Comparing LLC-memory Traffic between CPU and GPU Architectures

Mohammad Alaul Haque Monil*, Seyong Lee †, Jeffrey S. Vetter† and Allen D. Malony*
{mmonil, malony}@cs.uoregon.edu, lees2@ornl.gov, vetter@computer.org
* University of Oregon, † Oak Ridge National Laboratory, USA

***Abstract*—The cache hierarchy in modern CPUs and GPUs is becoming complex. The introduction of additional complexities by manufacturers makes understanding the handshake between the memory access pattern and the cache hierarchy difficult. Moreover, the details of different cache policies are not publicly available. Therefore, the research community relies on observation to understand the relation between memory access patterns and cache hierarchy. Our previous studies delved into different micro-architectures of Intel CPUs. In this study, GPUs from NVIDIA and AMD are taken into consideration. Even though the execution models in CPUs and GPUs are distinct, this study attempts to correlate the behavior of the cache hierarchy of CPUs and GPUs. Using the knowledge gathered from Intel CPUs, the similarities and dissimilarities between CPUs and GPUs are identified. Through model evaluation, this study provides a proof of concept that LLC-memory traffic can be predicted for sequential streaming and strided access patterns on GPUs.**

***Index Terms*—AMD; NVIDIA; Intel; GPU; Memory Access Pattern**

I. Introduction

After Dennard scaling [13] ended, heterogeneous systems have become the go-to solution for modern high performance computing (HPC) systems, and this trend is expected to be continued in the future [23]. Heterogeneity meets the diversified need of HPC users by hosting powerful CPUs and GPUs in the same node. However, it increases the complexity of programmability, hardware design, and the role of a runtime system. From a hardware design perspective, latency-focused CPUs are vastly different than throughput-focused GPUs. Moreover, instruction set architectures, programming models, and execution models are different. For this reason, finding similarities and identifying dissimilarities among CPUs and GPUs from different manufacturers can provide a better understanding of the performance. Since memory access is one of the critical parts of understanding performance, this study focuses on understanding the traffic between the Last level cache (LLC) and memory.

Cache hierarchy plays a significant role in deciding the compute and memory intensity of a kernel [24]. Since the traffic between LLC and memory is one of the slowest transactions when a kernel is in execution, exploring when and why an LLC-memory transaction takes place is essential to understand the performance on CPUs or GPUs [17]. Especially, GPUs need to be studied because the intense race between GPU manufacturers like NVIDIA and AMD to provide more computation power is propelling the release of new and more capable GPUs. For example, the most recent Ampere GPUs (A100) from NVIDIA [3] is countered by AMD's Instinct GPU (MI100) [1], where both of them host more than 30 GB of device memory (NVIDIA Ampere GPUs have more than double device memory from the previous Volta GPUs). To conquer the field of machine learning, both NVIDIA and AMD GPUs now have tensor cores and matrix cores, respectively. This competition is reflected in the large-scale supercomputers as well. For example, Frontier, the first exascale machine from Oak Ridge National Laboratory, will host AMD's Instinct GPUs, whereas Perlmutter, the NERSC supercomputer, hosts NVIDIA A100 GPUs. For this reason, it is high time to look closely to understand the impact of the memory hierarchies of different generations of NVIDIA and AMD GPUs.

Applications from different domains exhibit various memory access patterns [20]. The interplay between memory access patterns and the cache hierarchy needs to be explored to understand the role of the memory hierarchy. Generally, regular access patterns are more benefited by the cache hierarchy than irregular access patterns. For this reason, the design of the cache hierarchy is influenced by the common memory access patterns. Nowadays, all modern CPUs and GPUs employ prefetchers to facilitate regular access patterns. For example, sequential streaming access is mostly benefited by existing hardware prefetchers. However, there are other regular access patterns, such as strided and stencil access patterns [25]. Different access patterns need to be studied to understand and realize the benefit of the cache hierarchy fully.

Since CPUs have existed much longer than GPUs, the understanding in the research community is much higher for CPUs than the comparatively newer GPUs. Even though previous studies investigated the impact of memory access patterns in CPUs and GPUs, there have not been many studies that compare the LLC-memory traffic patterns between CPUs and GPUs [9], [18], [19]. Investigating similarities would provide opportunities to apply similar optimization techniques. Also, finding the dissimilarities would provide a better understanding of the differences between CPUs and GPUs. Therefore, it is essential to look for the similarities and dissimilarities between CPUs and GPUs.

This study adopts an experimental evaluation approach to explore and understand the impact of memory access patterns on different GPUs from NVIDIA and AMD and tries to find out the similarities and dissimilarities with Intel CPUs. Using the two most common memory access patterns, sequential streaming and strided access patterns, this study unveils the factors that decide LLC-memory transactions. In summary, the following contributions are reported.

- presenting the common factors in cache hierarchy that trigger an LLC-memory transaction;
- strategies to measure LLC-memory traffic by using NVIDIA's Nsight Compute and AMD's ROCm profiler;

- investigation and comparison of three NVIDIA GPUs, P100, V100, and A100, with Intel CPUs for two memory access patterns;
- investigation and comparison of three AMD GPUs, MI50, MI60, and MI100, to explore similarities and dissimilarities between Intel CPU and NVIDIA GPUs; and
- establishing a proof concept for predicting LLC-memory traffic of NVIDIA and AMD GPUs.

II. Background on Intel CPU

This section provides a background of the Intel Skylake CPU, inspired by our previous study [18]. A vector multiplication application that shows both sequential streaming (stride = 1) and strided access pattern (stride $>$ 1) is used. The number of LLC-memory transactions is presented for different strides (In this section, memory represents the system memory or DRAM). LLC-memory transactions for the Skylake CPU are measured using TAU [21] and PAPI [22]. Uncore counters are measured from the integrated memory controller of the Skylake CPUs [18]. These counters provide the count of LLC-memory transactions in the unit of the cache line, which is 64 Bytes in Intel CPUs, Skylake processor in this case. For this reason, total transactions are multiplied by 64 Bytes to calculate the total bytes transferred between LLC and memory. The findings demonstrated in this section also represent Cascade Lake and Cooper Lake micro-architectures of Intel.

A. Application and a lower bound of LLC-memory traffic

The vector multiplication application used to measure LLC-memory traffic has a separate $vecMul$ function to perform the multiplication. Only the $vecMul$ function's memory access is measured, leaving out the memory transfers while initializing the arrays and error checking. For every array index, two reads and one write take place in this function. All the graphs presented in this section show the LLC-memory bytes for array size of 100 Million 32 bit floating-points. Therefore, two read operations operate on 800 MBytes of data, and one write operation operates on a total of 400 MBytes of data. At least 1200 MBytes must be transferred between LLC and memory for the $vecMul$ function (when stride is 1), providing the lower bound of LLC-memory traffic. To observe the impact of sequential streaming and strided memory access patterns, the stride of $vecMul$ is varied from 1 to 8192. The memory traffic is plotted against the stride of $vecMul$. The main observations are presented in Fig. 1, 2, and 3.

B. Impact of initialization of the write array for stride of 1

To find out the impact of initialization, the write array of $vecMul$ is only allocated (malloc) but not initialized with any values. For this case, the read and write traffic is shown in black lines in Fig. 1. The read and write traffic for stride of 1 is close to the lower bound, 800 MBytes and 400 MBytes, respectively. Since many applications have the write arrays with some values, the write array is initialized with random values to observe the changes. The blue lines in Fig. 1 show the read traffic for stride 1 is 1200 MBytes (instead of 800 MBytes). However, the write traffic is 400 MBytes that follows the lower bound. The extra read traffic is incurring because Intel implements "allocating store" where a cache line is brought to cache from memory while performing a store. For this reason, the read traffic is three times the write traffic even though the actual read operations are twice as much as write operations.

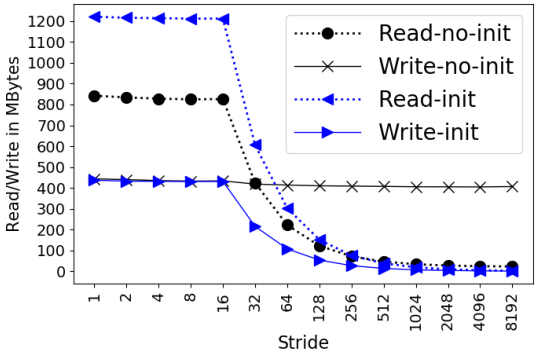


Figure 1: Read and write traffic for vector multiplication in Skylake processors when the write array is initialized and non-initialized.

C. Impact of the cache line size for strides greater than 1

The impact of the cache line size is visible in Fig. 1 for strides larger than 1. The LLC-memory traffic for the read traffic for initialized and non-initialized cases are portrayed by blue and black lines in Fig. 1. Even though stride is doubled (X-axis), traffic is the same until stride 16. This is because the $vecMul$ function operates on 32 bit floating-point data, and stride 16 equals the cache line length, which is 64 Bytes. Every time one array element is read, the entire cache line is transferred between LLC and memory. For this reason, traffic is the same until stride 16 and then reduces by half when the stride is doubled. The same is observed for the write traffic for the initialized case where write traffic is the same until stride 16 and then reduces by half. However, for the non-initialized case, the impact of the cache line is not visible because of "page-zeroing", which takes place in Intel processors to prevent information leakage. The Intel processor and the Linux operating system used for this experiment support "transparent huge pages" of 2 MBytes and 1 GBytes instead of default 4 KBytes pages. For this reason, the whole 2 MBytes pages are written for "page-zeroing", and the impact of the cache line is not visible (detail in [18]).

D. Prefetchers

The impact of enabling and disabling the prefetchers is shown in Fig. 2. Each core of Intel Skylake micro-architecture has four hardware prefetchers where two of which are L2 prefetchers. These prefetchers bring additional cache lines to cache to ensure better performance. Prefetchers only impact the read traffic. Read traffic is shown in Fig. 2 when prefetchers are enabled and disabled. Until stride 16, there is no visible difference; however, this is the region where prefetchers provide the most benefit since data is already made available

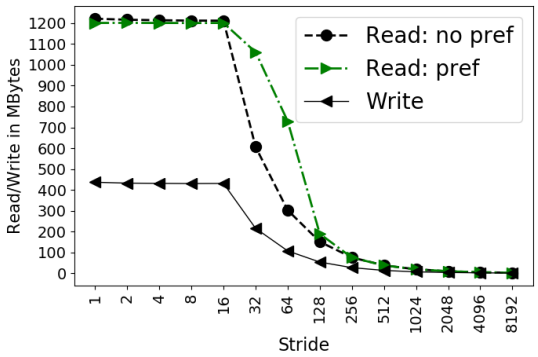


Figure 2: Read and write traffic when the prefetchers are disabled and enabled.

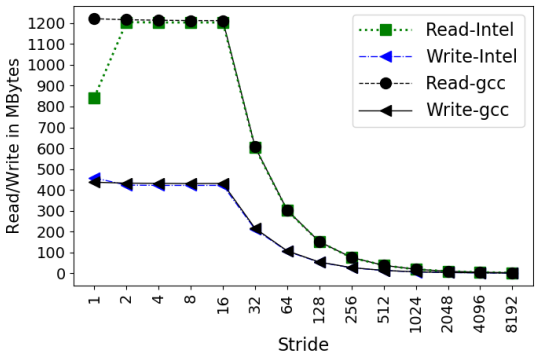


Figure 3: Read and write traffic comparison for Intel and GNU compilers.

in the cache for faster computation. The main difference is observed after stride of 16. Extra read traffic is observed from stride 16 to stride 128 when prefetchers are enabled. This is because the prefetching algorithm still triggers the prefetching operation even when there could be a little benefit since the stride size is much larger. However, after a particular stride, the prefetching and no prefetching cases are the same, which indicates that Intel prefetching algorithm allows prefetching up to a specific stride size.

E. Compiler

The impact of using different compilers is shown in Fig. 3. GNU and Intel compilers are compared. The only difference is observed when the stride size is 1. The Intel compiler shows the read traffic the same as the lower bound for the read traffic. Intel implements "streaming store" instead of "allocating store" for stride of 1. Intel compiler implements "streaming store" by default and can be turned off by using a compiler flag. However, the GNU compiler does not implement this feature by default. When "streaming store" is used, Intel processors use a write combining buffer to perform the store operation instead of fetching the cache line from memory. For this reason, the read traffic reduces for stride 1.

III. Methodologies for GPU access investigation

This section provides the methodologies to investigate LLC-memory traffic for sequential streaming and strided access patterns in NVIDIA and AMD GPUs. In this section, memory represents the device memory of the GPUs. At first, detailed hardware information of the NVIDIA and AMD GPUs used in this study is presented. Then, the application that exhibits sequential streaming and strided access patterns is presented (i.e., the strided $vecMul$ function for GPUs). Finally, the strategies to measure the LLC-memory traffic for different NVIDIA and AMD GPUs are discussed.

A. NVIDIA and AMD GPUs

NVIDIA and AMD have been releasing different GPGPUs for more than a decade. With the rejuvenation of machine learning, GPUs are getting even more attention. For this reason, there is an intense race between manufacturers to provide better performance for diverse workloads. The recent NVIDIA and AMD GPUs host tensor/matrix cores capable of performing fused matrix multiply and accumulate operations to facilitate machine learning workloads. This study considers three recent GPUs from both NVIDIA and AMD.

Table I: NVIDIA and AMD GPUs.

Item name	**NVIDIA Tesla GPUs**			**AMD Radeon GPUs**		
	Pascal: P100	***Volta: V100***	***Ampere: A100***	***Instinct: MI50***	***Instinct: MI60***	***Instinct: MI100***
Release year	2016	2017	2020	2018	2018	2020
Architecture	Pascal	Volta	Ampere	GCN 5.1	GCN 5.1	CDNA 1.0
Number of SMs/CUs	56	80	108	60	64	120
Number of Cores	3584	5120	6912	3840	4096	7680
Peak performance FP32 (float)	13.41 TFLOPS	14.13 TFLOPS	19.49 TFLOPS	13.41 TFLOPS	14.75 TFLOPS	23.07 TFLOPS
Peak performance FP64 (double)	6.705 TFLOPS	7.066 TFLOPS	9.746 TFLOPS	6.705 TFLOPS	7.373 TFLOPS	11.54 TFLOPS
Tensor/Matrix cores	No	Yes (640)	Yes (432)	No	No	Yes (each CU)
Device memory size	16 GB HBM2	16 GB HBM2	40 GB HBM2e	16 GB HBM2	32 GB HBM2	32 GB HBM
Memory Bus	4096 bit	4096 bit	5120 bit	4096 bit	4096 bit	4096 bit
Bandwidth	732.2 GB/s	900 GB/s	1555 GB/s	1,024 GB/s	1,024 GB/s	1,229 GB/s
L1 cache per SM/CU	24 KB	128 KB	192 KB	16 KB	16 KB	16 KB
L2 cache size	4 MB	6 MB	40 MB	4 MB	4 MB	8 MB
Cache line size	32 Bytes	32 Bytes	32 Bytes	64 Bytes	64 Bytes	64 Bytes
Warp/Wavefront size	32 threads	32 threads	32 threads	64 threads	64 threads	64 threads
Compiler	nvcc	nvcc	nvcc	hipcc	hipcc	hipcc
Profiler	nvprof	nvprof	Nsight compute	rocprof	rocprof	rocprof
Software stack	Cuda-11.0	Cuda-11.2	Cuda-11.2	rocm-3.9.0	rocm-4.3.0	rocm-4.3.0
Machine name	Oswald01	Leconte	Illyad	Gilgamesh	Explorer	Cousteau
Facility	ExCL	ExCL	OACISS	OACISS	ExCL	ExCL

Table I shows an overview of the hardware information of three NVIDIA Tesla and AMD Radeon GPUs, where the most recent are the A100 from NVIDIA and MI100 from AMD [1]–[6]. The machines used in this study are part of ORNL Experimental Computing Laboratory (ExCL) [8] and Oregon Advanced Computing Institute for Science and Society (OACISS) [7].

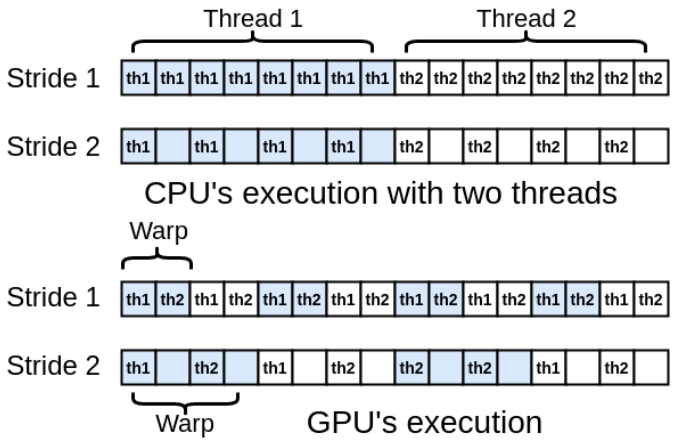


Figure 4: A simplistic representation of execution on CPU and GPU for strided access. Here, two threads are considered for CPU. For GPU only two threads per warp/wavefront is considered to show the difference (hypothetical). However, the warp/wavefront sizes are 32 and 64 threads in NVIDIA and AMD GPUs.

B. Strided vector multiplication for CUDA and ROCm

To investigate sequential streaming and strided access patterns, the vector multiplication application used for the CPU study [18] is modified for CUDA and ROCm platform. The basic difference between multi-threaded CPU code and GPU code is presented in Fig. 4. Programming model, such as OpenMP, divides a data structure among the available threads in a CPU where each thread continuously (based on stride size) executes the array indices. Multi-threaded execution for two threads is shown in Fig. 4. However, GPU decomposes the total computation in blocks consumed by warps/wavefronts in SM/CU. For this reason, blocks are usually chosen to be multiple of warp/wavefront size. Each warp/wavefront employs 32 (NVIDIA) and 64 (AMD) threads for computation.

To make the comparison more straightforward, the GPU execution shown in Fig. 4 displays a hypothetical situation where each warp has only two threads. So, the computation is done by hardware threads in a multi-threaded execution in CPUs (considering OpenMP) where the same thread processes the neighboring elements, which increases cache locality. On the other hand, warps/wavefronts consume thread blocks in GPU where different threads access adjacent data (depending on stride size). Because warps/wavefronts are scheduled in SM/CU, having the same warp threads working on neighboring data provides the best cache locality and can take advantage of memory coalescing in the L1 cache. For this reason, the vector multiplication application is modified in such a way so that neighboring threads in a warp/wavefront execute the adjacent elements (a standard practice [9]). Since this study focuses on the LLC-memory (L2-Global memory for GPUs), the impact on shared memory at L1 is not considered.

The modified CUDA code is shown in Listing 1 where neighboring threads execute neighboring elements for various strides. However, when the stride is 1, $vecMul$ exhibits a standard sequential stream access pattern like the STREAM benchmark [16]. Note that only the vectors read by $vecMul$ are allocated and initialized before transferring to the device (h_a and h_b). The write array is only allocated. While measuring the traffic, only $vecMul$ function is considered. The $hipify$ tool from the ROCm software stack is used to convert the CUDA code to HIP code. Since this is a small piece of code, the conversion using $hipify$ compiled without any error. For this study, $nvcc$ (cuda-11.0) and $hipcc$ (rocm-3.9.0 and rocm-4.3.0) compilers are used.

```
__global__ void vecMul(float *a, float *b, float *c,
     int n, int stride){
    // Get our global thread ID
    int id = blockIdx.x * blockDim.x + threadIdx.x;
    // Ensuring strided access and boundary checking
    if (id*stride < n)
        c[id*stride] = a[id*stride] * b[id*stride];
}

int main( int argc, char* argv[] ){
    int n = 100000000;
    // host and device data structures
    float *h_a, *h_b, *h_c, *d_a, *d_b, *d_c;
    size_t total_size = n*sizeof(float);
    // Allocate the vectors in the host
    h_a, h_b, h_c = allocate_host(total_size);
    // Initialize a and b vectors int the host
    for( int i = 0; i < n; i++ ) {
        h_a[i] = sin(i); h_b[i] = cos(i);
    }
    // Allocate the vectors in the device
    d_a, d_b, d_c = allocate_device(total_size);
    // Initiate host to device synchronous transfer
    copy_to_device(h_a, d_a, h_b, d_b)

    Start_memory_counters(); // done automatically
    by profiler
    vecMul<<<gridSize, blockSize>>>(d_a, d_b, d_c, n
    , stride);
    Stop_memory_counters(); // done automatically by
     profiler

    // Initiate device to host synchronous transfer
    copy_from_device(h_c, d_c);
    // free host and device memory
    free(h_a, h_b, h_c, d_a, d_b, d_c);
}
```

Listing 1: Strided vector multiplication in CUDA (this is not the exact code.)

C. Measuring LLC-memory traffic in GPUs

LLC-memory traffic is measured for the GPUs listed in Table I to investigate the impact of memory access patterns. NVIDIA's CUDA software stack provides $nvprof$, and Nsight Compute (ncu) for profiling GPU kernels. AMD's ROCm software stack provides ROCm profiler ($rocprof$) for the same

purpose for AMD GPUs. These tools are used in this study to gather LLC-memory traffic for the strided $vecMul$ function.

1) Using $nvprof$ and ncu for NVIDIA GPUs: Profiling tool $nvprof$ provides the functionality to measure hardware metrics for GPU kernels ($vecMul$ in this case). Unlike Intel CPU, where uncore counters are read from the integrated memory controller, $nvprof$ provides a direct metric for LLC-memory byte transfer. The name of the metrics are $dram\_read\_bytes$ and $dram\_write\_bytes$. Here, $dram$ indicates the device memory. These metrics are specified in the command line while the executable is attached to the $nvprof$. In some cases, the kernels are replayed multiple times to provide accurate LLC-memory bytes transferred. LLC-memory traffic for P100 and V100 is measured using $nvprof$. However, the support for $nvprof$ is discontinued for CUDA Compute Capability 8.0 and onward. For this reason, Nsight Compute (ncu) has been used for A100 where the metric names are $dram\_\_bytes\_read$ and $dram\_\_bytes\_write$ (one extra underline in the counter name).

2) Using $rocprof$ for AMD GPUs: AMD's ROCm profiler ($rocprof$) is used in this study to measure LLC-memory traffic for MI50, MI60, and MI100 GPUs. Like $nvprof$, the $rocprof$ command-line tool can measure basic hardware counters and derived metrics. Unlike $nvprof$, counters/metrics are specified in a file provided in the command line along with the executable. The value of the hardware counters/metrics is then generated and stored in a CSV file. Two derived metrics represent the LLC-memory traffic for AMD Instinct GPUs; they are $FETCH\_SIZE$ and $WRITE\_SIZE$. These metrics provide the traffic in the unit of KiB, which is converted to MBytes for an even comparison.

3) Scripts for data collection: To gather the data seamlessly, scripts are prepared both for NVIDIA and AMD GPUs to execute and collect the LLC-memory traffic. These scripts vary the stride size and collect the traffic, which is then plotted for analysis.

IV. Understanding NVIDIA and AMD GPUs

This section explores the similarities and dissimilarities of LLC-memory traffic between Intel CPU and GPUs from NVIDIA and AMD. LLC-memory traffic gathered from GPUs by following the methodologies presented in §III are compared to that of CPUs shown in §II. At first, three NVIDIA GPUs are investigated, followed by an exploration of AMD GPUs. All the graphs presented in this section have stride as X-axis and read/write traffic in MBytes in Y-axis. Through investigation, some key observations and hypotheses are formulated.

A. LLC-memory traffic of NVIDIA GPUs

Three NVIDIA GPUs from Pascal (P100), Volta (V100), and Ampere (A100) architectures are investigated.

1) Similarities between P100 GPU with Skylake CPUs: LLC-memory traffic for strided $vecMul$ on P100 is presented in Fig. 5. The blue lines represent the data for P100. When plotted, the traffic trend of P100 is found to be very similar to the CPU when the write data structure is initialized and compiled using the Intel compiler. It is noteworthy that $vecMul$ for GPU does not have the write array initialized. Even though the write array is uninitialized, P100 shows a similar trend as CPU with allocating store. The CPU traffic is presented using green lines (the same data from Fig. 3). The main difference observed, in this case, is for the stride of 8 and 16. This difference is because of the cache line sizes in NVIDIA GPU (32 Bytes) and Intel CPUs (64 Bytes). The following observations can be made from Fig. 5.

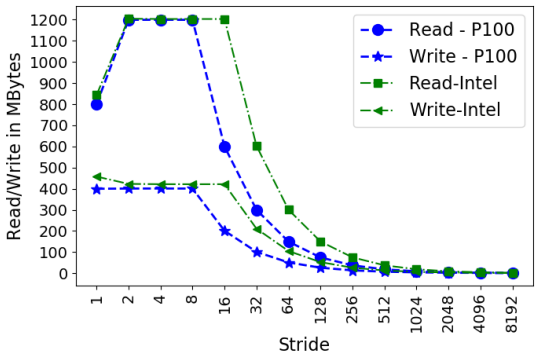


Figure 5: Read and write traffic for P100 GPU. Traffic of P100 follows similar trend of Skylake CPUs when write data structure is initialized and Intel compiler is used.

Observation-1 Like Intel compiler, $nvcc$ compiler implements streaming-store operation when stride is 1.

Observation-2 NVIDIA GPU performs allocating store even when the data structure is not initialized.

Observation-3 The write traffic of NVIDIA GPU is the same as the write traffic for the initialized case of the Skylake processors.

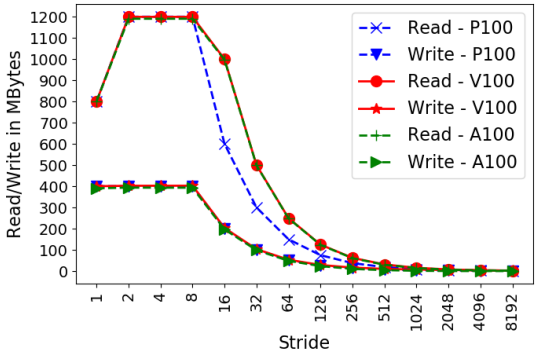


Figure 6: Read and write traffic for all GPUs

2) Similarities between P100, V100, and A100: Traffic from all NVIDIA GPUs are shown in Fig. 6. It is visible that the write traffic for all the GPUs is the same. Moreover, all the GPUs implement streaming store and allocating store. Therefore, observations 1, 2, and 3 are also applicable for V100 and A100 by following the transitive property. For both read and write traffic, V100 and A100 show little to no difference. The main difference between P100 and later GPUs is observed for the read traffic when the stride size is larger than the cache line size (stride of 8). On average, the read traffic in V100 and A100 is $1.6\times$ higher than that of

P100. When the stride size is larger than the cache line size, the difference between P100 and A100 (also V100) shows a striking similarity to the prefetching enabled and disabled cases for Intel CPUs presented in Fig. 2. This difference suggests that there has been a major change in the prefetchers of Volta architecture and onward. Therefore, the following observation and hypothesis can be made.

Observation-4 When the stride size is larger than the cache line on V100 and A100 GPUs, the read traffic shows a similar pattern of prefetching-enabled Skylake processor. Traffic is about $1.6\times$ higher than P100 and prefetching-disabled Skylake processor.

Hypothesis-1 From Volta architecture and onward, NVIDIA GPUs implement Intel CPU-like prefetching mechanism.

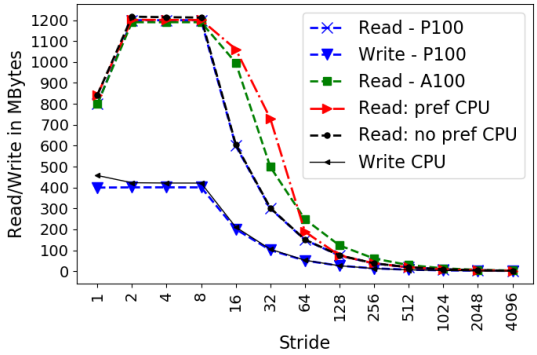


Figure 7: A tailored comparison between GPUs and CPU

3) A tailored graph to realize the similarities: A hypothetical scenario is considered where the cache line size of Skylake CPU is 32 Bytes instead of 64 Bytes to demonstrate the similarities between Intel CPU and NVIDIA GPUs. Even though the consideration is hypothetical, the data presented in Fig. 7 are actual data. To prepare Fig. 7, the read and write traffic for stride 16 is removed only for CPU and shifted left for all the strides greater than 16. This conversion shows the CPU data for a 32 Byte cache line because one read or write would fetch/store 32 Bytes instead of 64 Bytes for one access. The highest stride shown in Fig. 7 is 4096 instead of 8192. After this conversion, the write traffic for all GPUs and CPU shows a similar data and trend. The similarity is observed between the read traffic for P100 and CPU with prefetching disabled (overlapped blue and black lines in Fig. 7 for the read traffic).

All GPUs and CPU in Fig. 7 show similar data for the read traffic until stride of 8. A100 (V100 as well) and prefetching-enabled CPU show the same trend after stride size of 8 (denoted by green and red lines in Fig. 7). However, there is no overlap between the green and red lines, and the CPU reports higher read traffic than the A100 GPUs till stride 64, then the opposite is observed. The CPU keeps prefetching until the stride of 80 (not shown in the figure and determined experimentally). On the contrary, GPU keeps prefetching even for higher strides where there should not be any benefit for such action since the memory accesses are more than four cache lines apart. In summary, it can be said that there are more similarities than dissimilarities. Therefore, the following hypothesis can be formulated.

Hypothesis-2 Model prepared to predict LLC-memory traffic for sequential streaming and strided access pattern for an Intel Skylake CPU with a cache line size of 32 Bytes can be customized to predict LLC-memory traffic for NVIDIA GPUs.

B. LLC-memory traffic of AMD GPUs

In contrast to NVIDIA GPUs, AMD GPUs considered in this study are comparatively newer. MI50 and MI60 instinct GPUs have the same GCN 5.1 architecture, whereas MI100 adopts CDNA 1.0 architecture. Since these GPUs are released within two years, a high similarity is expected.

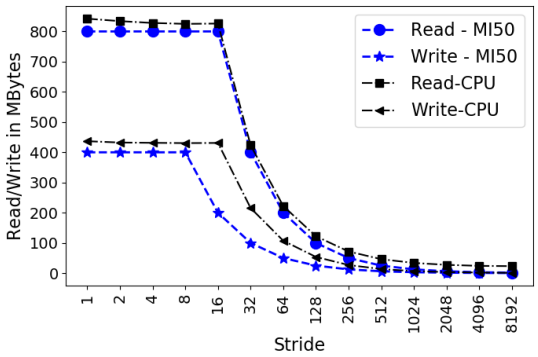


Figure 8: Read and write traffic for MI50 GPU. A comparison with Intel Skylake CPU.

1) Similarities between MI50 GPU with Skylake CPU: The LLC-memory traffic for MI50 is shown in Fig. 8. The read and write traffic for stride one is very close to the theoretical lower bound. The read traffic in Fig. 8 for MI50 is found to be following the same trend as the Skylake CPU when the data structure is not initialized (shown initially at Fig. 1). The $vecMul$ function for GPU does not have the write array initialized; so, unlike NVIDIA GPUs, MI50 does not implement allocating store, which is more appropriate for this function because there is no value in the write array and bringing the cacheline from memory is unnecessary.

Observation-5 AMD GPUs do not implement allocating store for uninitialized write array.

Even though the read traffic shows similarity with the non-initialized case, the write traffic in Fig. 8 shows the same trend as the write traffic for the initialized case. Hence, it proves that page zeroing like CPU is not occurring even though the write array is not initialized.

The most interesting observation in Fig. 8 is that a drop is observed for the read traffic when the stride is 16 but for the write when the stride is eight. This difference suggests that the cache line length is 64 Bytes for the read transactions and 32 Bytes for the write transactions. Such a scenario is not observed in Intel CPUs or NVIDIA GPUs. To confirm the cache line length, $rocminfo$ command is used in all the AMD GPUs where the cache line length is shown to be 64 Bytes. The write traffic dropping from stride of 8 instead of 16 even though the cache line length is 64 Bytes needs to be investigated.

To investigate this anomaly, the detail of the metric used to measure the write traffic is explored. The formula of the metric is $WRITE\_SIZE = (TCC\_MC\_WRREQ\_sum * 32)/1024$ (this formula is found from the metrics.xml file inside the rocprofiler directory of ROCm software stack). The explanation for this metric is the following, "The total kilobytes fetched from the video memory. This is measured with all extra fetches and any cache or memory effects taken into account." For further investigation, $TCC\_MC\_WRREQ\_sum$ metric is explored where the formula is $sum(TCC\_MC\_WRREQ, 16)$. The description of the metric is "Number of 32-byte transactions going over the TC_MC_wrreq interface. Sum over TCC instances." Therefore, this description confirms that AMD GPUs perform 32 Byte transactions for write traffic, and the write traffic in AMD GPUs follows the same trend as NVIDIA GPUs.

Observation-6 AMD GPUs show 64 Byte LLC-memory transactions for read and 32 Byte transactions for write traffic.

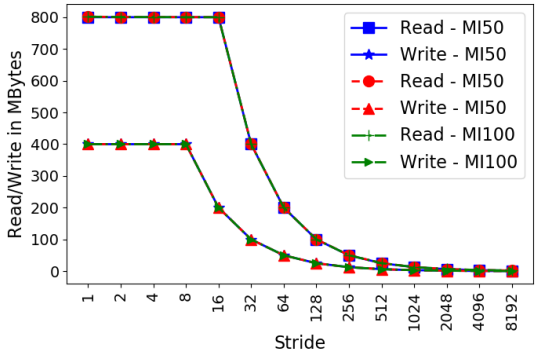


Figure 9: Read and write traffic for all AMD GPUs

2) Similarities between MI50, MI60, and MI100: The LLC-memory traffic for MI50, MI60, and MI100 are shown in Fig. 9. All the AMD GPUs show similar trends and data. Therefore, observations 4 and 5 apply to all the AMD GPUs (following the transitive property). So, the following hypothesis can be made.

Hypothesis-3 Because there are many similarities, a model prepared to predict LLC-memory traffic for sequential streaming and strided access patterns for an Intel Skylake CPU can be customized to predict LLC-memory traffic for AMD Instinct GPUs.

C. Comparison of the profiling tools

This study found that NVIDIA's ncu tool provides more hardware counters and metrics for GPU execution when compared to the $rocprof$. However, the detailed formula behind a metric can be found in $rocprof$, which is helpful to investigate further. Moreover, $rocprof$ provides the facility to write custom-derived metrics. From an accuracy standpoint, both ncu and $rocprof$ provided a reasonably accurate traffic count close to the theoretical traffic count for lower strides. Unfortunately, this is not the case for CPUs. Our previous study observed some extra traffic while measuring CPUs. One common problem found for all the cases is that when the number of accesses is low (i.e., higher strides), the traffic count is not accurate compared to the theoretical traffic. This study suspects that the inaccuracies in higher strides come from having low memory access and a smaller lifespan of an application. A smaller lifetime of the application makes pinpointing the memory traffic of a function difficult for the profiler. While modeling and comparing accuracy, this fact needs to be taken into consideration.

D. Discussion about the hypotheses

Three hypotheses are formulated in this study. To prove Hypothesis-1, one would need to study the details of the changes in the prefetching algorithm from Pascal to Volta architecture of NVIDIA. However, hypotheses 2 and 3 can be verified by preparing a prediction model.

V. Experiment and Prediction: A proof of concept

In this section, a prediction model is formulated to test hypotheses 2 and 3. The model is evaluated for different input sizes for the $vecMul$ function in the NVIDIA and AMD GPUs. In §II and §IV the input size used is 100M 32 bit floating-point data. However, input sizes of 50M and 200M 32 bit floating-point data are evaluated in this section. The predicted and measured total traffic are compared to find out the prediction error. Relative accuracy is considered to determine the error, where $error = Absolute[(measured - predicted)/measured * 100]$, and the formula for accuracy is $accuracy = [100 - error]$. While experimenting, it is observed that NVIDIA Nsight Compute does not generate data for smaller data sizes with less computation (also explained in §IV-C). For this reason, strides considered in this evaluation are from 1 to 1024.

A. Prediction model for NVIDIA GPUs

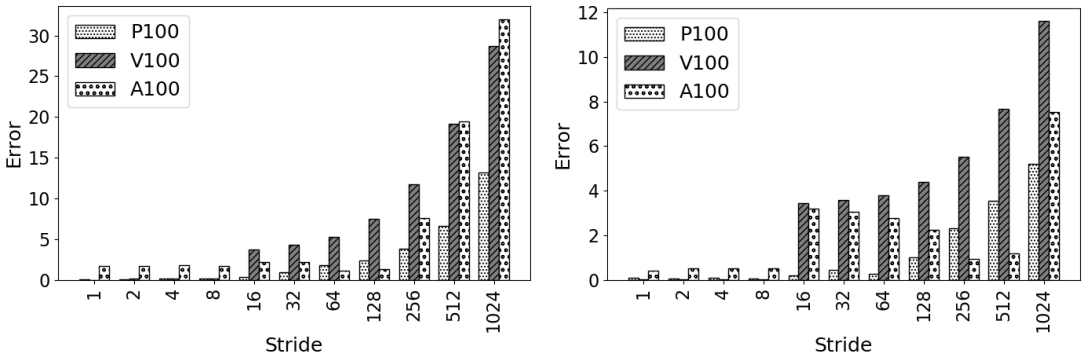
The prediction strategy for NVIDIA GPUs are presented in Table II. Table II incorporates the observations 1-4 that are reported in §IV. Here, "stream" stands for the calculated theoretical traffic for one data structure. For example, for a 100M 32 bit floating-point data structure size, stream = 400 MBytes. To reference the cell above, "prev" is used.

Table II: Prediction for NVIDIA GPUs.

Stride	Read P100	Read V100/A100	Write for all
1	stream * 2	stream * 2	stream
2	stream * 3	stream * 3	stream
4	stream * 3	stream * 3	stream
8	stream * 3	stream * 3	stream
16	prev/2	prev/2 * 1.6	prev/2
32	prev/2	prev/2	prev/2
64	prev/2	prev/2	prev/2
128	prev/2	prev/2	prev/2
256	prev/2	prev/2	prev/2
512	prev/2	prev/2	prev/2
1,024	prev/2	prev/2	prev/2

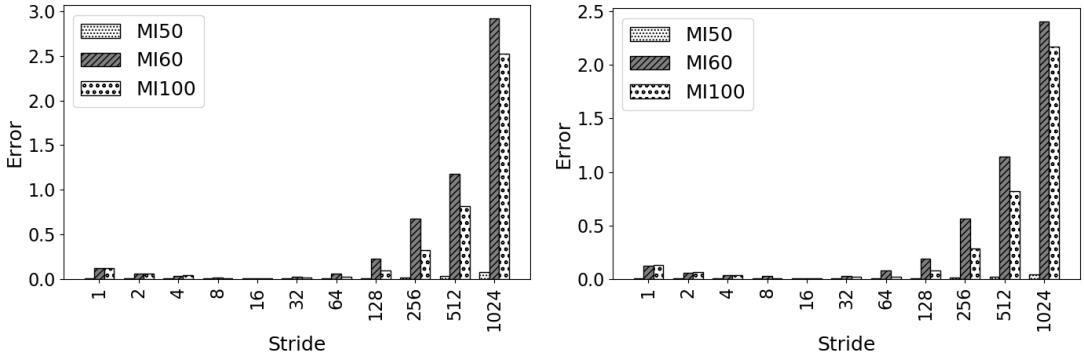
B. Prediction accuracy for NVIDIA GPUs

The strength of the model presented in Table II is depicted in Fig. 10. For the input size of 50M, high average accuracy is observed for NVIDIA GPUs (P100 showed 97.3% accuracy,



(a) Prediction accuracy for input size of 50M. (b) Prediction accuracy for input size of 200M.

Figure 10: Prediction accuracy for NVIDIA GPUs for different input sizes.



(a) Prediction accuracy for input size of 50M. (b) Prediction accuracy for input size of 200M.

Figure 11: Prediction accuracy for AMD GPUs for different input sizes.

V100 showed 92.6% accuracy, and A100 showed 93.4% accuracy). However, it is visible that with higher strides, the error increases. When the 200M input size is used, the average accuracy increased (P100 showed 98.8% accuracy, V100 96.4% accuracy, and A100 showed 97.9% accuracy). This increased accuracy for larger data sizes confirms that the profiler cannot report exact memory traffic for application with short life span which operates on smaller data sizes. Another observation can be made from Fig. 10b. From stride 16 and onward, V100 and A100 report a higher error. This inaccuracy stems from using a factor of $1.6\times$ to capture the impact of the prefetchers in the newer NVIDIA GPUs. Therefore, more understanding of the prefetchers of NVIDIA GPUs is needed to improve the model's accuracy. However, Fig. 10b still provides more than 88% accuracy for all the cases.

C. Prediction model for AMD GPUs

The prediction strategy for AMD GPUs is presented in Table III. Table III incorporates the observations 5 and 6 that are reported in §IV. Since all the AMD GPUs in this study follows a similar pattern, the same model is used for all.

D. Prediction accuracy for AMD GPUs

The error of the model presented in Table III is presented in Fig. 11. AMD GPUs provided higher accuracy than the NVIDIA GPUs. For the input size of 50M, higher average accuracy is observed for AMD GPUs (MI50 showed 99.98% accuracy, MI60 showed 99.5% accuracy, and MI100 99.6% accuracy). Even higher inaccuracies are observed 200M input size (MI50 showed 99.9% accuracy, MI60 showed 99.6% accuracy, and MI100 99.7% accuracy). For both input sizes, error increased for higher strides; however, all strides provided more than 97% accuracy.

Table III: Prediction for AMD GPUs.

Stride	Read for all	Write for all
1	stream * 2	stream
2	stream * 2	stream
4	stream * 2	stream
8	stream * 2	stream
16	stream * 2	prev/2
32	prev/2	prev/2
64	prev/2	prev/2
128	prev/2	prev/2
256	prev/2	prev/2
512	prev/2	prev/2
1,024	prev/2	prev/2

E. Discussion

The seemingly simplistic model generated higher accuracy than the CPU prediction model presented in our previous study [18]. Sequential streaming and strided memory access patterns are common in large applications. Model for other access patterns, such as stencil, can be derived from using these two patterns. Therefore, proof of concept for these two

patterns opens the door for predicting LLC-memory traffic for larger applications.

VI. Related Works

Memory access patterns play an important role in deciding the performance of an application running on GPUs [19]. Two studies delved into sequential streaming and strided access patterns. Allen et al. investigated the impact of memory access patterns on power and performance on GPUs [9]. Their focus was to understand the impact on attained bandwidth and average power. Ding et al. proposed instruction Roofline model for GPUs [14]. Even though the study focuses on different instructions, the authors looked into the impact of sequential streaming and strided memory access patterns to define the theoretical upper bound. Unlike these studies, we effort to understand the memory transactions that take place between LLC and memory. Other studies also focused on understanding the impact of memory accesses on performance and power on GPU [12], [15]. Ben-Nun et al. investigated different memory access patterns for multi-GPU scenario [10]. The main objective of their work was to provide task partitioning and device-level optimization for a multi-GPU environment. Our study considered regular access patterns with no possibility of bank conflict at the shared memory for a warp/wavefront. This is the best case for performance. However, multiple threads in a warp can access the same bank for irregular application, causing bank conflict that impacts performance. Burtscher et al. investigated such irregular applications on GPUs [11].

Our study differs from these studies because the main objective of our investigation is to separate the LLC-memory traffic from other observations such as execution time, attained bandwidth, or energy consumption. This separation allows us to find the apparent similarities between CPUs and GPUs.

VII. Conclusion and Future work

This study investigates the impact of sequential streaming and strided memory access patterns on different NVIDIA and AMD GPUs. By presenting the observation on Intel Skylake CPU, this study attempts to identify the similarities and dissimilarities in LLC-memory traffic on different generations of NVIDIA and AMD GPUs. Through investigations, some key observations and hypotheses are made. Models are prepared by incorporating those key observations. Experimental evaluation shows that the LLC-memory traffic can be predicted for different memory access patterns. Studying large applications with complex memory access patterns and investigating Intel GPUs are in the future plan.

References

[1] Amd cdna architecture. https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf. Accessed: 2021-09-01.

[2] Amd radeon instinct gpus. https://en.wikipedia.org/wiki/Radeon_Instinct. Accessed: 2021-09-01.

[3] Nvidia tesla a100 white paper. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf. Accessed: 2021-09-01.

[4] Nvidia tesla gpus. https://en.wikipedia.org/wiki/Nvidia_Tesla. Accessed: 2021-09-01.

[5] Nvidia tesla p100 white paper. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf. Accessed: 2021-09-01.

[6] Nvidia tesla v100 white paper. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Accessed: 2021-09-01.

[7] Oregon advanced computing institue for science and society (oaciss). https://blogs.uoregon.edu/oaciss/. Accessed: 2021-09-01.

[8] Ornl experimental computing laboratory (excl). https://excl.ornl.gov/. Accessed: 2020-09-01.

[9] T. Allen and R. Ge. Characterizing power and performance of gpu memory access. In *2016 4th International Workshop on Energy Efficient Supercomputing (E2SC)*, pages 46–53. IEEE, 2016.

[10] T. Ben-Nun, E. Levy, A. Barak, and E. Rubin. Memory access patterns: The missing piece of the multi-gpu puzzle. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2015.

[11] M. Burtscher, R. Nasre, and K. Pingali. A quantitative study of irregular programs on gpus. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*, pages 141–151. IEEE, 2012.

[12] G. Chen, B. Wu, D. Li, and X. Shen. Porple: An extensible optimizer for portable data placement on gpu. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 88–100. IEEE, 2014.

[13] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[14] N. Ding and S. Williams. An instruction roofline model for gpus. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 7–18. IEEE, 2019.

[15] S. Hong and H. Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 152–163, 2009.

[16] J. D. McCalpin. Stream benchmarks, 2002.

[17] M. Monil, M. Belviranli, S. Lee, J. Vetter, and A. Malony. Mephesto: Modeling energy-performancein heterogeneous socs and their trade-offs. In *Proceedings of the 29th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2020.

[18] M. Monil, S. Lee, J. Vetter, and A. Malony. Understanding the impact of memory access patterns in intel processors. In *MCHPC'20: Workshop on Memory Centric High Performance Computing*. IEEE, 2020.

[19] I. Paul, W. Huang, M. Arora, and S. Yalamanchili. Harmonia: Balancing compute and memory power in high-performance gpus. *ACM SIGARCH Computer Architecture News*, 43(3S):54–65, 2015.

[20] I. B. Peng, R. Gioiosa, G. Kestor, J. S. Vetter, P. Cicotti, E. Laure, and S. Markidis. Characterizing the performance benefit of hybrid memory system for HPC applications. *Parallel Computing*, 76:57–69, 2018.

[21] S. Shende and A. Malony. The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[22] D. Terpstra, H. Jagode, H. You, and J. Dongarra. Collecting performance data with papi-c. In *Tools for High Performance Computing 2009*, pages 157–173. Springer, 2010.

[23] J. Vetter and et. al. Extreme heterogeneity 2018-productive computational science in the era of extreme heterogeneity: Report for doe ascr workshop on extreme heterogeneity. Technical report, USDOE Office of Science (SC), Washington, DC (United States), 2018.

[24] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[25] L. Yu, D. Li, S. Mittal, and J. S. Vetter. Quantitatively modeling application resiliency with the data vulnerability factor. *ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2014.