mem. Clock	Bandwidth	TDP
Units	#	Mhz	GFLOPS (SP/DP)	GB	Bits	Mhz	GB/s	Watts
Core i7-2600	4	3400	108.8/54.4	8	128	1333	21	95
Llano's CPU Llano's fGPU	4 400	2900 600	46.4/23.2 480/0	8	128	1866	29.9	100
Redwood	400	775	620/0	0.5	128	1000	64	64
Cypress	1600	825	2640/528	2	256	1150	147.2	225

Table 1: Architecture Specifications. Note that Llano's CPU and fGPU cores share the same interface to physical memory and have a combined TDP of 100W.

mance can be quantified by comparing an APU to a traditional CPU. This loss can then be weighed against increased parallel performance on an application-specific basis.

3.1 Experimental Platform

We compare a current APU architecture to a traditional CPU and discrete GPU can help illustrate some of the tradeoffs of fused designs. Specifically, we evaluate the concrete examples (shown in Figure 2) of AMD's A8-3850 Llano APU (with fGPU) and two discrete GPUs: the ATI Radeon HD5670 (codenamed Redwood), most similar to the fGPU of Llano, and the high-end ATI FirePro v8800 (Cypress). Also, any power or performance measurements involving the Redwood GPU use the Llano as a host system with the fGPU disabled to ensure consistency. On some occasions, we also compare Llano to an Intel Core i7-2600 CPU (Sandy Bridge architecture). This CPU is similar in terms of core count and power envelope, but devotes far fewer resources to graphics operations. Its integrated graphics hardware does not support OpenCL and is not used in any of our tests. Table 1 gives a more detailed listing of the differences among Llano, Redwood, Cypress, and Sandy Bridge. None of the test configurations used ECC memory, which is not supported by these consumer versions of the architectures. For further introduction to GPU architectures, in general, we refer the reader to the overview from Owens [19] and the discussion of the AMD architecture by Daga et al [8].

Unless otherwise specified, all measurements were taken using the AMD APP SDK v2.4, SHOC [9] v1.1.2, GCC v4.4.5, and Scientific Linux v6.1.

3.2 CPU Performance

To evaluate Llano CPU performance, we compared the performance of a simple dense matrix-matrix multiply operation with another contemporary processor with similar power envelope, the Intel Core i7-2600 CPU (with Sandy Bridge microarchitecture). This benchmark is intended to show the processor's practical upper limit for computation rate. We also compared the performance of the High Performance Computing Challenge (HPCC) [10] benchmark suite on both processors. Because this benchmark suite measures the performance of several types of operations, it provides a more complete picture of CPU performance than the simple matrix-matrix multiply benchmark. The specifications of the systems used in this comparison are shown in Table 1. On the Sandy Bridge system, HyperThreading was disabled to avoid sharing floating point hardware between multiple threads within each core; thus, the Sandy Bridge test system supported four hardware threads total, like the Llano system. Turbo mode was enabled on the Sandy Bridge system, allowing the processor to increase the clock rate of a single core if it was performing a computationally intensive task while other cores on the chip were less busy. Also, the software stack for this system differs slightly from the listing in Section 3.1, with the Sandy Bridge tests using the Intel compiler v2011 SP1.8.274, MPI v4.0.3, and MKL math library v10.3.8 while the Llano was tested using the PGI compiler v11.10, ACML v4.4 and v5.0, and OpenMPI v1.5.4.

Llano and Sandy Bridge performance for the SGEMM and DGEMM benchmarks are shown in Figures 3 and 4,

for several threading configurations and a range of matrix sizes. For DGEMM, the Core i7-2600 consistently outperforms the Llano CPU, more than doubling its peformance on the largest matrix. No results are shown from the Llano fGPU since it does not support double precision.

In contrast, the Llano's fGPU outperforms the Core i7-2600 on the SGEMM benchmark on large matrices, even when four threads are used. Note that the Llano benchmark used AMD's APPML math library v1.6, which is OpenCL-based.

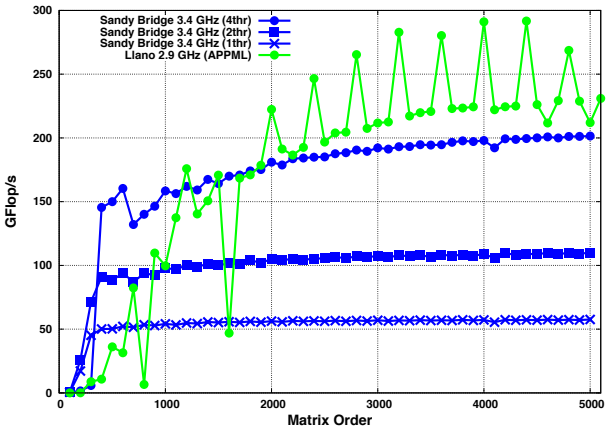

Figure 3: SGEMM Performance (one, two, and four CPU threads for Sandy Bridge and the OpenCL-based AMD APPML for Llano's fGPU)

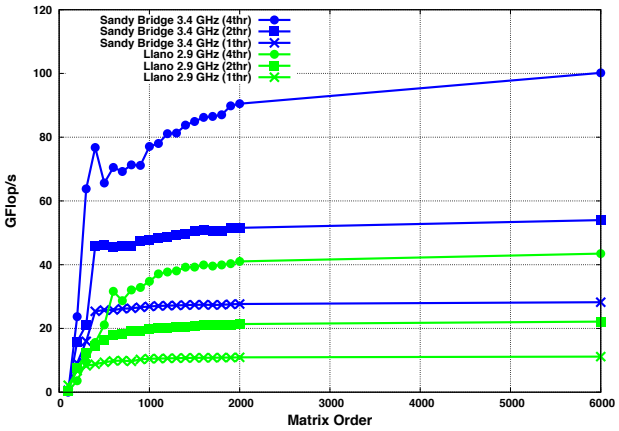

Figure 4: DGEMM CPU Performance for one, two, and four threads

Llano and Core i7-2600 performance on the HPCC benchmark suite is summarized in Table 2. The measurements from the table were obtained using four MPI processes (one per core) on each system. On the AMD system, we built HPCC with both the ACML 4.4 and 5.0 math libraries, and we tried explicitly setting the number of OpenMP threads to be used by the ACML library. We found slightly better performance from using ACML 4.4 on this system for the HPL and DGEMM subtests, without specifying the number of OpenMP threads, so the numbers presented in Table 2 reflect this scenario. On the Intel system, we ran HPCC under several scenarios. We tried pinning the each of the four HPCC processes to a specific processor core, and we tried explicitly setting the number of OpenMP threads to be used by the MKL math library. With these single node tests, performance did not vary much under these different scenarios, but we did notice slightly better performance for the HPL and DGEMM subtests when pinning processes but not explicitly setting the number of OpenMP threads. Hence the Sandy Bridge numbers presented in Table 2 are for this scenario.

Benchmark	Units	AMD A8-3850	Intel i7-2600
HPL	TFlop/s	0.0290	0.0949
SingleDGEMM	GFlop/s	11.1037	24.7620
StarDGEMM	GFlop/s	10.9816	25.8956
PTRANS	GB/s	0.4093	2.3594
SingleRandomAccess	GUP/s	0.0363	0.0671
StarRandomAccess	GUP/s	0.0165	0.0253
MPIRandomAccess	GUP/s	0.0490	0.0912
SingleSTREAM_Triad	GB/s	5.4585	13.3438
StarSTREAM_Triad	GB/s	1.9530	3.5079
SingleFFT	GFlop/s	0.9810	2.8007
StarFFT	GFlop/s	0.6822	1.6154
MPIFFT	GFlop/s	1.6292	3.5563

Table 2: HPCC Performance

Llano's design reflects the perspective that the fGPU should be the computational workhorse, thus the CPU microarchitecture is relatively simple, and the performance penalty for not following this distribution of work is substantial. In contrast, the Core i7-2600 design reflects the perspective that the CPU is the primary computational device in the system. These differing perspectives are evidenced in the HPCC measurements, where the Core i7-2600 outperformed the A8-3850 by a margin larger than would be expected based on clock speed differences alone. Interestingly, the gap was largest for the computationally intensive subtests, but narrowed significantly for subtests like RandomAccess that are more focused on memory hierarchy performance.

4. CAPACITY VS. BANDWIDTH

The next tradeoff is the type of physical memory used for the base of the fused memory hierarchy. Traditionally, CPUs have attempted to optimize latency to high capacity memory, like the DDR3 used in the Llano APU. Conversely, GPUs, which have traditionally been concerned with repeatedly streaming a fixed-size frame buffer, focus on achieving maximum bandwidth. In order to achieve this bandwidth, GDDR (*graphics* double data rate) standards began to diverge from their DDR counterparts to place an emphasis on wider memory busses and higher effective clock speeds. However, given a similar power budget, GDDR tends to have much less capacity than DDR. In a discrete architecture, each type of core can be paired with the most applicable memory, but in a fused architecture, the latency and throughput-oriented cores must use the same type of physical memory. In the near term, this is essentially a choice between DDR and GDDR memory. In the future, however,

this tradeoff is likely to be become much more complex as the configuration of 3D stacked memories and advanced memory controllers may allow for increased flexibility in combining different types of memories in one node.

In our concrete examples, both the Redwood and Cypress discrete GPUs use GDDR5, while Llano's fGPU shares DDR3 with the CPU cores. As shown in Table 1, the gap in capacity is quite large, but the most salient difference is bandwidth. This is in large part due to the fact that GDDR5 is quad-pumped while DDR3 is double-pumped. That is, DDR3 delivers two bits of data per signal line per clock cycle (on the rising and falling edge of the clock) while GDDR5 transfers those two bits and an additional two bits at each midpoint using a second clock that is ninety degrees out of phase with the first. The difference in peak bandwidth illustrates the impact of these architectural differences—with the same bus width and a lower clock speed, Redwood has roughly twice the bandwidth of Llano's fGPU.

4.1 DDR3 vs. GDDR5 Memory

We evaluated how this difference translates to application performance using the Scalable Heterogeneous Computing (SHOC) benchmark suite [9]. SHOC includes synthetic benchmarks designed to measure bandwidth to each of the OpenCL memory spaces as well as real application kernels.

Figure 5 shows results from SHOC on the fGPU in Llano using DDR3 memory and the two discrete GPUs using GDDR5. Results are on a logarithmic scale and show the speedup of the discrete GPUs over Llano.

The HD5670 is the closest GDDR5-based discrete GPU to Llano's fGPU: the shader architecture is essentially the same, and it has only a 30% increase in clockspeed. Performance results from SHOC's MaxFlops benchmark reflect this, showing an approximately 25% improvement in pure floating point performance. Tests limited by global memory bandwidth show a much larger improvement, with the GDDR5 resulting in an approximate 3x speedup. The FirePro v8800 shows improvements of 2x to 8x, commensurate with or exceeding the increase in power consumption.

The lower half of Figure 5 shows much smaller speedups for the discrete GPUs. These lower results are benchmarks with a dependence on PCIe bus transfers, whether as an inherent part of the benchmark or as a GPU-centric benchmark with the CPU-GPU transfer of input and output data included. Although use of architecture-specific flags when creating buffers and special techniques when performing data transfers can allow somewhat higher bandwidths between System and Local memory in Fusion (see Section 2.1), the standard OpenCL mechanism for these transfers performs only slightly better than the discrete v8800 and is somewhat slower than transfer to the discrete HD5670. However, the tighter coupling of the fGPU with the CPU — seen, in part, in the faster Queue Delay performance due to the removal of PCIe latency — results in a net performance *increase* in the Llano, despite its slower shader clock and reduced memory bandwidth.

4.2 Contention

A performance model may be able to predict the preferred memory type for a given application based on its memory access characteristics. However, in a fused architecture, this tradeoff is complicated by contention effects between the CPU and GPU cores. For example, in the worst case, the throughput-optimized cores may generate enough memory requests to starve the latency-optimized cores, and, at a minimum, their traffic is likely to inject significant, unexpected latency. This type of resource exhaustion has traditionally been difficult to model.

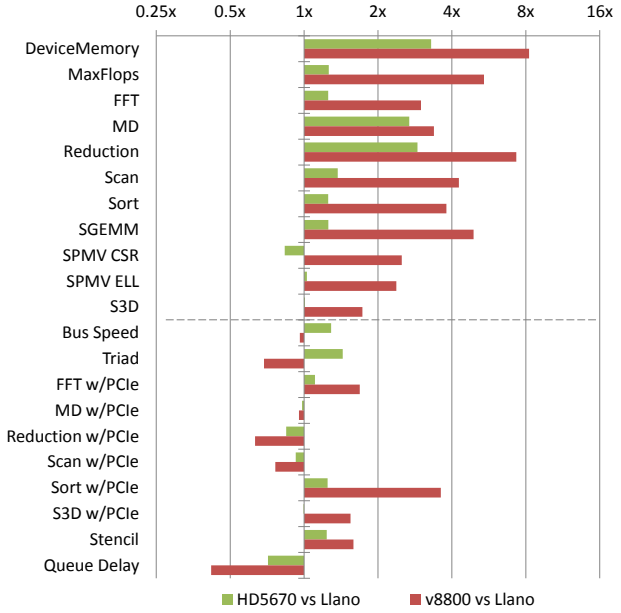

Figure 5: Speedup of a discrete Radeon HD5670 (Redwood, GDDR5) GPU and a discrete FirePro v8800 (Cypress, GDDR5), versus the fused GPU with DDR3 in Llano, on a logarithmic scale. Values greater than 1x indicate the discrete GPU outperforming Llano.

To measure these contention effects on Llano we ran the SHOC benchmark suite on the fGPU under two conditions. First, we measured performance with the CPU cores idle and then with the CPU cores running a bandwidth-bound kernel which lasted for the duration of the benchmark suite. For the CPU kernel, we used JACOBI, a micro-kernel that solves Laplace equations using Jacobi iteration, available in any numerical analysis text [24]. The tested kernel uses two 12288 x 12288 matrices with one hundred iterations. Our hypothesis was that the increased memory traffic from the Jacobi kernel would saturate memory bandwidth and cause degraded performance in the fGPU. The results in Figure 6 show a penalty ranging from fifteen to twenty-two percent for bandwidth-bound benchmarks and smaller penalties for those that incorporate at least a modest floating point intensity.

We include results from the same experiment using the HD5670 to confirm that this penalty is unique to the fused memory hierarchy. That is, no contention should occur with the HD5670, and while the measurements show some minor noise, the magnitude of that noise is less than one percent.

The CPU performance under these same scenarios shows the average runtime of the Jacobi benchmark increasing under contention. As seen in Table 3, we observe almost no

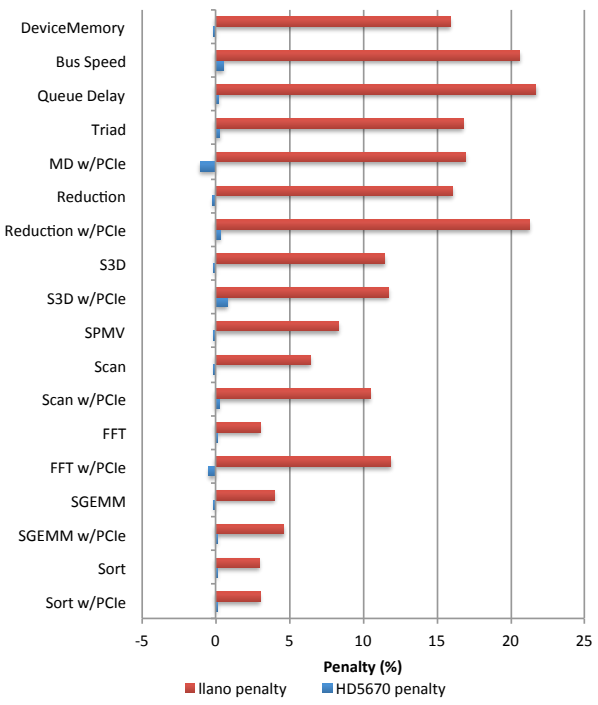

Figure 6: Performance penalty caused by memory contention between the CPU and the GPU.

contention with the HD5670, but a 10% penalty when simultaneously using the fGPU.

	Average Runtime
No CPU/GPU Contention	22.3 s
Contention with HD5670 GPU	22.5 s
Contention with Llano fGPU	24.5 s

Table 3: Performance of the CPU Jacobi kernel under various contention scenarios.

5. POWER VS. PERFORMANCE

5.1 GPU Power Usage

Tighter integration of separate components like a GPU and CPU generally results in lower power usage. The reduction in redundant hardware requires less power, and when components are physically closer together, less energy is required for data movement across wires and board traces. This, in turn, can also lead to secondary reductions, such as lower requirements for cooling. We study the power usage of Llano fGPU and compare it with the discrete Radeon HD5670. We measure the whole system power with an inline AC power meter and record the peak power consumption of the system. During the measurement, the default power saving features (i.e., employing appropriate power gating when possible) are enabled for both Llano and the discrete GPU.

Figure 7 displays system power consumption for specific GPU operations. We first notice that Llano using the fGPU has a lower power draw than the HD5670 in idle state by about 16.7%. We attribute this difference to the extra power consumption of global memory and peripheral circuitry on the HD5670. We further notice that the power difference is increased when the system is performing data transfers between the host and device (labeled 'Host/Dev Transfer' in the figure), floating point operations (labeled 'Compute FLOPS') and memory intensive operations (labeled 'Device Memory'). The lower power consumption of data transfer in Llano comes from the shorter data path and the elimination of the PCI bus and controller; the lower power consumption of floating point and memory operations comes from a lower shader clock on the Llano fGPU and from the simplified peripheral circuitry on the fGPU. Llano achieves an average power savings of approximately 20% compared to the HD5670 when running the computation kernels in the Level 1 tests from SHOC benchmark suite.

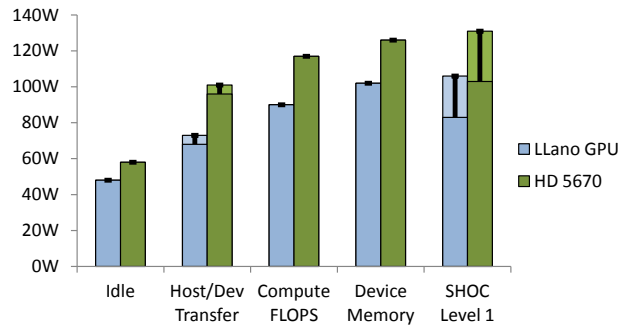

Figure 7: Full-system maximum power usage when using the integrated GPU in Llano and when using a discrete Radeon HD5670 GPU. The dark and light colors indicate the range of power usage across tests in each category.

5.2 Evaluation

Figure 8 shows the increase in power consumption versus the increase in performance when using the HD5670 compared to Llano for individual SHOC test results which do *not* depend on CPU-GPU data transfers. The vertical axis shows the best speedup obtained by using the discrete HD5670, and the horizontal axis shows the increase in peak power usage from the HD5670. We first notice that all points within the 2D performance-power plane are on the upper right side of the point (1.0x, 1.0x), which demonstrates better performance of HD5670 accompanied with greater power consumption.

The results in Figure 8 can be roughly clustered into three groups. The results above $y = 2.0x$, including DevMemRead, DevMemWrite, MD, and Reduction show the largest performance speedup with a relatively small power increase. The benchmarks in this group are generally characterized as memory operation intensive. The groups on the right side of the line $x = 1.3\times$, including MaxFlops and S3D, represent those with the least speedup and the greatest power consumption. The benchmarks in this group are, in general, characterized as floating point operation intensive. The remaining benchmarks show a wider range of floating point

and memory operation intensities; however, they do not approach either the performance improvement or the power increase of the other two groups.

Figure 9 shows similar results, this time for SHOC benchmarks which *do* include CPU-GPU data transfers. Note that some results are included in both Figures 8 and 9, but with different performance; in the latter, the data transfer times to GPU memory are incorporated. The relationship between power and performance is less straightforward in these results; the Llano fGPU performs relatively better once data transfers are included, often outperforming the HD5670, while power usage of the HD5670 is typically 30% higher than the Llano fGPU.

These results reveal that the high bandwidth of HD5670 brings performance benefit for memory intensive applications at the cost of small increase in power consumption, (i.e., the energy-delay product (EDP) is higher) while the tighter integration of Llano's fGPU may result in better energy efficiency for compute-intensive applications or those that require frequent CPU-GPU memory transfers.

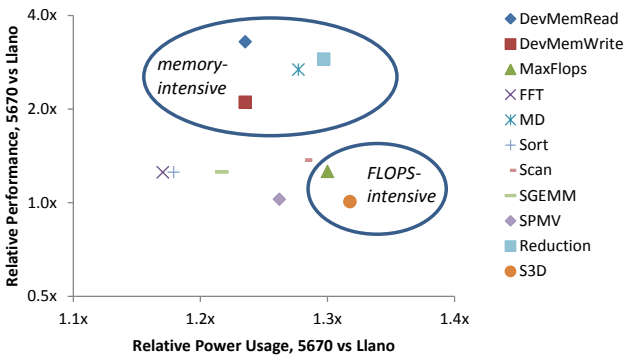

Figure 8: Performance and system power usage of the discrete HD5670 GPU on individual SHOC benchmarks relative to using the integrated GPU in Llano. This set of results does *not* include CPU-GPU transfers.

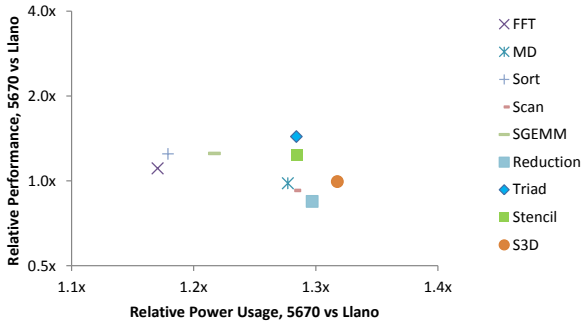

Figure 9: Performance and system power usage of the discrete HD5670 GPU on individual SHOC benchmarks relative to using the integrated GPU in Llano. This set of results *does* include CPU-GPU transfers.

6. PROGRAMMING MODELS VS. RUNTIME SYSTEMS

One of the reasons for the success of GPGPU and, especially CUDA, is the effectiveness of the programming model. CUDA's useful abstractions allowed access to the hardware resources of the GPU without having to learn graphics operations or very low-level device characteristics. For instance, the notion of a grid of thread blocks is easy to understand and hides many details about how the GPU actually schedules operations. The increased complexity of a fused heterogeneous architecture raises new questions for programming models. Which cores should a task run on? How does the programmer indicate that a task requires cache coherency among different cores? These questions begin to capture the next tradeoff, the contention between the desire for high-level abstractions and the complexity of the runtime systems required to support those abstractions.

Consider the problem of scheduling in more detail. Current capabilities in OpenCL allow the programmer to express a task that can run on the CPU or GPU cores. The programmer must then explicitly specify which device to use. The task may perform much better on one type of core, but it is the application's responsibility to track this and submit the task to the appropriate OpenCL device queue. Advanced scheduling features like preemption and task migration (from one core type to the other) are not yet available. However, the argument can be made that OpenCL is at a low-level of abstraction and the appropriate scheduling should be done by the application. At this extreme end of the tradeoff, maximal domain knowledge can be exploited, but the burden on application developers is large.

On the other hand, investment in a robust runtime system could significantly reduce this burden. Indeed, there have been several successes for these runtimes on CPUs including Cilk++ [15], with its work-stealing scheduler, and StarPU [2] for runtime scheduling on CPUs and discrete GPUs. These runtime systems often outperform static scheduling strategies. It seems reasonable to assume that static scheduling will only become more difficult on the APU, as the scheduler will have to include task affinities for a given type of core, costs of moving memory between pageable and "local" memory partitions, and coscheduling requirements for deciding when to enable cache coherency between CPU and fGPU, which cannot be used in the general case for performance reasons. Also, it remains to be seen if the increased flexibility of the abstractions for an APU could be successfully incorporated into a distributed environment in a model similar to Charm++ [14] or Chapel [3].

Unfortunately, at the time of this writing, no such runtime systems were available for evaluation on the APU. Despite this, we believe the availability of an effective runtime will be an important factor in the success of any APU.

7. RELATED WORK

While the Fusion APU architecture [4] is the first of its kind, the trend toward increased heterogeneity (and specifically GPGPU) was recognized by Owens et al. in their review of heterogeneity in HPC, covering several successes in game physics and biophysics [19]. The trend towards tighter integration is mentioned by Kaeli and Akodes who document the movement of multicore processors and GPUs into mainstream consumer electronics like tablets and cellular

phones [13] and identify the rapidly expanding class of problems which is amenable to GPGPU.

Prior to the release of the APU, several teams evaluated the performance of the ATI "Evergreen" GPU architecture, which includes two of the GPUs studied in this work, Cypress and Redwood. Both of these GPUs heavily influenced the GPU core architecture of Llano. These include a general study of performance and optimization techniques [16], a detailed characterization of the performance of atomic operations [17], and an evaluation of the energy efficiency of the architecture [26]. Others have also studied performance optimization on discrete NVIDIA GPUs [7, 20, 21, 22], some of which use a similar set of kernels to our experiments including GEMM [1, 11, 25], FFT [18], and molecular dynamics [5, 6].

There has also been an assessment of the Fusion architecture using the low power "Zacate" APU by Daga et al. [8] with a particular emphasis on absolute performance compared to a discrete GPU and the importance of the APU for scientific computing. Our results with the SHOC benchmark suite [9], also used by Daga, confirm the applicability of their findings at a different power scale. SHOC has also been used to study contention effects in previous work by Spafford et al. on NVIDIA-based platforms [23] with multiple GPUs that explores how PCIe contention becomes more complicated when traffic to the interconnect is considered.

8. CONCLUSIONS

We have identified five important tradeoffs for those designing or programming heterogeneous architectures with fused memory hierarchies. When constrained to a single type of physical memory, the choice between designs which focus on capacity and those that maximize bandwidth has a huge impact on GPU performance. So great, in fact, that this choice largely shaped the energy efficiency results in the last tradeoff, providing a fairly consistent method for grouping the mixed results of Figure 8. When system CPU-GPU data transfers become a more dominant portion of runtime (as seen in Figure 9), the efficiency benefits of tighter coupling do seem to overcome the specialized memory.

In the design of these fused memory hierarchies, the benefits of the tradeoff between cache coherency and scalability remains highly workload-specific and is not yet well understood. The addition of the FCL and the capability for cache coherency in Llano is an important first step. However, it will take time for this capability to make its way into mainstream programming models and scientific applications.

Furthermore, an APU-like design may only be beneficial if the application has a substantial parallel fraction and that fraction can be run on the throughput-oriented cores. The difference in performance of Llano's CPU cores and the Sandy Bridge CPU reflect the costs of not utilizing the appropriate core type or the potential penalty for devoting too many resources to a set of cores which won't be fully used by an application. When moving to a fused heterogeneous platform, an effective performance model and characterization of the instruction mix will be critical for choosing the correct core type for a kernel.

Finally, in the exploration of the tradeoff between simple abstractions and complex runtimes, we have identified a significant need for a robust APU runtime system. New concepts from the APU architecture impose additional complexity on programming models including kernel core affinity, coherence specification, and coscheduling requirements. While advanced applications may develop custom infrastructure to account for this complexity, the level of effort required to do so precludes many programmers from making this investment.

9. ACKNOWLEDGMENTS

This research is sponsored in part by the Office of Advanced Computing Research; U.S. Department of Energy. The work was performed at Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No.DE-AC05-00OR22725.

10. REFERENCES

[1] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical Linear Algebra on Emerging Architectures: the PLASMA and MAGMA Projects. *Journal of Physics: Conference Series*, 180, 2009.

[2] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In *Euro-Par 2009 Parallel Processing*, volume 5704 of *Lecture Notes in Computer Science*, pages 863–874. 2009.

[3] D. C. B. Chamberlain and H. P. Zima. Parallel Programmability and the Chapel Language. *The International Journal of High Performance Computing Applications*, 2007.

[4] N. Brookwood. AMD Fusion Family of APUs: Enabling a Superior, Immersive PC Experience. `http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf`, Mar 2010.

[5] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington. Implementing Molecular Dynamics on Hybrid High Performance Computers — Particle–Particle Particle–Mesh. *Computer Physics Communications*, 183(3):449 – 459, 2012.

[6] W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington. Implementing Molecular Dynamics on Hybrid High Performance Computers — Short Range Forces. *Computer Physics Communications*, 182(4):898 – 911, 2011.

[7] S. Carrillo, J. Siegel, and X. Li. A Control-Structure Splitting Optimization for GPGPU. In *Proceedings of the 6th ACM Conference on Computing Frontiers*, CF '09, pages 147–150, New York, NY, USA, 2009. ACM.

[8] M. Daga, A. Aji, and W. Feng. On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing. In *2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, pages 141 –149, July 2011.

[9] 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 '10, pages 63–74, New York, NY, USA, 2010. ACM.

[10] J. J. Dongarra and P. Luszczek. Introduction to the HPCChallenge Benchmark Suite. Technical Report

ICL-UT-05-01, Innovative Computing Laboratory, University of Tennessee-Knoxville, 2005.

[11] T. Endo, A. Nukada, S. Matsuoka, and N. Maruyama. Linpack Evaluation on a Supercomputer with Heterogeneous Accelerators. In *2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1 –8, 2010.

[12] J. Dongarra, P. Beckman et al. International exascale software roadmap. *International Journal of High Performance Computing Applications*, 25(1), 2011.

[13] D. Kaeli and D. Akodes. The Convergence of HPC and Embedded Systems in Our Heterogeneous Computing Future. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 9 –11, oct. 2011.

[14] L. V. Kale and G. Zheng. Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects. *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*, pages 265–282, 2009.

[15] C. Leiserson. The Cilk++ Concurrency Platform. *The Journal of Supercomputing*, 51:244–257, 2010.

[16] M. Daga, T. Scogland, and W. Feng. Performance Characterization and Optimization of Atomic Operations on AMD GPUs. In *Technical Report TR-11-08, Computer Science, Virginia Tech*, Retrieved from `http://eprints.cs.vt.edu/archive/00001159/`.

[17] M. Elteir, H. Lin, and W. Feng. Performance Characterization and Optimization of Atomic Operations on AMD GPUs. In *2011 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 234 –243, sept. 2011.

[18] 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*, SC '09, pages 30:1–30:10, New York, NY, USA, 2009. ACM.

[19] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[20] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, and W. Hwu. Program Optimization Space Pruning for a Multithreaded GPU. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '08, pages 195–204, New York, NY, USA, 2008. ACM.

[21] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S. zee Ueng, and W. Hwu. Program Optimization Study on a 128-Core GPU. In *Proceedings of the First Workshop on General Purpose Processing on Graphics Processing Units*, 2007.

[22] S. Ryoo, C. I. Rodrigues, S. S. Stone, J. A. Stratton, S.-Z. Ueng, S. S. Baghsorkhi, and W. Hwu. Program Optimization Carving for GPU Computing. *Journal of Parallel and Distributed Computing*, 68(10):1389 – 1401, 2008. General-Purpose Processing using Graphics Processing Units.

[23] K. Spafford, J. S. Meredith, and J. S. Vetter. Quantifying NUMA and Contention Effects in Multi-GPU Systems. In *Proceedings of The Fourth Workshop on General Purpose Processing on Graphics Processing Units*. ACM, 2011.

[24] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer; 2nd edition, 1996.

[25] V. Volkov and J. W. Demmel. Benchmarking GPUs to Tune Dense Linear Algebra. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages –11, Piscataway, NJ, USA, 2008. IEEE Press.

[26] Y. Zhang, Y. Hu, B. Li, and L. Peng. Performance and Power Analysis of ATI GPU: A Statistical Approach. In *2011 6th IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 149 –158, July 2011.