On the Performance, Energy, and Power of Data-Access Methods in Heterogeneous Computing Systems

Rubasri Kalidas,* Mayank Daga,† Konstantinos Krommydas,‡ and Wu-chun Feng*‡

*Dept. of Electrical and Computer Engineering ‡Dept. of Computer Science
Virginia Tech, USA
{krubasri, kkrommy, feng}@vt.edu

†AMD Research
Advanced Micro Devices, Inc., USA
Mayank.Daga@amd.com

***Abstract*—Graphics processing units (GPUs) have delivered promising speedups in data-parallel applications. A discrete GPU resides on the PCIe interface and has traditionally required data to be moved from the host memory to the GPU memory via PCIe. In certain applications, the overhead of these data transfers between memory spaces can nullify any performance gains achieved from faster computation on the GPU. Recent advances allow GPUs to *directly* access data from the host memory across the PCIe bus, thereby alleviating the data-transfer bottlenecks.**

Another class of accelerators called accelerated processing units (APUs) mitigate data-transfer overhead by placing CPU and GPU cores on the same physical die. However, APUs in the current form provide several different data paths between the CPU and GPU, all of which can differently affect application performance. In this paper, we explore the effects of different available data paths on both GPUs and APUs in the context of a broader set of computation and communication patterns commonly referred to as dwarfs.

***Keywords*-GPU, Accelerated Processing Unit (APU), Heterogeneous System Architecture (HSA), data transfer, characterization, access methods**

I. Introduction

GPUs have become increasingly popular as an accelerator platform due to their remarkable performance-price and performance-power ratios. However, the intrinsic computational capability of GPUs is not always optimally utilized because of their imposed data-transfer requirements over the PCIe interface. The transfer bandwidth of PCIe is limited to 16GB/s whereas the GPU itself can consume data at the rate of hundreds of GB/s from its memory [1], [2]. Therefore, only such applications that amortize the cost of these data transfers will benefit from GPU execution [3]. To overcome the data-transfer overheads, recent GPU advancements enable GPUs to directly access data from the host memory as well as enable CPUs to directly access data from the GPU memory. Such data is required to be marked as *zero-copy* during allocation.

A novel class of accelerators called *accelerated processing units (APUs)* place CPU and GPU cores together on a single silicon die [4]. Unlike a traditional CPU paired with a discrete GPU (dGPU) connected over PCIe, an APU unifies the memory address spaces of the CPU and GPU. Therefore, an APU eliminates the overhead associated with data transfer, which occurs with dGPUs [5]. In its present incarnation, an APU provides different data paths between the CPU and GPU memory spaces depending on (1) where the data resides, i.e., in host memory or GPU memory and (2) whether the data is marked as zero-copy. As to be discussed in Section III, different combinations of the above result in *five* possible data-access methods between the CPU and GPU, all of which can impact application performance.

In this paper, we first propose the data-access methods and then study their effects on a broad set of applications with varying characteristics. We then qualitatively ascertain the best performing method, depending upon the application and the architecture, and characterize the data-access methods via the use of three different applications from the OpenDwarfs benchmark suite [6]: (1) breadth-first search (BFS), (2) Needleman-Wunsch (NW) dynamic programming, and (3) speckle-reducing anisotropic diffusion (SRAD).

Figure 1 shows the percentage of time spent in data transfers and kernel execution for the aforementioned three dwarfs running on an AMD Radeon™ HD 7970 discrete GPU. The percentage of time spent in data transfers is more than twice the kernel execution in the cases of BFS and SRAD. These cases are representative of applications where optimizing data transfer would improve overall performance. Although the data transfer time constitutes less than 20%

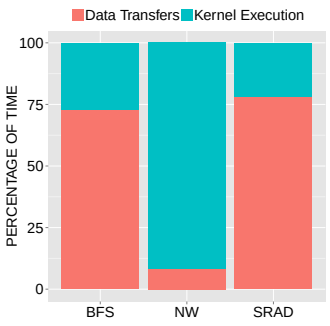


Figure 1. Percentages of data transfer and kernel execution times

of total time in case of NW, the data dependencies in the NW algorithm are high, which in turn, limits parallelism. Thus, optimizing the data transfer in NW would not result in significant speedup, but including this application can provide insight into whether any of the optimizations would suit all kinds of applications. We study these three dwarfs under different data-access methods to gain insight into the best performing technique for each dwarf across various architectures.

We observed that by implementing different data-access methods, we can achieve up to a 148× speedup in performance over the default data-access method. Our results demonstrate that APUs consume an order of magnitude less energy compared to a discrete GPU, e.g., energy consumption on an AMD Opteron™X2150 APU is up to 114× less than a discrete GPU.

The rest of the paper is organized as follows. In Section II we present the background on different data paths in discrete GPUs and (integrated) APUs. Section III discusses the different data-access methods. In Section IV we present our experimental setup, followed by results and observations in Section V. Section VI discusses related work. We present conclusions in Section VII.

II. Background

This section presents the background information on the data paths that exist on both discrete GPUs and APUs. We also present an overview of the APU architecture since it is a fairly new accelerator platform.

A. Discrete GPUs

A discrete GPU resides on the PCIe interface in the traditional CPU+GPU setup, as shown in Figure 2. The x86 cores and the unified north bridge (UNB) are part of the CPU core, whereas the discrete GPU is a separate physical entity with vector cores and its own memory space. This is why GPUs must have data copied to their own memory over the PCIe interface, which leads to significant bottlenecks for some applications. Modern GPUs mitigate the data-transfer overheads by allowing the GPU to directly access data from the host memory. Such data is required to be marked as zero-copy and is accessed over PCIe. Therefore, two data paths exist on a discrete GPU: (1) copying the data from the host memory to GPU memory and then accessing it from the GPU memory and (2) directly accessing data from the host memory over the PCIe interface. Because GPU memory has an order of magnitude higher bandwidth than host memory, the first data path is preferred *if* an application can amortize the cost of the data transfers. Alternatively, data can also be directly allocated into the GPU memory, which the CPU can access over PCIe, as required.

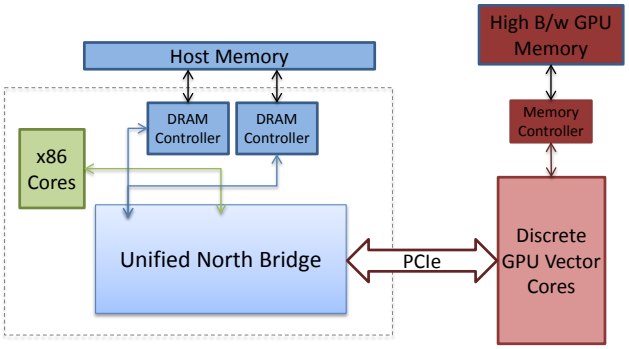


Figure 2. High-level block diagram illustrating a CPU and discrete GPU setup across PCI Express (PCIe).

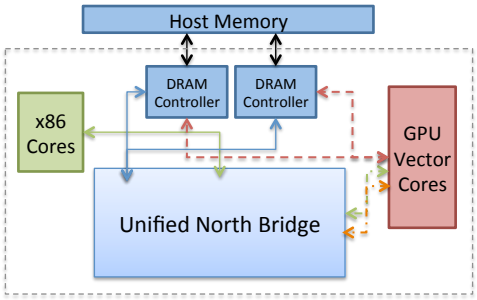


Figure 3. High-level block diagram of an AMD A10-7850K APU. The different data paths between CPU and GPU are shown using arrows.

B. Accelerated Processing Units (APUs)

An accelerated processing unit (APU) is AMD's implementation of the emerging heterogeneous system architecture (HSA) [7]. Fundamentally, an APU combines the general-purpose x86 scalar cores of a CPU and vector cores of a GPU onto a single silicon die. Figure 3 depicts a high-level block diagram of an AMD APU. Having a fused CPU-GPU allows an APU to provide direct mapping between the CPU and GPU virtual address space and to enable demand paging, which allows the GPU to bring pages on-demand into memory, just like the CPU. The GPU on the current generation of APUs can access host memory via several data paths. First, GPUs can directly access a part of host memory designated as a "GPU frame buffer." This path at present provides the highest bandwidth and works similarly to the GPU memory of a discrete GPU. There also exists two different paths via the UNB, depending on whether the data that the GPU is accessing is coherent with the CPU.

III. Data-Access Methods

The data to the GPU can be made available through buffers residing in host memory or device memory. Depending on where the buffer is located and how the data is transferred between the host and device, we present five different data-access methods, as shown in Figure 4. Table I summarizes the characteristics of various methods.

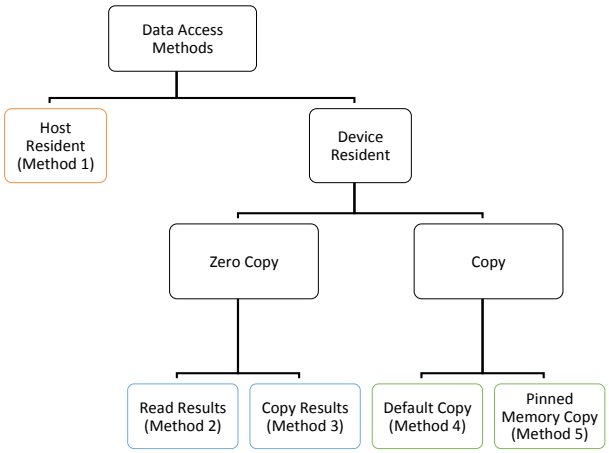


Figure 4. Data-access methods

A. Host Resident (Method 1)

With the *Host Resident* (HR) method, buffers are created on the host using the OpenCL™ flag `CL_MEM_ALLOC_HOST_PTR`, as shown in Figure 5a. The host buffer does not require any backing buffer and is directly initialized with data. This buffer is allocated as pinned memory, and hence, its contents are not pageable by the operating system. Since there does not exist any copy of the buffer on the device, a host-resident buffer is a zero-copy buffer. Discrete GPUs directly access this buffer across PCIe, whereas the APUs access it via the unified north bridge (UNB). There is no explicit data-transfer involved in this method.

B. Device Resident - Zero Copy - Read Results (Method 2)

With the *Device Resident - Zero Copy - Read Results* (DR-ZC-RR) method, buffers are created using the flag `CL_MEM_USE_PERSISTENT_MEM_AMD`. As the name suggests, buffers of this type reside in device memory without a copy of it on the host as shown in Figure 5b. Data to such buffers is written directly by the CPU through the uncached write-combine path, and the results are stored in the GPU memory after the computation. The CPU accesses these results by reading directly from the GPU memory across the PCIe interface and UNB on discrete GPUs and APUs, respectively. As a result, the CPU reads are very slow.

C. Device Resident - Zero Copy - Copy Results (Method 3)

In the *Device Resident - Zero Copy - Copy Results* (DR-ZC-CR) method, the buffer resides in device memory. The difference between this method and DR-ZC-RR is that, in this case, the results are copied back to the host memory as shown in Figure 5c, whereas for the latter, the results were kept in the GPU memory after computation. This method is implemented using the flag `CL_MEM_USE_PERSISTENT_MEM_AMD` along with `clEnqueueReadBuffer()` to copy the results back. Depending on the size of the results buffer, copying the results may or may not be beneficial.

D. Device Resident - Copy - Default Copy (Method 4)

In the *Device Resident - Copy - Default Copy* (DR-C-DC) method, data is copied into a device buffer created using no flags. The data and results are copied back and forth as shown in Figure 5d using `clEnqueueWriteBuffer()` and `clEnqueueReadBuffer()`, respectively.

E. Device Resident - Copy - Pinned Copy (Method 5)

In the *Device Resident - Copy - Pinned Copy* (DR-C-PC) method, the data resides in device memory. This method "copies" the data at the *peak interconnect bandwidth* to the device. We use two buffers, a pinned host side buffer created using the flag `CL_MEM_ALLOC_HOST_PTR` and a device buffer created without any flags as shown in Figure 5e. The data is made accessible to the GPU by first filling the host buffer and then copying from it to the device buffer. Since the host buffer is pinned, the data is transferred to the second buffer at peak interconnect bandwidth.

IV. Experimental Setup

A. Software Environment

The software environment consists of a Debian Linux 64-bit kernel v3.2.0 and GCC v4.7.2. We use OpenCL™ to program the applications on both the discrete GPU and APUs [8]. The driver versions used are AMD Catalyst™ v14.20 for AMD A10-7850K APU and AMD Catalyst v13.25 for the remaining systems.

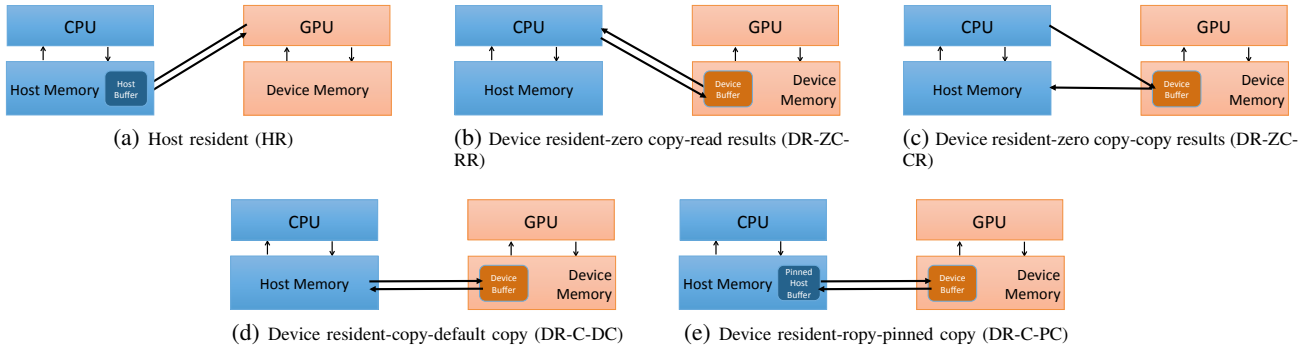
Power measurements for our experiments are taken using a WattsUp Pro meter. Power readings are taken every second, and the total power is averaged across an application's execution time. Since all the applications run for more than one second, the accuracy was sufficient. We repeated the experiment ten times and took the median across the ten runs. Energy is calculated by taking a product of the power consumed and the application's execution time.

B. Hardware Environment

We use four different accelerator platforms to characterize the data-access methods: (1) AMD Radeon™HD 7970 discrete GPU (with AMD Opteron 6272 CPU), (2) AMD A10-5800K, a second-generation APU, (3) AMD Opteron X2150, a low-power APU, and (4) AMD A10-7850K, a third-generation APU that is HSA-compliant. The hardware specifications of these platforms is shown in Table II.

C. Applications

Breadth-First Search (BFS): BFS is a graph-traversal algorithm that traverses nodes in a graph in a breadth-first manner and identifies the unvisited neighbors for each node. Irregular memory-access patterns arise in BFS as different nodes can have different numbers of neighbors.



(a) Host resident (HR)

(b) Device resident-zero copy-read results (DR-ZC-RR)

(c) Device resident-zero copy-copy results (DR-ZC-CR)

(d) Device resident-copy-default copy (DR-C-DC)

(e) Device resident-ropy-pinned copy (DR-C-PC)

Figure 5. Illustration of various data-access methods

Table I
CHARACTERISTICS OF DATA-ACCESS METHODS

Method	OpenCL Flag	Buffer Location	Zero-Copy
Host Resident (HR)	CL_MEM_ALLOC_HOST_PTR	Pinned host memory	Yes
Device Resident-Zero Copy-Read Results (DR-ZC-RR)	CL_MEM_USE_PERSISTENT_MEM_AMD	Device memory	Yes
Device Resident-Zero Copy-Copy Results (DR-ZC-CR)	CL_MEM_USE_PERSISTENT_MEM_AMD	Device memory	Yes
Device Resident-Copy-Default Copy (DR-C-DC)	No flags	Device memory	No
Device Resident-Copy-Pinned Copy (DR-C-PC)	CL_MEM_ALLOC_HOST_PTR and No flags	Device memory	No

Table II
HARDWARE SPECIFICATIONS OF ACCELERATORS

Name	AMD Radeon^{TM} HD 7970	AMD A10-5800K	AMD A10-7850K	AMD Opteron X2150
Stream Processors	2048	384	768	128
Compute Units	32	6	12 (4 CPU + 8GPU)	2
Core Clock Frequency	925 MHZ	800 MHZ	720 MHZ	495 MHZ
Memory Bus type	GDDR5	DDR3	DDR3	DDR3
Device Memory	3 GB	800MB	3 GB	800MB
Local Memory	32KB	32 KB	32KB	32 KB
Max Work-group size	256	256	256	256
Host Processor	AMD Opteron 6272	AMD A10-5800K	AMD A10-7850K	AMD Opteron X2150
CPU frequency	2.1 GHZ	1.4 GHZ	1.7 GHZ	800 MHZ
System memory	16 GB	8 GB	6 GB	8 GB

Needleman-Wunsch (NW): NW is a dynamic programming algorithm that calculates the optimal global alignment between two DNA sequences. NW consists of filling a matrix with weighted scores and tracing it back after it is filled. NW accesses memory intensely as computing the score of a cell in the matrix involves accessing all the neighboring cells.

Speckle Reducing Anisotropic Diffusion (SRAD): SRAD is an iterative application that removes locally uncorrelated noise using a set of partial-differential equations (PDEs). SRAD requires synchronization among the GPU execution units due to the inter-dependency between iterations. SRAD finds use in several areas of image compression.

V. RESULTS AND DISCUSSION

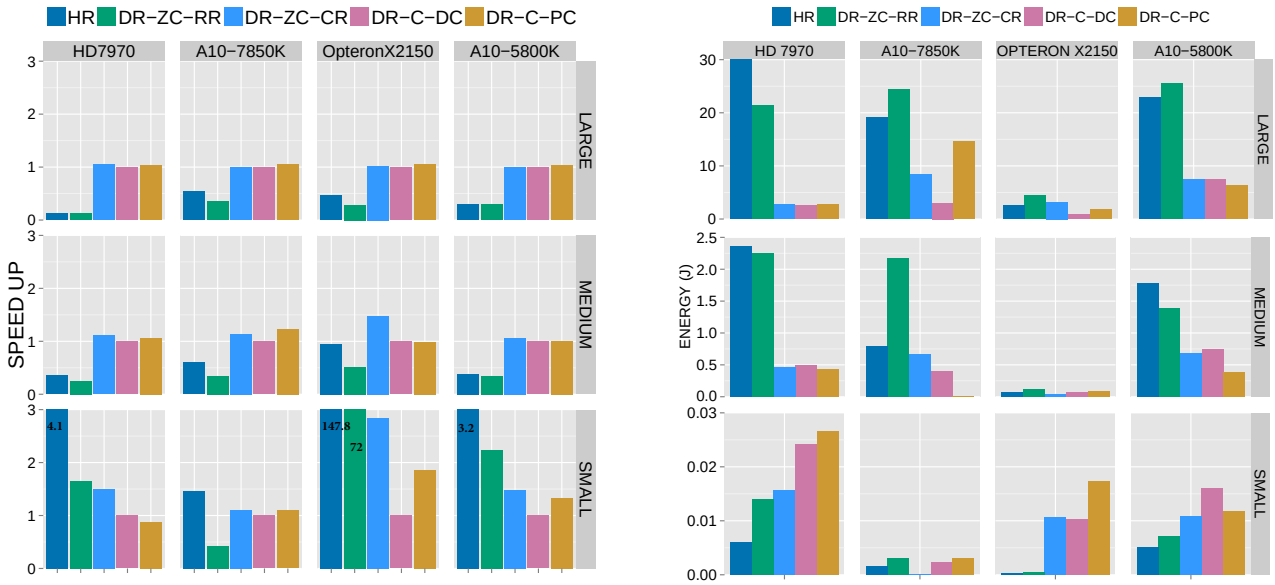
This section briefly presents the observations and findings of the different data-access methods.

A. Graph Traversal: Breadth First Search (BFS)

1) Performance: The performance for BFS on different platforms across all the data-access methods relative to DR-C-DC is shown in Figure 6a. The HR method performs the best for a small dataset (i.e., a graph with 1K nodes) on all the platforms. Specifically, on the Opteron X2150 APU, HR provides a speedup of 148× whereas on other platforms, the speedup ranges between 1.5× and 4.0×. However, the performance of HR worsens for medium and large datasets, which consist of 256K and 2M nodes, respectively. This is because for larger datasets, an increased amount of time is spent in accessing data from the host memory, which in turn, adversely affects performance due to numerous irregular memory accesses that occur in BFS.

Device-resident methods perform the best for medium and large datasets. We find that the performance of device-resident methods is dependent on how the results buffer is accessed by the host after the computation. Reading the results directly from the host without copying the buffer first, i.e, DR-ZC-RR, is 20%-30% slower on average. This is because the CPU reads from the device buffer occur at an extremely low bandwidth, as shown in Table III.

DR-ZC-CR, DR-C-DC, and DR-C-PC methods all perform on par with each other. DR-ZC-CR has a slight edge in



(a) Performance (b) Energy

Figure 6. Performance and Energy for Breadth-First Search (BFS). Baseline for performance is the default data-access method (DR-C-DC).

Table III
BUFFERBANDWIDTH BENCHMARK RESULTS

Platform	CPU Read Bandwidth
AMD Radeon HD 7970	0.01 GB/S
AMD A10-5800K	0.01 GB/S
Opteron X2150	0.02 GB/S
AMD A10-7850K	0.02 GB/S

some cases because of reduced data transfers than the other two methods. DR-ZC-CR requires data to be transferred only for copying the results back to the host; no data copies are required to initiate computation on the GPU. To summarize, DR-ZC-CR is the best for BFS on all the platforms.

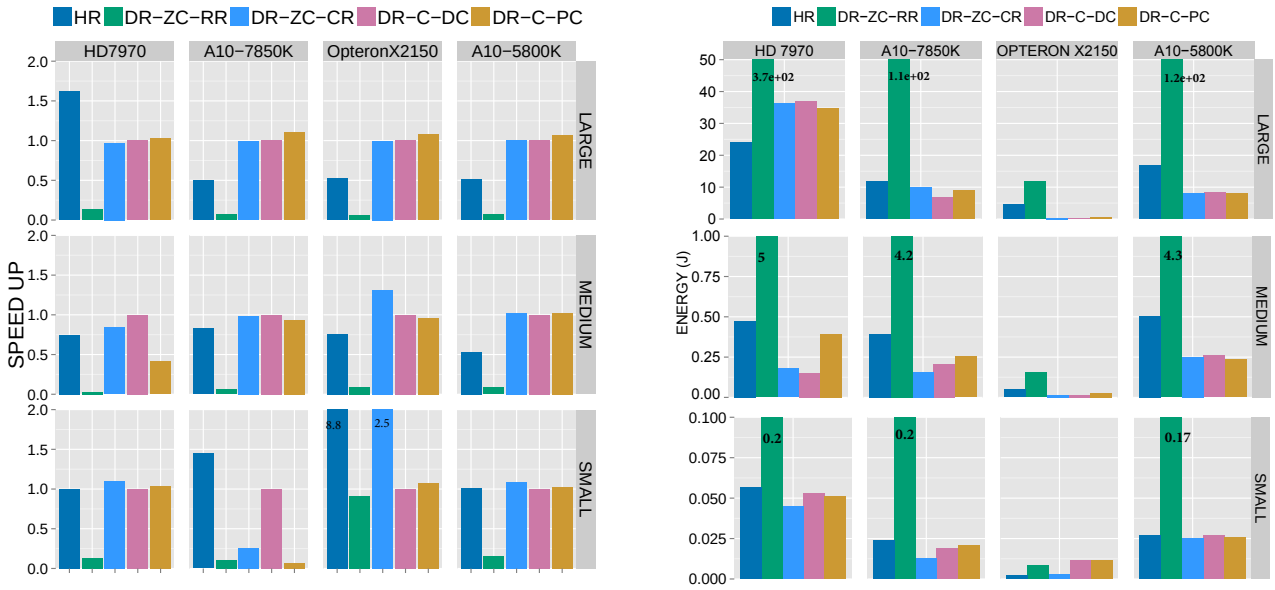
2) Energy: Figure 6b depicts the energy consumption of BFS for the different access methods across all four platforms. The AMD Opteron X2150 APU, an ultra low-power GPU designed for data center servers, is the least energy consuming for datasets which matter in a real scenario. The AMD A10-7850K APU is best for small datasets. The HR method consumes the least energy for small datasets, whereas device-resident methods are the best for larger datasets.

The HR method consumes the most energy on a discrete GPU because it requires data to be accessed from the host memory across PCIe and we suspect that the memory controller being busy throughout the runtime of application in case of HR method might be the reason for the high energy consumption. Since the time taken for the DR-ZC-RR method is much higher, although the power consumption is comparable to other device resident methods, the overall energy consumed is much higher.

B. Dynamic Programming: Needleman-Wunsch (NW)

1) Performance: The performance for NW on different platforms across all the data-access methods relative to DR-C-DC is shown in Figure 7a. Since NW is computationally intensive and involves numerous memory accesses, the performance of HR tends to worsen as the size of the dataset increases (2× slower for the large dataset in case of APUs). DR-ZC-RR performs the worst when compared to all methods. The performance of DR-ZC-RR is almost 10× slower than the other methods. This is because a large amount of data is read back by the CPU (almost $1/2$ of the total data accessed), and the CPU read bandwidth is low, as shown in Table III. Since the percentage of data transfer time is less than 20% for NW, as shown in Figure 1, the device-resident methods, DR-ZC-CR, DR-C-DC and DR-C-PC perform the same for almost all of the cases. Overall, DR-C-CR, DR-C-PC and DR-C-DC methods perform the best.

2) Energy: Figure 7b shows the energy consumption of NW for the different access methods across all platforms. From the figure, we note that the energy consumption of DR-ZC-RR is 9× to 10× higher (on average) than the other access methods. The device resident methods do not involve the host throughout and so the power consumed is fairly the same. But because of the long time taken to run the application in case of DR-ZC-RR, its energy consumed is higher than the other device resident methods. The device-resident methods DR-ZC-CR, DR-C-DC and DR-C-PC do not differ much in terms of energy consumption since the



(a) Performance (b) Energy

Figure 7. Performance and Energy for Needleman-Wunsch (NW). Baseline for performance is the default data-access method (DR-C-DC).

performance numbers of these three methods are fairly close. The host-resident method consumes almost 2× more energy than the best performing device resident methods because in HR method GPU has to access the host every time there is a need of data and as a result memory controllers might be kept busy throughout, thus consuming more energy.

When energy consumption between the platforms is compared, AMD Opteron X2150 outperforms other GPUs/APUs by a great margin for all the methods. Energy consumption on an AMD Opteron X2150 APU for DR-ZC-CR is 114× less than the discrete GPU and 30× less than the APUs.

C. Structured Grids: Speckle Reducing Anisotropic Diffusion (SRAD)

1) Performance: The performance for SRAD on different platforms across all the data-access methods relative to DR-C-DC is shown in Figure 8a. When comparing the data-access methods, DR-C-PC outperforms the other methods for the large dataset on all the platforms. HR method performs sub-optimally because SRAD entails large amount of data movement even before any computation begins.

So overall, DR-C-PC is an ideal choice when it comes to performance but it requires the extra pinning step on the part of the programmer, thereby, trading programmability for performance

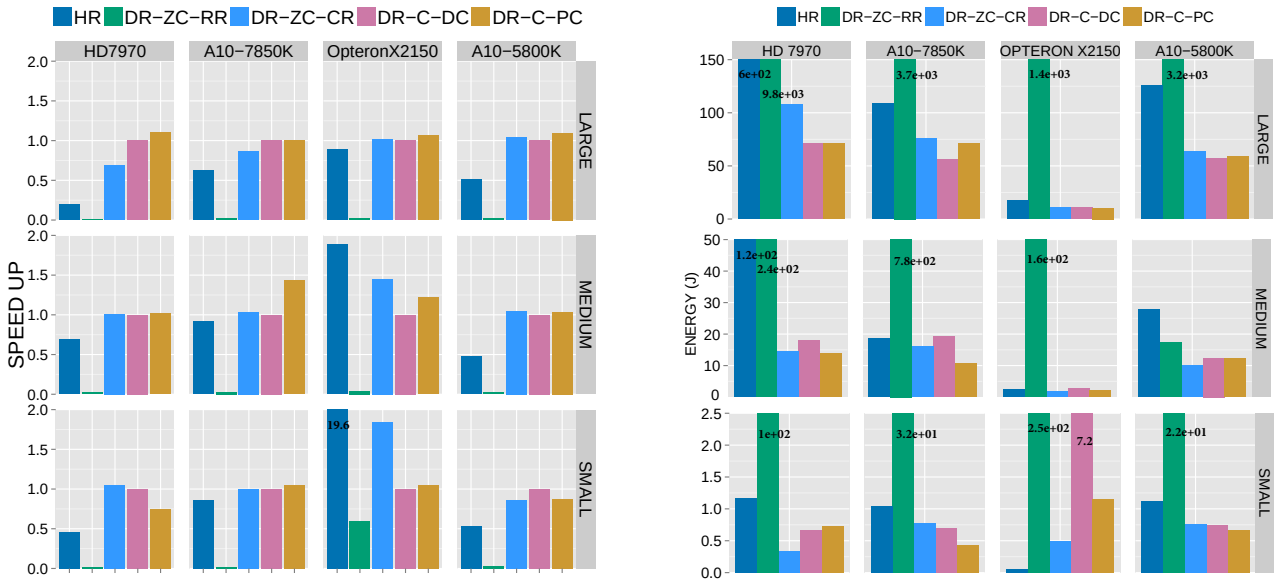
2) Energy: Even in the case of SRAD, out of the four platforms considered, the AMD Opteron X2150 APU is clearly a winner in terms of energy consumption, as shown in Figure 8b. Although the AMD Opteron X2150 APU has higher energy consumption for a small data set, it consumes 10× less energy than the other APUs and 33× less energy than the discrete GPU when the HR method is compared.

Out of the five access methods, DR-ZC-CR consumes least energy for large data sets. For small and medium datasets, DR-C-DC and DR-C-PC are better at energy consumption than the rest of the methods. Energy for DR-ZC-RR on AMD Opteron X2150 is high for the SRAD application because of the higher execution time of SRAD. DR-ZC-RR consumes more energy incase of SRAD than the other two applications.

D. Power vs. Performance

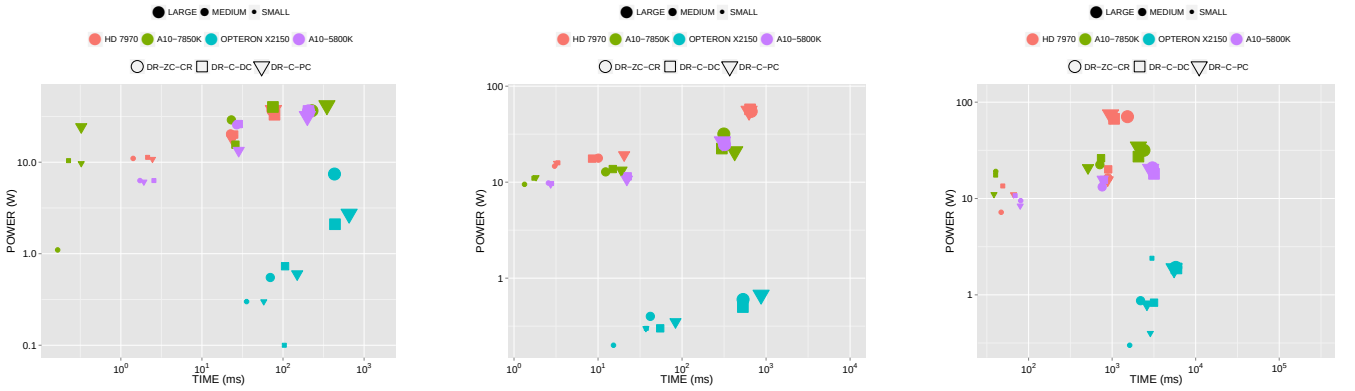
Power vs. performance plots give us a way to evaluate and choose the appropriate platforms and methods based on the tradeoffs between performance and power consumption. Power vs. performance graphs are plotted on a log-log scale and consists of power-performance points of the access methods on the various platforms considered for small, medium and large datasets. In plotting the Power vs. performance plots, only the three best performing methods, DR-ZC-CR, DR-C-DC and DR-C-PC, are considered for clarity purpose.

1) BFS: The points on the lower left signify high performance with low power consumption. AMD Opteron X2150 APU has all the points on the bottom part of the plot signifying low power consumption. If power consumption is of more importance, like in data centers, AMD Opteron X2150 APU is ideal, though it requires some trade-off in performance. If performance is key then the APUs are ideal as implied by data points on the left, .e.g. DR-C-PC and



(a) Performance (b) Energy

Figure 8. Performance and Energy for Speckle Reducing Anisotropic Diffusion (SRAD). Baseline for performance is the default data-access method (DR-C-DC).



(a) Breadth First Search (BFS) (b) Needleman-Wunsch (NW) (c) Speckle Reducing Anisotropic Diffusion (SRAD)

Figure 9. Power vs Performance (Execution Time)

DR-ZC-CR methods for large dataset in the case of AMD A10-7850K APU and AMD A10-5800K APU.

2) NW: Even for NW, AMD Opteron X2150 APU has all the data points in the bottom part of the plot signifying low power consumption. The data points of AMD Opteron X2150 APU for small and medium datasets on the bottom left mean the ideal case, i.e., high performance with less power consumption.

3) SRAD: AMD Opteron X2150 APU's points lie on the bottom part in the case of SRAD as well which shows that the machine is highly power efficient irrespective of the application and data sizes. However, saving power means trading off performance.

VI. Related Work

With the advent of GPUs in data centers power and energy concerns have become of high importance. Data transfers constitute a significant bottleneck in GPU computing. While many studies exploiting GPUs claim huge speed-ups, they more often than not ignore data transfer overhead, which is an important factor affecting performance [9]. Daga et al. [10] re-visit Amdahl's law to account for the parallel overhead incurred by data transfers, and Boyer et al. [11] attempt to improve GPU performance prediction with data transfer modeling.

The fused CPU-GPU core architectures and memory hierarchies partially address the severity of PCIe data transfer

overheads. Furthermore, more options became available for both discrete and fused GPUs. Lee et al. have discussed different memory movement techniques of micro-benchmarks on discrete GPUs and second generation APUs [12]. Spafford et al. have studied the trade-offs of fused memory hierarchies in heterogeneous systems [13]. In our work, we acknowledge that data transfers are an important aspect in GPU computing and attempt to ameliorate their effects by systematically studying various data-access methods on discrete GPUs and APUs. We use three representative cases to identify the effect of data transfers on specific computation and communication patterns using applications from the dwarfs categorization.

Various works have explored the energy efficiency of GPUs for scientific computing [14], and proposed new metrics for evaluating energy efficiency in HPC systems [15]. Kim et al. discussed the efficiency of integrated GPUs in terms of both performance and energy in data canter workloads [16]. Other works have proposed different memory hierarchies for GPU computing with an emphasis on energy efficiency [17], [18]. Rofouei et al. conclude that employing GPUs is only efficient if performance gains exceed a specific bound [19]. In this work, we shed light on the implications of different data-access methods on performance gains and consequently energy consumption.

VII. Conclusion

Data transfers are often cited as an impediment towards the widespread adoption of GPUs for compute. Discrete GPUs have mitigated the data transfer overhead by allowing the GPUs to directly access data in the host memory, albeit over the PCIe interface. Accelerated Processing Units (or APUs) promise to alleviate data transfer overheads by combining the CPU and GPU cores on the same silicon die, thereby, not requiring any data transfers. As such both discrete GPUs and APUs provide several paths for accessing data which can contrastingly affect application performance. In this paper, we present five different data-access methods and study their performance and energy implications on three representative classes of computation and communication patterns using the OpenDwarfs benchmark suite.

Our experiments suggest that it is best to allocate data directly in the GPU memory (marking it as *zero-copy*) as compared to the conventional wisdom of allocating on the host and then transferring to the GPU. We also find that the results of GPU computation should be first copied to the host instead of directly accessing it in the GPU memory. We illustrate that the use of such method can provide a manifold speedup compared to the default method of moving data back and forth between host and GPU memory. We highlight that APUs consume an order of magnitude less energy than discrete GPUs for the class of applications we study while providing comparable performance.

VIII. Acknowledgment

This work was supported in part by NSF I/UCRC IIP-1266245 via the NSF Center for High-Performance Reconfigurable Computing (CHREC).

References

[1] "The PCIe Gen. 3 Specification," https:www.pcisig.comspecificationspciexpressbase3.

[2] "The R9 Specification," http://www.amd.com/en-us/products/graphics/desktop/r9.

[3] M. Daga, W. Feng, and T. Scogland, "Towards Accelerating Molecular Modeling via Multiscale Approximation on a GPU," in *1st IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*, 2011, pp. 75 –80.

[4] D. Bouvier and B. Sander, "Applying AMD's Kaveri APU for Heterogeneous Computing." in *Hot Chips: A Symposium on High Performance Chips (HC26)*, 2014.

[5] M. Daga, M. Nuter, and M. Meswani, "Efficient Breadth-First Search on a Heterogeneous Processor," in *IEEE International Conference on Big Data (BigData)*, 2014.

[6] K. Krommydas, W. Feng, M. Owaida, C. Antonopoulos, and N. Bellas, "On the characterization of opencl dwarfs on fixed and reconfigurable platforms," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, June 2014, pp. 153–160.

[7] "The HSA Foundation," 2012, http://hsafoundation.com/.

[8] A. Munshi, "The OpenCL Specification, Version 1.2," *Khronos OpenCL Working Group*, 2013.

[9] C. Gregg and K. Hazelwood, "Where Is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer," in *2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2011, pp. 134–144.

[10] M. Daga, A. M. Aji, and W. Feng, "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing," in *Proceedings of the 2011 Symposium on Application Accelerators in High-Performance Computing*, ser. SAAHPC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 141–149.

[11] M. Boyer, J. Meng, and K. Kumaran, "Improving gpu performance prediction with data transfer modeling," in *2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, May 2013, pp. 1097–1106.

[12] K. Lee, H. Lin, and W. Feng, "Performance Characterization of Data-intensive Kernels on AMD Fusion Architectures," *Comput. Sci.*, vol. 28, no. 2-3, pp. 175–184, May 2013.

[13] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, P. C. Roth, and J. S. Vetter, "The Tradeoffs of Fused Memory Hierarchies in Heterogeneous Computing Architectures," in *Proceedings of the 9th Conference on Computing Frontiers*, ser. CF '12. New York, NY, USA: ACM, 2012, pp. 103–112.

[14] S. Huang, S. Xiao, and W. Feng, "On the Energy Efficiency of Graphics Processing Units for Scientific Computing," in *IEEE International Symposium on Parallel Distributed Processing, 2009. IPDPS 2009*, May 2009, pp. 1–8.

[15] B. Subramaniam and W. Feng, "The Green Index: A Metric for Evaluating System-Wide Energy Efficiency in HPC Systems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, May 2012, pp. 1007–1013.

[16] S. Kim, I. Roy, and V. Talwar, "Evaluating integrated graphics processors for data center workloads," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, ser. HotPower '13. New York, NY, USA: ACM, 2013, pp. 8:1–8:5.

[17] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A Locality-aware Memory Hierarchy for Energy-efficient GPU Architectures," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 86–98.

[18] J. Zhao, G. Sun, G. H. Loh, and Y. Xie, "Energy-efficient GPU Design with Reconfigurable In-package Graphics Memory," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 403–408.

[19] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-aware High Performance Computing with Graphic Processing Units," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, ser. HotPower'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 11–11.