Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects

Carl Pearson, Abdul Dakkak,
Sarah Hashash, Cheng Li
University of Illinois
Urbana-Champaign
Urbana, Illinois
{pearson,dakkak}@illinois.edu
{hashash2,cli99}@illinois.edu

I-Hsin Chung, Jinjun Xiong
IBM T. J. Watson Research Center
Yorktown Heights, New York
{ihchung,jinjun}@us.ibm.com

Wen-mei Hwu
University of Illinois
Urbana-Champaign
Urbana, Illinois
w-hwu@illinois.edu

ABSTRACT

Data-intensive applications such as machine learning and analytics have created a demand for faster interconnects to avert the memory bandwidth wall and allow GPUs to be effectively leveraged for lower compute intensity tasks. This has resulted in wide adoption of heterogeneous systems with varying underlying interconnects, and has delegated the task of understanding and copying data to the system or application developer. No longer is a malloc followed by memcpy the only or dominating modality of data transfer; application developers are faced with additional options such as unified memory and zero-copy memory. Data transfer performance on these systems is now impacted by many factors including data transfer modality, system interconnect hardware details, CPU caching state, CPU power management state, driver policies, virtual memory paging efficiency, and data placement.

This paper presents Comm|Scope, a set of microbenchmarks designed for system and application developers to understand memory transfer behavior across different data placement and exchange scenarios. Comm|Scope comprehensively measures the latency and bandwidth of CUDA data transfer primitives, and avoids common pitfalls in ad-hoc measurements by controlling CPU caches, clock frequencies, and avoids measuring synchronization costs imposed by the measurement methodology where possible. This paper also presents an evaluation of Comm|Scope on systems featuring the POWER and x86 CPU architectures and PCIe 3, NVLink 1, and NVLink 2 interconnects. These systems are chosen as representative configurations of current high-performance GPU platforms. Comm|Scope measurements can serve to update insights about the relative performance of data transfer methods on current systems. This work also reports insights for how high-level system design choices affect the performance of these data transfers, and how developers can optimize applications on these systems.

CCS CONCEPTS

• **General and reference** → **Measurement**; **Performance**; • **Hardware** → **Buses and high-speed links**;

KEYWORDS

CUDA, GPU, NVLink, Benchmarking, POWER, x86, NUMA

ACM Reference Format:
Carl Pearson, Abdul Dakkak, Sarah Hashash, Cheng Li, I-Hsin Chung, Jinjun Xiong, and Wen-mei Hwu. 2019. Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3297663.3310299

1 INTRODUCTION

Multi-GPU systems are increasinglycommon and pivotal in tackling problems arising in big data, machine learning, and HPC. Today, a variety of systems are available for cloud usage [8, 16, 26] and HPC clusters [1, 2, 25] with 2-, 4-, 8-, or 16-GPUs connected via varying interconnect typologies. However, there is currently a lack of tools that facilitate comprehensive understanding of the behavior of such systems under different data placement and communication scenarios.

Comprehensive understanding demands comprehensive measurement across transfer types and sizes, and a systematic methodology, since ad-hoc data transfer measurements — frequently carried out in the course of application design — can have common pitfalls. This paper proposes both a new benchmark suite and describes a methodology for systematic measurement of data transfers. More specifically, this paper makes the following contributions:

- Presents a comprehensive set of latency and bandwidth microbenchmarks for NUMA-aware point-to-point CPU-GPU and GPU-GPU bulk transfer primitives in Nvidia's CUDA, including analogous zero-copy and unified memory operations.
- Systematically describes and addresses measurement pitfalls: describing the effects of CPU caches, CPU clock speed, and synchronization costs of measuring bandwidth.
- Evaluates intra-node communication on representative high-performance nodes and interconnects.Describes high-performance uses of implicit zero-copy and unified memory transfers.
- Provides insights for system designers on how to architect next-gen hardware and developers on how to orchestrate copies to optimize applications for current systems.

ICPE '19, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3310299

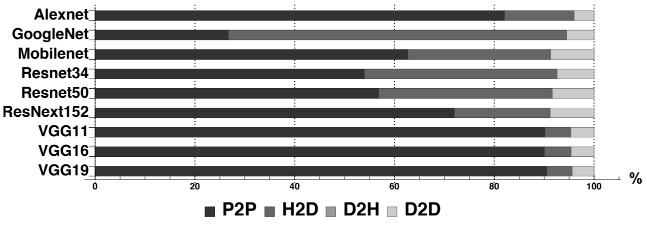


Figure 1: Percentage of communication time for a single epoch of network training using MXNet (P: Peer, H: Host, D: Device). Peer refers to direct memory access between GPUs. Note that for some networks communication time is dominated by the host-to-device memory copy.

To motivate the need for understanding intra-node communication performance – consider CUDA unified memory. Conventional wisdom about CUDA communication does not necessarily hold up on modern systems with fast interconnects. For example, many developers are leery of using unified memory due to perception of its poor performance. While it is slower than the fastest explicit transfers, this work shows that it is actually faster than the traditional `cudaMalloc`/`cudaMemcpy` pair (Section 4). Detailed measurements also show that zero-copy memory may be the preferable way to move data between GPUs, and that `cudaMalloc`/`cudaMemcpy` is limited by single-threaded CPU performance (Section 4). These findings show that data transfer performance is more than just a function of the underlying interconnect technology.

Existing works fall short in covering all aspects of GPU communication patterns (Shown in Table 1) or do not measure current state-of-the-art interconnects. The existing works that measure current state-of-the-art interconnects [6, 10, 15, 22, 24, 30] concentrate on the interplay between peer GPU and inter-node communication — omitting the host-to-device and device-to-host communication, which we observe to have a substantial impact in AI and HPC applications. For example, Figure 1 shows the communication time breakdown for training a deep neural networks using the MXNet [14] framework with the sizes of data transfers ranging from bytes to hundreds of megabytes. As can be seen, there is a mixture of transfer types — with host-to-device transfers potentially taking a substantial amount of the communication time.

The paper is organized as follows: Section 2 motivates Comm|Scope and describes the common pitfalls of benchmarking GPU communication. Section 3 describes the microbenchmarks in Comm|Scope and implementation decisions. In Section 4, we evaluate three current systems using Comm|Scope and present the observations. Section 5 describes related work before we conclude in Section 6 and present future work in Section 7.

2 DESIGN OBJECTIVES

Comm|Scope's microbenchmarks are designed to (1) measure the bandwidth and latency of all CUDA point-to-point bulk transfer primitives while (2) systematically addressing some common pitfalls in prior measurements. This section describes these two pillars of Comm|Scope design and how those design goals are met. We hope that these benchmarks will serve as a definitive approach for measuring point-to-point primitive transfers in CUDA systems and serve as a foundation for other CUDA measurement tools.

Host Allocation	Device Allocation	Transfer	Kind	Our	[30]	[10]	[24]	[15]	[22]	[6]
NUMA / pageable	cudaMalloc	explicit	H2D	✓						
			D2H	✓						
			bi	✓						
pageable	cudaMalloc	explicit	H2D	✓				✓	✓	
			D2H	✓				✓	✓	
			bi	✓						
NUMA / pinned	cudaMalloc	explicit	H2D	✓		✓				
			D2H	✓						✓
			bi	✓						
pinned	cudaMalloc	explicit	H2D	✓		✓		✓		✓
			D2H	✓		✓		✓		✓
			bi	✓						✓
mapped	–	implicit (zero-copy)	H2D	✓		✓				
–	cudaMalloc	implicit (zero-copy)	D2D	✓		✓				
			D/D bi	✓		✓				
–	cudaMalloc	explicit / peer	D2D	✓	✓	✓	✓			✓
			bi	✓		✓	✓			✓
–	cudaMalloc	explicit / no peer	D2D	✓	✓		✓			✓
			bi	✓			✓			
cudaMallocManaged		demand page migration	H2D	✓					✓	
			D2H	✓					✓	
			H/D bi	✓						
			D2D	✓						
			D/D bi	✓						
cudaMallocManaged		prefetch	H2D	✓						
			D2H	✓						
			H/D bi	✓						
			D2D	✓						
			D/D bi	✓						

Table 1: Comm|Scope data transfer microbenchmark coverage, and summary of where related work overlaps. Comm|Scope defines bandwidth benchmarks for all unidirectional and bidirectional primitive CUDA point-to-point transfers.

2.1 Comprehensive Communication Coverage

The first pillar of Comm|Scope design is to provide comprehensive coverage of CUDA communication primitives. Previous works provide fragmented and incomplete coverage of CUDA communication performance (Table 1), and do not measure prefetch performance of unified memory or NUMA-aware pageable explicit transfers. This section describes the principles Comm|Scope uses to cover all point-to-point CUDA communication paradigms.

2.1.1 *Unidirectional Operations.*

Comm|Scope provides microbenchmarks for unidirectional operations. When asynchronous operations exist, elapsed time is measured using CUDA events to avoid introducing additional synchronization costs. When operations are synchronous with the host (as in unified memory or zero-copy transfers involving the host), the host wall-clock time is used to measure events.

2.1.2 *Bidirectional Operations.*

Comm|Scope uses multiple simultaneous independent asynchronous unidirectional transfers to measure aggregate bidirectional transfer performance. Independent operations are queued in separate CUDA streams. To time these independent operations, a pair of CUDA events is created for each operation in each stream. The measured time is between the beginning of the first operation to the end of the last operation. This avoids introducing additional synchronization between operations. When streams must be associated with separate devices, CUDA events cannot be synchronized across devices. In this circumstance, an unavoidable synchronization to ensure the operation is complete gets included in the elapsed time. As shown in Table 1, prior works frequently fail to measure bidirectional transfer bandwidth.

2.1.3 NUMA Pinning. Comm|Scope measures the NUMA effect on multi-GPU and multi-CPU systems through libnuma [31]. When NUMA control is not utilized on NUMA systems, there can be extreme variability in the measured performance depending on how data is placed on the underlying system. Some prior works [10, 15, 22] that evaluate CPU/GPU transfers do not address NUMA effects on bandwidth.

2.1.4 Peer Access. Comm|Scope defines GPU-to-GPU transfer benchmarks between GPUs that support peer access. As host transfers implicitly exist when peer access is not available, Comm|Scope also defines GPU-to-GPU transfer benchmarks with peer access explicitly disabled, and with NUMA pinning, even though no host allocation is explicitly performed. Peer access allows CUDA GPUs to directly access each others' memory without involving the CPU. When peer access is not available, device-to-device transfers implicitly involve host memory: a non-peer GPU-to-GPU transfer must first pass the data to the host, which then passes it to the destination GPU. This is managed transparently by the CUDA software stack, so no host memory is exposed to the user and the benchmark cannot flush any cached host memory. Comm|Scope defines GPU-to-GPU transfer benchmarks between GPUs with and without peer access support.

2.1.5 CUDA Zero-copy. Comm|Scope's benchmarks define zero-copy operations(Section 3.3) that are analogous to the CUDA explicit bulk transfers. CUDA includes a unified address space, where the host and device can access data through the same pointer. In the zero-copy paradigm, pinned allocations on the host can be mapped into this address space and directly accessed from device kernels. Likewise, data allocated on one device may be directly accessed from another device. These accesses are served over the device interconnects without any explicit control from the programmer. The performance of zero-copy transfers depends on the degree of parallelism, the access pattern, and the interconnect speed. Comm|Scope enables study of zero-copy performance.

2.1.6 CUDA Unified Memory. Like the zero-copy paradigm, CUDA Unified Memory [18] allows allocations to be used on both the host and device without any explicit transfers. Using unified memory requires no "boilerplate" code to transfer data back and forth between the CPU and GPU, or between GPUs - it guarantees a coherent view of data across all participating system components. Unified memory also provides automatic communication/computation overlap, as a GPU thread waiting on unified memory data can be suspended while other GPU threads are executed. Finally, unified memory can allow system atomics between the CPU and GPU when supported.

In unified memory, each page has a "home" location. The home location of the page is dynamically controlled by CUDA based on which devices access it most frequently, thereby allowing those accesses to usually be served from the device memory instead of over the interconnect. A full examination of unified memory data accesses depends on the underlying hardware, system software, CUDA driver hints, access pattern, degree of CPU/GPU memory system integration, and parallelism, and is not covered in this work. Comm|Scope defines a focused set of unified memory benchmarks that measure the speed of page migration for regions of various sizes between devices. This migration scenario is analogous to the bulk migrations that occur though the explicit data-transfer primitives. These page migrations can be caused on-demand when the CUDA system determines that the home location of the page should change, or following a hint from the application. Comm|Scope measures both scenarios (Section 3.4). The measured bandwidth may include the cost of page faults, control signals, and the actual data movement, but should accurately reflect the application-visible data movement performance.

2.2 Common Pitfalls

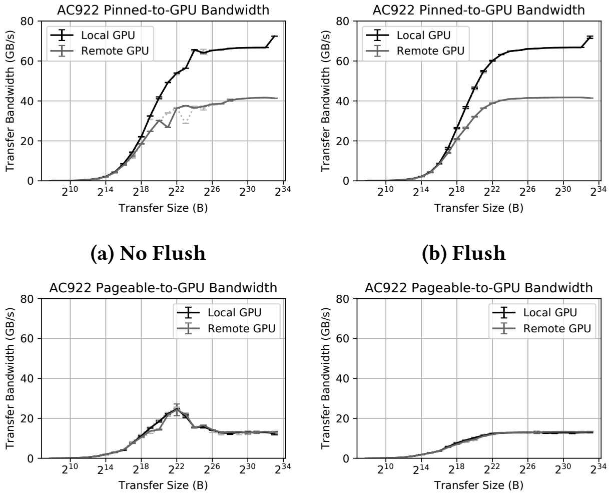
The second pillar of Comm|Scope design is to avoid measurement pitfalls present in prior CUDA communication benchmarks as well as the ad-hoc communication measurements made during application design.

2.2.1 Synchronous Operations. Some prior works [15] measure CUDA data transfer performance by using host [1] wall-clock time with synchronous CUDA operations, or asynchronous CUDA operations followed by `cudaDeviceSynchronize`. They end up incorrectly including two unknown times in their measurement: the time between the function call and the start of the operation, and the time between the end of the operation and the end of the synchronization with the host.

Wherever possible, Comm|Scope measures the time of asynchronous CUDA operations without introducing additional synchronization. In this context, "asynchronous" means operations that are asynchronous with respect to the host. An example of such an operation is `cudaMemcpyAsync`, which invokes a data transfer, but may return before that transfer is complete. Such operations cannot be timed directly by recording the wall time on the host before and after the call, as the underlying operation may begin and end arbitrarily long after the function is called and returns. Comm|Scope wraps asynchronous operations in CUDA events, and uses the `cudaEventGetElapsedTime` to measure the time of the operation as closely as possible without measuring the cost of synchronization. CUDA events are associated with a stream and device, and events associated with separate devices cannot be directly compared. When asynchronous operations occur on separate devices, a full synchronization with the host is required during the timing measurement.

2.2.2 Performance Variation. Even when background operations are minimized, repeated measurements will show some variation. Comm|Scope automatically provides statistical summaries of repeated runs, and this work reports the standard deviation over 5 repeated runs of each set of benchmark iterations as error bars in the figures (box-whisker chart). No prior microbenchmarking work in this area reports any variability across runs.

[1] This work sometimes uses CUDA terminology of "host" and "device". The CUDA host is the processing and memory components of the system that are not part of the GPU, i.e. the CPU and CPU memory hierarchy. The CUDA devices are the GPUs and GPU memory.



(a) No Flush

(b) Flush

(c) No Flush

(d) Flush

Figure 2: Effect of cache flushing on transfer bandwidth. Figures 2a demonstrate CPU caches introduce complexity and irregular performance into the transfer. This is caused by interaction between higher bandwidth caches and the data transfer. For example, Figure 2c shows a peak at transfer sizes that fit in the CPU cache, as this transfer involves an intra-CPU copy (Section 4.1). Figures 2b and 2d show how how flushing the CPU caches during benchmark iterations strongly influence some results.

2.2.3 Variable CPU Clock Speeds. Many systems feature dynamic CPU frequency adaptation to conserve power and boost performance for transient tasks. This presents a challenge when measuring performance, as CPU frequency may not be the same from run to run. In the context of this work, the CPU performance could have a substantial impact on performance of the CUDA unified memory system and CUDA driver operations. On Linux, Comm|Scope automatically reports when the CPU performance governor is not locked in the high-performance state, allowing the user to act appropriately before measuring performance. Results in this report use the Linux "performance" CPU governor, which locks all CPUs at their highest sustainable frequencies.

```
cpupower frequency-set --governor performance
```

Prior works make no report of whether or how this variable is controlled.

2.2.4 CPU Data Caching. CPU caches have a measurable effect on CPU/GPU data transfer performance. For example, Figure 2 shows the effect of flushing CPU cache lines prior to CPU-to-GPU transfers. Comm|Scope uses `dcbf` [20, p. 773] on POWER and `clflush` [7, p. 139] on AMD64 to control this effect. These instructions invalidate and flush the cache lines associated with a particular virtual address from all CPU data transfers. By looping over all addresses in a host buffer, Comm|Scope can ensure that no CPU holds an part of that buffer in any cache. This is done in some benchmarks before the transfer is initiated. Prior works that measure CPU/GPU transfers do not address this consideration.

3 IMPLEMENTATION

This section describes the implementation decisions made in the microbenchmarks. The full code is available at scope.c3sr.com.

3.1 Benchmark Commonalities

The benchmarks are written to have an initialization phase, followed by one or more iteration phases, followed by a finalization phase.

In the initialization phase, CUDA streams and events are created, memory is allocated, and memory is initialized. If a CPU is involved in the benchmark, libnuma is used to bind the process and allocations to the desired NUMA node. When GPUs are involved, `cudaSetDevice` is used to control the active device. These bindings may be modified several times during the benchmark as desired to ensure operations and allocations occur in the correct places. Host allocations are performed with `aligned_alloc` and `sysconf(_SC_PAGESIZE)` to align them to host page boundaries. Device allocations are produced with `cudaMalloc`. Unified memory allocations are performed with `cudaMallocManaged`. Allocations are initialized with `memset` or `cudaMemset` to ensure they have backing physical pages.

In the iteration phases, the operation to be measured is invoked. The active CUDA device may be modified as needed to determine which devices are executing the workloads. The iteration phases are executed until the cumulative operation time reaches one second. All benchmarks share the pattern of moving data across one or more interconnects from a source component to a destination component.

In the finalization phase, the resources used during the execution of the benchmark are released, so that successive benchmark runs do not leak resources.

3.2 Explicit Point-to-point Transfer Benchmarks

Comm|Scope defines unidirectional and bidirectional host/device and device/device data transfer benchmarks. When supported, GPU-GPU transfer benchmarks are defined with peer access enabled and disabled.

- *initialization*: Host and device allocations are created and initialized. Pinned host allocations are produced with `cudaHostRegister`. A "start" and "stop" event are created for each transfer. A CUDA stream is created for each transfer. Optionally, peer access is enabled between participating GPUs.
- *iterations*: The benchmark optionally flushes CPU caches (Section 2.2.4). For each operation, the start event is recorded, `cudaMemcpyAsync` is triggered, and the stop event is recorded. The benchmark iteration time is set to be the elapsed time between events.

3.3 Zero-copy Transfer Benchmarks

Comm|Scope defines unidirectional host/device and device/device zero-copy benchmarks, as well as bidirectional device-device zero-copy benchmarks. Each data transfer benchmark can either be a "read" or a "write" operation. For a read, the destination device reads data from an allocation on the source. For a write, the source device writes data to the destination. Since the host cannot modify data on the device through the zero-copy paradigm, only unidirectional host/device benchmarks are defined.

- *initialization*: Host and device allocations of the desired size are created and initialized. Host allocations are mapped to the device address space with `cudaHostRegister` and `cudaHostRegisterMapped`. Peer access is enabled between devices for GPU/GPU transfers.
- *iterations*: If a CPU is involved, the caches are flushed. For a "read" operation, the destination device is made active or for a write operation, the source device is made active. A grid of 256 blocks of 256 threads on the GPU iterates over the allocation, using consecutive threads to make a 4-byte read or write to consecutive elements of the allocation. For duplex transfers, one of these workloads is asynchronously executed on each device. For single GPU workloads, CUDA events are used to measure the time. For simultaneous GPU workloads, the host wall-clock time is used.

3.4 Unified Memory

Comm|Scope defines unidirectional host/device and device/device unified memory benchmarks. These benchmarks measure bandwidth during demand and prefetch page migrations. To generate demand page migrations, a "write" workload is defined. On the CPU, this workload writes a single 0-byte to the first byte of each page. This causes the page to migrate to the CPU while doing the minimal amount of work. Comm|Scope uses multiple CPU threads to generate sufficient pressure on the unified memory system. The allocation is evenly partitioned, and each CPU thread does this to its respective partition. On the GPU, a fixed grid of 256 blocks of 256 threads iterates across the allocation. Each warp writes a single 0-byte in a page before striding the size of the grid to the next page. This causes the page to migrate to the GPU while doing the minimal amount of other work.

- *initialization*: A unified memory allocation is created for each transfer. Allocations are initialized with `cudaMemset`. A "start" and "stop" event are created for each transfer. A CUDA stream is created for each transfer.
- *iterations*: `cudaMemPrefetchAsync` is used to ensure the backing pages are on the source device of each transfer. Then, the write workload is executed on the destination device of each transfer. For transfers involving a single GPU executing a workload, CUDA events are used to time the iteration. For transfers involving data migrating to the CPU, the host wall-clock time is used. For bidirectional GPU/GPU transfers, host wall-clock time is used as events on separate devices cannot be meaningfully compared.

The prefetch benchmarks are the same as the demand migration benchmarks, except with no "write" workload. Instead, we use `cudaMemPrefetchAsync` to move pages from the source device to the destination device, which is followed by the measured operation.

4 RESULTS

Comm|Scope is used to evaluate data-transfer performance on three systems that represent modern heterogeneous computing platforms. These systems feature the types of CPUs, GPUs and interconnect technologies that run neural-network, machine learning, and data-driven workloads. Table 2 summarizes the evaluation systems.

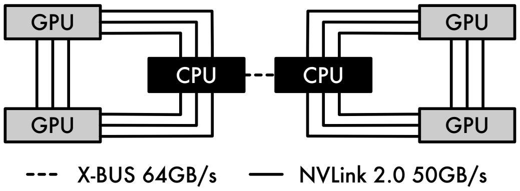


Figure 3: IBM AC922 interconnect schematic annotated with theoretical peak bidirectional transfer rates.

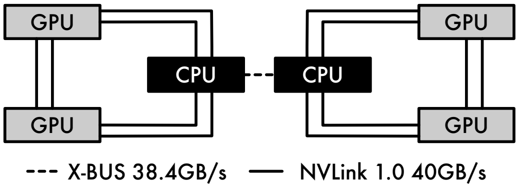


Figure 4: IBM S822LC interconnect schematic annotated with theoretical peak bidirectional transfer rates.

- The IBM AC922 3885-GTG [12], whose topology is shown in Figure 3, features two IBM POWER9 CPUs and four Nvidia V100 [4] GPUs. The system is organized into two triads comprising one CPU and two GPUs. Each triad is fully-connected with a three-lane NVLink 2.0 interconnect, offering up to 150GB/s of bidirectional bandwidth. The CPUs in the two triads are connected with an x-bus that offers 64GB/s of bidirectional bandwidth.
- The IBM S822LC 8335-GTB [13] is similar to the AC922, except the CPUs are IBM POWER8s, the GPUs are Nvidia P100s[3], the intra-triad interconnects are two-lane NVLink 1.0 with 80 GB/s bidirectional bandwidth, and the x-bus has 38.4 GB/s bidirectional bandwidth. Figure 4 shows a schematic of the system topology.
- The SuperMicro 4029GP-TVRT [29], whose topology is shown in Figure 5, is organized into two pentads of one Intel Xeon Gold 6148 CPU and four Nvidia V100 GPUs. The CPUs are connected to the GPUs in their pentad by PCIe 3.0 x16. The GPUs are connected within and across pentads by either one or two lanes of NVLink 2.0. Not all GPUs are directly connected — communication between such GPUs must traverse more than one link and, when communicating to the CPU, through a PCIe Switch (PEX 9765). The CPUs are connected by three lanes of Intel Ultra Path Interconnect (UPI), with each lane offering 20.8 GB/s of bidirectional bandwidth.

Comm|Scope uses the Google Benchmark v1.4.0 library [17] to manage the benchmark runs. Each measured operation is repeated one or mote times, with a constraint that it must be executed for at least one iteration and takes at least one second. Comm|Scope automatically computes the standard deviation over repeated executions. The results plots in this work use error bars at each measurement point to represent the standard deviation of five executions.

Comm|Scope has 27 microbenchmarks to cover various CUDA transfer types. Table 1 summarizes the included configurations. The rest of this section highlights a selection of results on the

Model	CPU	GPU	CPU-CPU	CPU-GPU	GPU-GPU	Kernel	CUDA	Driver	Page Size
AC922 8335-GTH [12]	2x IBM POWER9	4x V100 (32GB)	x-bus (64 GB/s)	NVLink 2.0 x3		4.15.0	9.2	396.26	64K
S822LC 8335-GTB [13]	2x IBM POWER8	4x P100	x-bus (38.4 GB/s)	NVLink 1.0 x2		4.4.0	9.2	396.26	64K
4029GP-TVRT [29]	2x Intel Xeon Gold 6148	8x V100 (16GB)	3x UPI (62.4 GB/s)	PCIe 3.0	NVLink 2.0 x1/x2	4.15.0	9.1	396.26	4K

Table 2: Summary of Evaluation Systems.

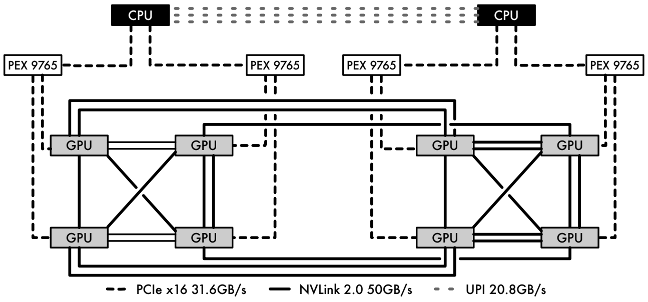


Figure 5: SuperMicro 4029GP-TVRT interconnect schematic annotated with theoretical peak bidirectional transfer rates.

evaluated systems as they relate to high-performance interconnects on modern systems. For more raw results on more systems, please refer to scope.c3sr.com.

4.1 Pageable Host Allocations and Fast Interconnects

Explicit transfers between pageable host allocations (e.g. those produced with `new`, `malloc`, or `aligned_alloc`) and the device through `cudaMemcpyAsync` achieve low bandwidth relative to the advertised link capabilities. Figures 6a, 6b, and 6c show the overall transfer rate on the evaluation systems. These transfers involve two phases. The CUDA system first copies data from the user-visible pageable allocation to a driver-managed pinned buffer that is not visible to the user (Figures 6b, 6e, 6h), which happens at a relatively slow rate allowed by a single CPU thread moving memory. Then, a DMA is invoked for the GPU to access the data from the internal pinned buffer (Figure 6c, 6f, 6i). The existence of this intermediate pinned buffer is necessary because the pages must be guaranteed to be in memory during the DMA from the GPU. The DMA transfers data at a rate closer to the theoretical interconnect bandwidth. For example, Figure 6c shows that local pinned-to-GPU transfers on AC922 achieve nearly the theoretical 80GB/s, while remote transfers get up to 40 GB/s over the 64GB/s x-bus.

On the IBM systems with NVLink between the CPUs and GPUs, the pageable-to-GPU transfer rate does not exceed the pageable-to-pinned transfer rate, even though the pinned-to-GPU transfer rate is always higher. This suggests that the intra-CPU copy limits the performance, which is substantially slower than how fast the data can be moved to the device over the interconnect, even on the relatively low-speed PCIe 3.0 interconnect on the 4029GP.

Bidirectional transfers involving pageable host allocations are further limited. Comm|Scope bidirectional transfers comprise two simultaneous host/device transfers in opposite directions between the same pair of components. Each of those individual pageable-to-GPU transfer has the same performance limitations described above, and additionally, the CUDA system does not allow two simultaneous pageable-to-pinned copies. Though the device-device DMA may occur simultaneously, any required host-host copies are serialized. Figure 7 highlights the consequences of this effect. Pinned/device bidirectional transfers achieve roughly double the bandwidth of unidirectional transfers of the same kind (as the two transfers are fully overlapped). This allows these duplex transfers to nearly saturate the interconnects. Pageable/device bidirectional transfers are only slightly faster than unidirectional transfers. They are not meaningfully overlapped, as the performance-limiting pageable-to-pinned copies are serialized.

4.2 High Bandwidth Systems Exhibit Strong Locality Effects

In device/device transfers and host/device transfers with pinned host allocations, transfer performance is not limited by a slow pageable-to-pinned copy and underlying interconnect performance becomes a dominant effect. On the IBM platforms, all pairs of devices are connected by a relatively high-speed interconnect: two-lane NVLink 1.0 at 80 GB/s, three-lane NVLink 2.0 at 150 GB/s, or X-bus at 38.4 GB/s or 64 GB/s. Transfers between directly-connected devices traverse a fast NVLink, but “remote” transfers always cross at least one of the slower X-bus links. Figures 6c, 6f, and 6i show pinned-to-GPU bandwidth with local and remote components. On AC922, where the NVLink bandwidth is substantially higher than the X-bus bandwidth, there is a large difference in achieved performance between local and remote components. For example, pinned-to-GPU transfers can utilize up to 72 GB/s (or 95% of the theoretical peak 75 GB/s unidirectional three-lane NVLink 2.0) for local transfers, but are limited to 41 GB/s for a remote transfer over the X-bus. On S822LC, the X-bus bandwidth is comparable to the NVLink bandwidth, and on 4029GP, the UPI bandwidth between CPUs is actually higher than the PCIe 3.0 bandwidth between CPUs and GPUs, so neither of those systems show a strong locality effect. Similar locality effects are seen for GPU-GPU transfers (Figure 8).

Unlike CPU-GPU, GPU-GPU locality effects are particularly complicated on the 4029GP. Pairs of GPUs may be connected through one-lane (50 GB/s bidirectional) or two-lane (100GB/s bidirectional) NVLink 2.0. Remote GPU pairs always have at least one of the slower one-lane NVLink 2.0 interconnects between them. Figure 8e shows the four distinct transfer patterns: GPU0-to-GPU1 (“1”), over a single NVLink lane, GPU0-GPU2 (“2”) over a single NVLink 1.0, or two hops of two lane NVLink, GPU0-GPU3/4 over a two-lane NVLink, and GPU0-GPU5/6/7 (“‘5,6,7”) over NVLink x2 followed by NVLink x1.

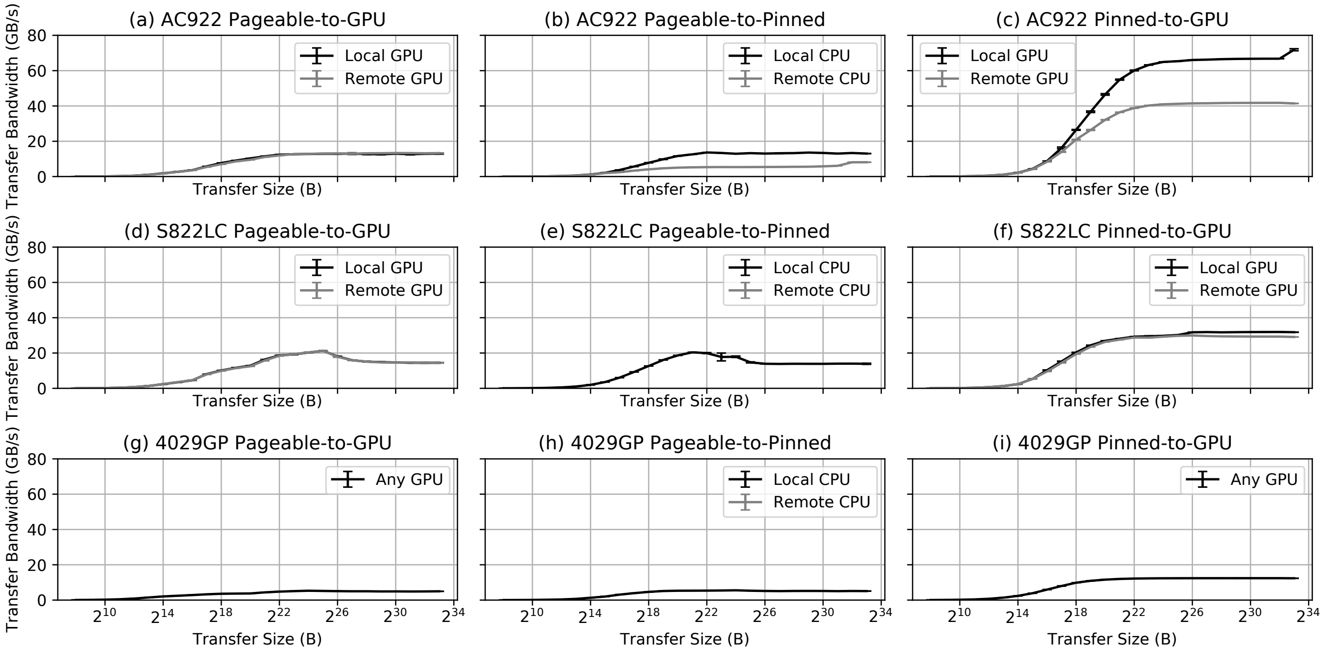


Figure 6: Bandwidth measurements for pageable-to-GPU transfers. (a,d,g) show the achievable transfer bandwidth from pageable host allocation to local (directly-connected) and remote (multiple-hop) GPUs on the three systems. Each of these transfers involves a pageable-allocation to pinned-allocation copy (b,e,h) and a pinned-allocation to GPU transfer (c,f,i) on the same systems.

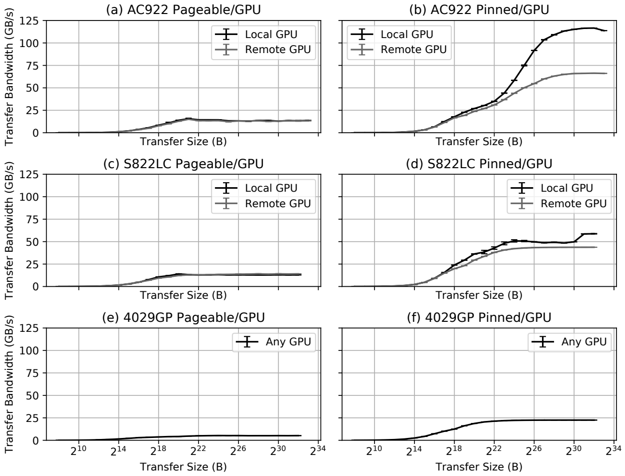


Figure 7: Bidirectional `cudaMemcpyAsync` bandwidth measurements on the evaluation systems. Pageable and pinned refer to the host allocation style for the overlapping transfers.

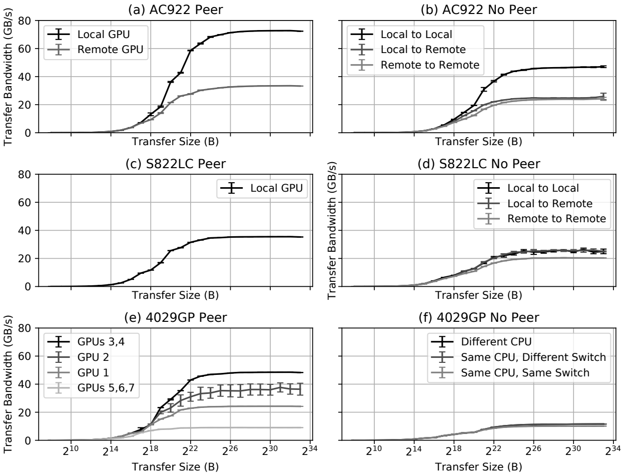


Figure 8: Unidirectional GPU-GPU `cudaMemcpyAsync` bandwidth measurements on the evaluation systems, with and without peer access enabled.

4.3 High Bandwidth Links need Peer Access

Peer access allows a GPU to access memory present on another GPU without involving the CPU. Whether or not the CPU-GPU interconnect is fast, peer access is crucial for making full use of the interconnects. On some platforms, not all pairs of GPUs support peer access. For example, on S822LC, only local GPUs may engage in peer access. On AC922 and 4029GP, any pair of GPUs may engage in peer access. Even if peer access is available, it may be selectively enabled and disabled. When peer access is disabled, Comm|Scope uses NUMA pinning to involve a fixed CPU in the transfer.

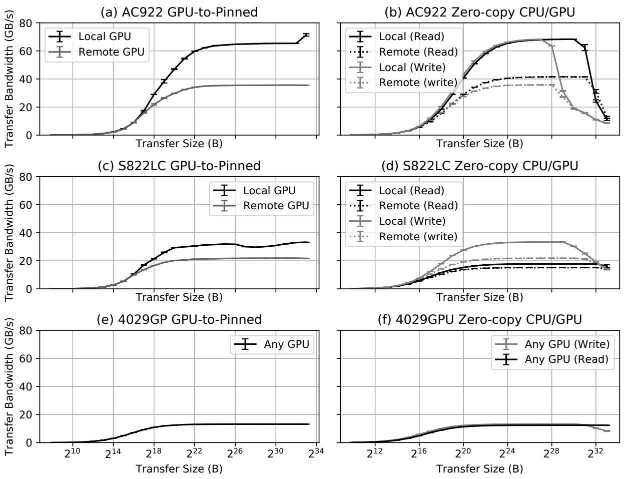


Figure 9: Explicit GPU-to-CPU (a,c,e) and zero-copy (b,d,f) bulk transfer bandwidths. “Read” refers to the GPU reading from host memory, and “write” refers to the GPU writing to host memory (Section 3.3).

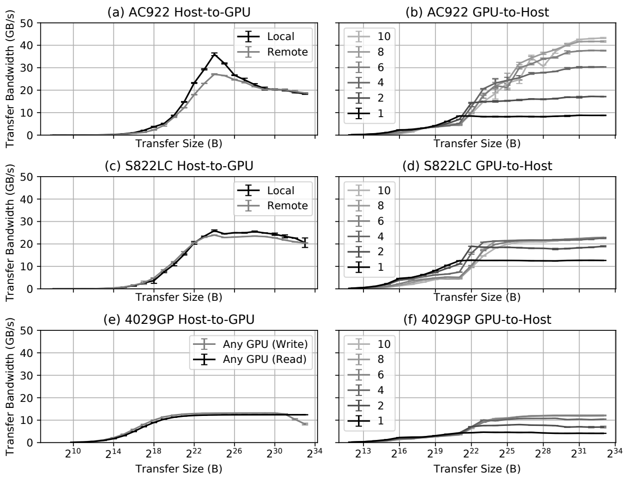


Figure 10: Unified memory demand CPU/GPU transfers. (a), (c), (e) show host-to-device transfers. (b), (d), (f) show device-to-host bandwidth vs. number of requesting host threads for a directly-connected CPU/GPU pair.

Figure 8 shows GPU/GPU transfers with and without peer access enabled. For local GPUs, the availability of peer access is necessary to fully utilize the interconnect. When peer access is enabled, transfer bandwidth increases by around 10 GB/s. S822LC does not support peer access for remote GPUs. On 4029GP, if peer access is not enabled, all transfers must go through the relatively slow PCIe CPU/GPU interconnect, completely limiting the performance available over NVLink.

4.4 Zero-Copy Memory can match Explicit Transfers

Figure 9 shows a comparison of zero-copy and explicit transfers. On AC922, zero-copy bandwidth matches explicit transfers for intermediate sizes (Figs 6c, 9a, and 9b). On S822LC, the same is true for GPU-to-pinned transfers (a zero-copy write), but the zero-copy read is about 10GB/s slower (Figs 6f and 9d). The limited 4029GP CPU/GPU bandwidth again hides most effects.

When zero-copy bandwidth matches that of explicit transfers, it may be preferable as it can eliminate boilerplate data-transfer code. On all systems, the zero-copy bandwidth drops significantly for extremely large transfers, so explicit transfers may be preferable in this regime, Moving data with zero-copy also requires kernel execution, which could take GPU resources away from other uses.

4.5 Unified Memory Demand Page Migration Bandwidth can be higher than Explicit Transfers from Pageable Allocations

CUDA unified memory is presented as a programmer-friendly way to develop CUDA-accelerated applications on current systems. Conventional wisdom is that unified memory is slower than explicit memory management, though a comparison of Figures 10 and 6 show that unified memory coherence bandwidth can exceed explicit management and pageable allocations when enough host or device threads are used. The number of device threads can be expected to always be sufficient, but more than one host thread is needed for the GPU-to-host unified memory bandwidth to achieve its maximum throughput. A single CPU thread is not able to generate enough memory traffic to saturate the unified memory system or underlying interconnects.

Since bandwidth is not the only consideration for application transfer performance (for example, the unified memory system could cause redundant data transfers not present in an explicitly-managed system), this does not suggest that translating an application to unified memory will necessarily be faster, only that the raw capability of the unified memory system does not necessarily imply a slowdown.

4.6 Demand Page Migration Bandwidth does not Exhibit Strong Locality Effects

Unlike the explicit transfers (Section 4.2), Figure 10 shows that coherence bandwidth is not strongly affected by component locality. Figs. 10a, 10c, and 10e show little different between demand page migration accesses from directly-connected or multi-hop devices. This suggests that interconnect bandwidth is not the primary performance limiter (though it clearly has some effect, as 4029GPU exhibits lower bandwidth). The strongest locality effect occurs around 16MB transfers on AC922, which is in the regime of the last-level cache size. Data in the CPU cache can be accessed much more quickly than in DRAM, so unified memory bandwidth may be influenced by CPU memory bandwidth, just as with pageable explicit transfers.

4.7 Unified Memory Prefetch Bandwidth Slow at Intermediate Sizes

Unified memory demands page migration provide substantially lower bandwidth than GPU/GPU or pinned/GPU transfers. Figure 11 shows that unified memory prefetching using `cudaMemPrefetchAsync` can nearly match explicit transfers for large sizes, and also brings the commensurate dependence on locality. Unfortunately, for intermediate transfers of around 4KiB-32MiB, the bandwidth is substantially lower than explicit transfers at the same size. For large data, using unified memory for convenience and dropping back to prefetch hints for bulk transfers remains a viable path to high-performance data transfer.

5 RELATED WORK

Table 1 shows how prior works have overlapped with the microbenchmarks in Comm|Scope. Though some of the specific measurements made by Comm|Scope have been made previously, we believe Comm|Scope represents the most comprehensive coverage of CPU-CPU and CPU-GPU point-to-point communication performance to date.

Li et. al [23, 24] created Tartan, a benchmark suite for evaluating GPU interconnects in the context of machine-learning workloads. Tartan includes microbenchmarks for point-to-point and collective GPU-GPU communication within and across nodes. To that end, Tartan measures bandwidth, latency, and efficiency of GPU-GPU explicit memopry copies and the Nvidia Collective Communications Library (NCCL) on PCIe, NVLink 1.0, NVLink 2.0, and Infiniband systems with GPUDirect RDMA. Unlike Tartan, Comm|Scope includes CPU/GPU transfers, but only measures point-to-point transfer bandwidth within a single node. Tartan also includes 14 larger application benchmarks. Those benchmarks are categorized by what communication pattern they exhibit. Some of those benchmarks are categorized into the CPU-GPU communication pattern, but Tartan does not include corresponding CPU-GPU microbenchmarks.

Tallent et. al [30] evaluate the effect of multi-GPU systems on deep-learning workloads. Along the way, they measure point-to-point explicit GPU-GPU copy bandwidth with and without peer access available, which are two of the microbenchmarks included into Comm|Scope. They also have some GPU/GPU latency and collective communication bandwidth measurements, similar to Tartan.

Ben-nun et al. [9, 10] present Groute and MGBench. Groute is a multi-GPU programming model, and MGBench was developed partially to understand multi-GPU communication patterns before developing Groute. MGBench includes some host/GPU and GPU/GPU zero-copy benchmarks for coalesced and random accesses. MGBench includes device synchronization time in their bandwidth measurements.

The SHOC benchmark suite [15] is meant to measure the performance of hetoerogeneous systems running OpenCL and CUDA workloads. SHOC includes unidirectional bandwidth measurements point-to-point transfers between CPU and GPU.

Landaverde, Zhang, Coskun, and Herbordt [22] investigate the effect of porting several benchmarks to use CUDA's unified memory system on PCIe systems. They present microbenchmarks that correspond to the unidirectional host/device coherence measurements in Comm|Scope. They also have the corresponding implementation using explicit point-to-point copies. They do not account for NUMA topology in their measurements and only consider PCIe systems.

Spafford, Meredith, and Vetter [28] measure NUMA and contention in multi-GPU systems. They overlap with Comm|Scope by measuring point-to-point NUMA-aware CPU/GPU bandwidth on PCIe systems. Though they do not specify, the results suggest that they are measuring bandwidth from pinned host allocations.

Though the CUDA SDK Samples [6] are not intended as a performance measurement tool, it provides demonstration code that measures bandwidth between two GPUs with peer access enabled, bandwidth for duplex pinned/GPU copies, host/device bandwidth for pageable and pinned allocations as well as device-device bandwidth, and bandwidth and latency of unidirectional and bidirectional GPU/GPU transfers with and without peer access available. None of these samples consider NUMA effects, and there are no comparable unified memory performance measurement programs.

Pai [5] develops some benchmarks for unified memory in CUDA 6.0. The benchmarks are developed to understand which situations incur repeated accesses, instead of measuring bandwidth. Mukherjee et al. [27] describes a microbenchmark that measures data transfer performance in the HSA 1.0 unified memory system and the OpenCL 2.0 shared virtual memory. This work includes similar demand migration bandwidth measurements in CUDA.

6 CONCLUSION

Comm|Scope defines benchmarks with thorough coverage of explicit point-to-point intra-node CUDA communication methods, as well as analogous zero-copy and unified memory operations. Though the explicit transfer methods have existed since the earliest versions of CUDA, the enormously increased bandwidth provided by NVLink interconnects throws the significance of system configuration into stark relief. Furthermore, the ability of zero-copy memory and unified memory demand paging as a stand-in for explicit data transfer is poorly understood. As a whole, these benchmarks examine all basic data transfer methods and form a foundation for building more sophisticated communication operations, like collective operations. Without tools to comprehensively measure primitive performance, it is difficult to reason about or optimize more complicated patterns.

Comm|Scope can be used to inform the design of system features that allow applications to best use fast interconnects, including:

- Large host memories should be provided so that as many pinned allocations as possible can be used.
- All pairs of GPUs should support peer access.
- The CPU memory subsystem should be as robust as possible to improve the performance of CUDA unified memory transfers and explicit transfers involving pageable allocations.

These results suggest some basic approaches for applications:

- Pinned allocations should be used when fast transfers are required.
- Unified memory demand migrations should only be relied on with multiple host threads.
- The CPU memory subsystem should be as robust as possible to improve the performance of CUDA unified memory transfers and explicit transfers involving pageable allocations.

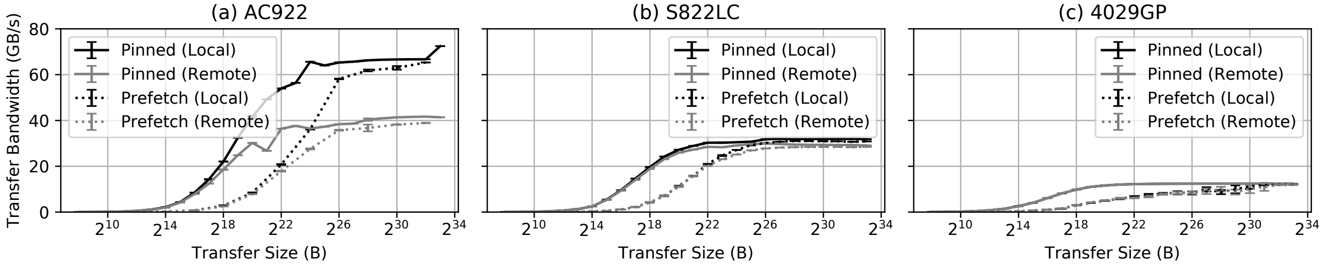


Figure 11: CPU-to-GPU pinned and prefetch transfer Bandwidth.

- Applications need to be careful about GPU placement for best communication performance.

These benchmarks also can uncover irregular behavior, such as drops in zero-copy performance and slower unified memory prefetch bandwidth at intermediate sizes, which could be an indication of hardware or system software misconfiguration.

7 FUTURE WORK

Substantial investigation of using unified memory in the context of various applications has been carried out, both for its programming simplicity [19, 21] and coordination between CPUs and GPUs [11]. This work frequently identifies challenges in understanding the underlying behavior of the unified memory system.

Comm|Scope only includes a small slice of the many dimensions of zero-copy and unified memory performance, which are influenced by parallelism, access pattern, degree of simultaneous access, CUDA driver hints, system software versions, and the degree of integration between CPU and GPU memory systems. In the future, we hope to extend Comm|Scope to measure how these different dimensions jointly affect those communication paradigms.

ACKNOWLEDGMENTS

This work is supported by IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizon Network. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation award OCI-0725070 and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

REFERENCES

[1] [n. d.]. Sierra Supercomputer. https://computation.llnl.gov/computers/sierra. Accessed: 2018-10-11.

[2] [n. d.]. Summit Supercomputer. https://www.olcf.ornl.gov/summit/. Accessed: 2018-10-11.

[3] 2016. *NVIDIA Tesla P100*. Technical Report. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[4] 2017. *NVIDIA Tesla V100 GPU Architecture*. Technical Report. http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[5] 2018. Benchmarking unified memory in CUDA 6.0. https://users.ices.utexas.edu/~sreepai/automem/.

[6] 2018. CUDA 9.2 Toolkit Downloads. https://developer.nvidia.com/cuda-92-download-archive.

[7] Advanced Micro Devices 2018. *AMD64 Architecture Programmer's Manual* (3.26 ed.). Advanced Micro Devices.

[8] amazon [n. d.]. Amazon AWS. https://aws.amazon.com. Accessed: 2018-10-11.

[9] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. 2017. Groute: An asynchronous multi-GPU programming model for irregular computations. In *ACM SIGPLAN Notices*, Vol. 52. ACM, 235–248.

[10] Tal Ben-Nuun. 2017. mgbench. https://github.com/tbennun/mgbench.

[11] Rajesh Bordawekar and Pidad Dasfar D'Souza. 2018. Evaluation of Hybrid Cache-Coherent Concurrent Hash Table on POWER9 System with NVLink 2.

[12] Alexandre B Caldeira. 2018. *IBM Power System AC922 Introduction and Technical Overview* (1 ed.). IBM.

[13] Alexandre B Caldeira, Volker Haug, and Scott Vetter. 2016. *IBM Power System S822LC for High Performance Computing Introduction and Technical Overview* (1 ed.).

[14] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[15] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 63–74.

[16] google [n. d.]. Google Cloud Platform. https://cloud.google.com. Accessed: 2018-10-11.

[17] Google. 2018. Benchmark – A microbenchmark support library. https://github.com/google/benchmark.

[18] Mark Harris. 2013. Unified Memory in CUDA 6. (2013). https://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/

[19] Richard Hayden and Oleg Rasskazov. 2018. Juicing up ye old Monte Carlo GPU code.

[20] IBM 2018. *POWER ISA* (2.07B ed.). IBM.

[21] Jiri Kraus. 2016. High Performance and Productivity with Unified Memory and OpenACC: A LBM Case Study.

[22] Raphael Landaverde, Tiansheng Zhang, Ayse K Coskun, and Martin Herbordt. 2014. An investigation of unified memory access performance in CUDA. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*. IEEE, 1–6.

[23] Ang Li. 2018. Tartan. https://github.com/uuudown/Tartan.

[24] Ang Li, Shuaiwen Leon Song, Jieyang Cheng, Xu Liu, Nathan Tallent, and Kevin Barker. 2017. Tartan: Evaluating Modern GPU Interconnect via a Multi-GPU Benchmark Suite. In *International Symposium on Workload Characterization, IEEE*.

[25] Celso L Mendes, Brett Bode, Gregory H Bauer, Jeremy Enos, Cristina Beldica, and William T Kramer. 2014. Deploying a large petascale system: The blue waters experience. *Procedia Computer Science* 29 (2014), 198–209.

[26] microsoft [n. d.]. Microsoft Azure Cloud Platform. https://azure.microsoft.com. Accessed: 2018-10-11.

[27] Saoni Mukherjee, Yifan Sun, Paul Blinzer, Amir Kavyan Ziabari, and David Kaeli. 2016. A comprehensive performance analysis of HSA and OpenCL 2.0. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 183–193.

[28] Kyle Spafford, Jeremy S Meredith, and Jeffrey S Vetter. 2011. Quantifying NUMA and contention effects in multi-GPU systems. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*. ACM, 11.

[29] SuperMicro 2018. *SuperServer 4029GP-TVRT* (1 ed.). SuperMicro.

[30] Nathan R Tallent, Nitin A Gawande, Charles Siegel, Abhinav Vishnu, and Adolfy Hoisie. 2017. Evaluating On-Node GPU Interconnects for Deep Learning Workloads. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 3–21.

[31] Cliff Wickman, Christoph Lameter, and Lee Schermerhorn. 2015. numactl v2.0.11. https://github.com/numactl/numactl.