Performance Trade-offs in GPU Communication: A Study of Host and Device-initiated Approaches

Taylor Groves*, Ben Brock†*, Yuxin Chen‡, Khaled Z. Ibrahim*, Lenny Oliker*,
Nicholas J. Wright*, Samuel Williams*, Katherine Yelick*†,

*Lawrence Berkeley National Lab {tgroves, kzibrahim, loliker, njwright, swwilliams, kayelick}@lbl.gov
†University of California, Berkeley, brock@cs.berkeley.edu
‡University of California, Davis, yxxchen@ucdavis.edu

***Abstract*—Network communication on GPU-based systems is a significant roadblock for many applications with small but frequent messaging requirements. One common question for application developers is, "How can they reduce the overheads and achieve the best communication performance on GPUs?" This work examines device initiated versus host initiated inter-node GPU communication using NVSHMEM. We derive basic communication model parameters for single message and batched communication before validating our model against distributed GEMM benchmarks. We use our model to estimate performance benefits for applications transitioning from CPUs to GPUS for fixed-size and scaled workloads and provide general guidelines for reducing communication overheads. Our findings show that the host-initiated approach generally outperforms the device-initiated approach for the system evaluated.**

***Index Terms*—Performance evaluation, High Performance Computing, GPU Communication, SHMEM**

I. Introduction

Over the past ten years, GPU-based systems have become dominant among the fastest supercomputers around the world. In the United States, each of the exascale systems currently being built will derive most of its computational power from GPUs. As the computational power in supercomputer systems has concentrated in GPUs, *communication* has become more of a bottleneck, including both communication over the network between GPUs and communication within a node between CPUs and GPUs. In the traditional accelerator model, all communication is handled by the CPU, which introduces additional software and hardware overheads for communication taking place between two GPUs over the network. In terms of raw performance, leaving CPUs in charge of all communication introduces additional transfers over the PCI bus, lowering performance. In addition, users also now have to manage the added software complexity of multiple kernel launches and data transfers.

Recent advances by hardware and software vendors aim to ease this communication bottleneck by allowing GPUs to take a more active role in communication. These include new hardware mechanisms for GPU communication, such as NVLink and GPUDirect RDMA, which allow for direct communication between GPUs within and between nodes, respectively. These also include software mechanisms for CPU-initiated (e.g. CUDA-aware MPI or NVSHMEM) and device-initiated (e.g. NVSHMEM) communication between GPUs. Unfortunately new technologies leave users with complex design choices of understanding varying software complexity and performance costs.

In this paper, we extend the well established LogP[1] performance model to include GPU-initiated communication. This enables application developers to reason about and predict the performance of different GPU-to-GPU communication strategies. We first build a LogP model for CPU-to-CPU communication, then extend this model to GPU-to-GPU communication initiated by either the host CPU or the GPU itself. Our analysis highlights the overheads of GPU-initiated communication, which is currently emulated using assistance from progress threads on the CPU. We evaluate NVSHMEM for the purposes of this work because, (1) it provides both a cpu-initiated and device initiated approach and (2) it is built directly on top of IB verbs, incurring minimal software overheads. We then examine the implications for application speedup from potential future hardware improvements.

Our results in Section IV suggest that host-initiated performance provides moderate performance advantages to the device initiated approach of NVSHMEM 0.3. However, our findings conclude that the more crucial factor in application performance is the application's ability to increase its message size and batch message transfers. This holds true irrespective of whether communication is host or device initiated. In summary, our paper makes the following contributions:

1) Derive LogP models to represent GPU communication.
2) Validate our model using a distributed matrix multiplication benchmark
3) Use our model to predict application speedups for a variety of scenarios
4) Provide recommendations for best practices of GPU communication

We begin with an introduction to GPU communication

mechanisms and communication performance modeling, followed by our experimental methodology for deriving communication model parameters, considering the relationship between performance, message size, and temporal locality of communication. We then apply our analytical model to a set of workloads in order to evaluate our model's accuracy before predicting application speedups along with the potential performance improvements that could be gained by future hardware improvements. Finally, we provide an overview of how our approach extends related work in this field and summarize our insights.

II. Background

A. Remote data access methods for GPUs

In a supercomputing setting, conventional CPUs have typically been in charge of all communication, which introduces both hardware overheads and software complexity. If all communication is to be processed on the CPU, this requires that when a message needs to be sent, the application developer must exit the current kernel launch, where GPU code is being executed, and return to the CPU. In the case where data from a GPU is being sent across the network, the CPU must issue a call to transfer data across the Peripheral Component Interconnect (PCI) bus from the GPU to the CPU before it can be sent across the network. On the receiving side, the CPU must also manually transfer data across the PCI bus in order to move the data into position on the remote GPU, often leading to a bottleneck, particularly when there are multiple GPUs per node.

An alternative to CPU-only communication is GPU-based communication, where data can pass directly from a GPU on one node to a GPU on another. NVIDIA's GPUDirect RDMA is one technology that offers this capability, allowing remote direct memory access (RDMA) transfers between GPUs on different nodes [2]. As with CPU-based RDMA, each GPU has a *shared segment* of pinned memory that is accessible for other GPUs to read and write to over the network using one-sided put and get operations. However, it is important to note that while the current generation of GPUDirect RDMA has a data path that transfers data directly from one GPU to another across the network, its control path still requires involvement from the CPU. The CPU is responsible for preparing an Infiniband request and sending it to the network interface card (NIC), at which point the NIC will copy the data directly from one GPU to another.

While GPUDirect RDMA improves performance by allowing direct transfers between GPUs across the network, it still requires involvement from the CPU. This in turn may require synchronizing threads and thus force the application to exiting and subsequently relaunching a CUDA kernel context after the communication operation has been intitiated. This results in additional performance overheads and software complexity.

NVSHMEM, which is an extension of the OpenSHMEM API, allows both CPU-initiated and GPU-initiated communication between GPUs [3]. GPU-initiated communication is implemented using progress threads on the CPU, each of which has a message queue associated with it. When a GPU thread wishes to issue a communication operation, it will insert the metadata associated with the operation into the CPU queue. When the associated CPU progress thread dequeues the operation, it will prepare an Infiniband request and send it to the NIC. At this point, the NIC will directly copy the data over the network to the remote GPU.

NVSHMEM therefore offers a software API that allows for *CPU-initiated* as well as *GPU-inititated* communication between GPUs in a supercomputer environment. While new hardware technologies may appear that change the performance costs of using these APIs, NVSHMEM represents a low overhead baseline for current GPU architectures.

In light of this, in this paper we develop our experiments on top of NVSHMEM while developing a performance model for GPU communication. From our experiments, we also found NVSHMEM to be significantly faster than MPI implementations on the architecture evaluated.

B. Communication Modeling

In order to reason about GPU performance, we leverage the popular LogP model [1]. Introduced in 1993, the LogP model has been studied and extended for nearly thirty years. At the core of LogP-based models are the original model parameters:

- L an upper bound on the latency, or delay, incurred in communicating a message containing a word (or small number of words) from its source module to its target module. This parameter is significant for small messages.
- o the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform other operations. This parameter is significant for small messages.
- g the gap, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor. The reciprocal of g corresponds to the available per-processor communication bandwidth. This parameter is significant for small messages.
- P the number of processor/memory modules. We assume unit time for local operations and call it a cycle.

The LogP model has been extended in numerous publications [4–9]. These extensions generally add additional parameters to account for differences across message size, synchronization, contention, and divergent control paths. One example is that big G in the LogGP model[4] is added to represent the cost per byte of data transferred rather than using the fixed cost per message g from the original model. This tends to better approximate modern network systems, which can achieve higher bandwidths with larger message sizes. As with all models, there is a trade-off between the simplicity of the model and its accuracy to represent real systems. In this work, we build a LogGOP model, which means that in addition to the o and g parameters for small message sizes, we calculate

large message overhead O and gap G as a function of message size m (bytes). This adds two parameters:

- G gap, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor with respect to a message of size m bytes. This parameter is significant for large messages.
- O overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; with respect to an m byte message. This parameter is significant for large messages.

Another addition from Ino, Fujimoto and Hagihara [6] added a parameter S, for synchronization within MPI rendezvous protocols. We adapt S to account for GPU synchronization and consistency operations.

- S time spent in synchronization and consistency operations for necessary transfers between GPU cache to HBM2 and host DRAM. This parameters is a fixed cost irrespective of message size.

Lastly, the original description of overhead states, "the processor cannot perform other operations when engaged in the transmission or reception of each message". Because of the complexities of parallel execution and the fact that on modern system overhead is split across multiple processors (namely the CPU, GPU and the NIC) we relax this assumption.

Using these model parameters, we derive the parameters using experimental results. As with parameter selection, there are a number of papers that explore the trade-offs of different measurement techniques. A good summary is provided by work by Hoefler, Lichei and Rehm [10]. Generally, engineers derive two of the three parameters experimentally and any remaining time is associated with the remaining parameter. When measuring the parameters there are many obstacles, such as:

1) Absent accurate fine-grained synchronization across clocks, latency must be derived from round-trip times.
2) Overhead for receiving (O_r) is not always equal to the overhead of sending (O_s)
3) Sophisticated NICs may act as another processor allowing for overlap with CPU overheads.
4) Protocol changes at various message sizes (e.g. MPI eager/rendezvous) change the overhead cost.
5) Gap and Overhead may overlap.
6) Pipelining may increase round-trip time while increasing throughput.
7) Caching may decrease the observed latency.
8) Scheduling of shared resources on CPUs and GPUs can be a source of variability.

This partial list illustrates where sources of errors may arise as we assess values for our model parameters. We will revisit these issues in our evaluation.

III. Methodology

A. Benchmarks

We use two sets of benchmarks to derive our LogP model parameters: OFED and NVSHMEM Perftests.

1) OFED Perftest: For the cases where it is desirable to reduce software and CPU overheads as much as possible, we rely on the OFED performance tests [11]. These are written directly using the low level Infiniband Verbs (IBV) API to minimize software overhead. In the `read_lat` tests, work requests are posted one at a time with a "delta" value measuring the time between consecutive post events. For any message k_i that is posted, the message k_{i+1} is not posted until the completion queue signifies success with `IBV_WC_SUCCESS`, reporting half round trip time (RTT/2). All OFED Perftests utilize CPU-initiated communication, but it includes CPU-to-CPU as well as GPU-to-GPU benchmarks, which allows us to compare latency and bandwidth as the data path changes from the CPU's DRAM memory to the GPU's HBM2 memory.

2) NVSHMEM Perftest: As discussed in Section II, the NVSHMEM [3, 12] communication library provides lightweight communication operations for accessing GPU memory. NVSHMEM is written using the IBV interface. We have chosen NVSHMEM for our experiments for two reasons, the first is that it provides a lightweight interface on top of the IB network provider interface. From our experiments, we found this to be significantly faster than MPI implementations on the architecture evaluated (the system default, Spectrum MPI took over 4us in 8B get latency tests compared to 2.4us for NVSHMEM). The more important reason is that NVSHMEM provides two separate control paths for communication. One is invoked directly by the CPU (e.g. `*_get_<all_variants>_on _stream`) and the other is invoked directly within a GPU kernel (e.g. `*_get_nbi_block`). As discussed in Section II, the GPU-initiated interface currently relies on progress threads on the CPU to help with sending the necessary Infiniband requests to the NIC. We compare the get and put-based NVSHMEM perftests included with NVSHMEM 0.3.0. A similar device initiated control path is not available in any MPI implementation to date. There is an option in the benchmark to discard the first k iterations (this is common in many communication benchmarks to avoid reflecting cache performance). We found that if the warm-up phase is disabled, get latency could be as high as 8us. For our parameter assessment we enable the default warm-up of 20 iterations.

We note that NVSHMEM supports *intra-node* messages between GPUs sent over PCIe or NVLink [13]. However this work is focuses on device vs. host-initiated inter-node communication and Intra-node performance is not the focus.

B. Model Parameter Assessment Methodology

In order to build our LogP-based model for GPU communication, we need to isolate and derive each individual model parameter. First, we evaluate the cost of latency L. To do this, we evaluate a set of latency benchmarks as we change the

location of data. By increasing the physical distance from the initiator and target we can approximate the latency of the wire.

Regarding gap, small messages are bound by the transactions per second the NIC can handle. For the hardware evaluated here (Mellanox ConnectX-5) the vendor specification is 200 million messages per second (Mpps) [14]. For an 8-byte message this corresponds to a gap of 5ns, and we use this as our cost for g. For big G we use the formula for gap in the original LogP model (the reciprocal of per processor bandwidth) with the numerator being the message size to derive a lower bound on the gap per byte (m). We refer to this as either $G(m)$ or simply G. This is a lower bound because we know the throughput cannot exceed the theoretical bandwidth of the NIC. Therefore if the interval between consecutive message transmissions is greater than $g + G$ our model will presume the increased time is in the result of overhead $o + O$.

Serial processing performance has not changed substantially compared to increases in bandwidth since the publication of LogP, therefore overhead (o and O) have remained relatively high while g and G have decreased significantly. (Network bandwidth has continued to double every two to four years since 2001.) We can evaluate the cost of some software overheads by comparing benchmark results with an increasing amount of library overheads. Specifically we compare the performance of IB-verbs perftests against the additional software overheads of NVSHMEM. This provides some approximation of overhead cost, but the reality is there are additional overheads we are unable to measure such as overhead in the NIC. With this in mind, we calculate overhead of an 8 byte message (o) as the observed performance of a benchmark minus our calculated values of L g and G. We can perform a similar exercise, varied across message size, to calculate the overhead per byte ($O(m)$ or just O).

C. System Evaluated

We utilize Oak Ridge Leadership Computing Facility's Summit supercomputer for our evaluation [16], summarized in Figure 1. Each node of Summit has two IBM Power9 processors as well as 6 NVIDIA Tesla V100 GPUs. Each node contains 512GB of DDR4 DRAM memory (CPU memory) and 16GB of HBM2 memory per GPU. Within a socket, NVIDIA's NVLink provides CPU-GPU and GPU-GPU communication. NVLink provides two links between every processor, each with a 25GB/s peak bandwidth in each direction. A 64GBps X-Bus (SMP) provides bandwidth between sockets. Each node contains two Mellanox IB EDR NICs each providing 100Gbps (12.5GB/s) of inter-node bandwidth. Each NIC is connected to a Power9 CPU by 16GB/s PCI-e Gen 4. A three-level non-blocking Fat Tree topology is used to connect each node in the system together.

IV. GPU Communication Model

A. Measuring Latency

In our first set of experiments we perform a set of 8-byte OFED `ib_read_lat` tests varying the distance of initiator and target to use the loopback device, a path to the adjacent socket, and a path across a single IB switch. In each case, the target memory being read is located in DRAM. For a fourth test we place the data being read on the target node's GPU memory. We perform 1000 iterations and report the minimum, median, and maximum values in Fig 2. By comparing the results of each tests we can derive a value for L. The data shows a 280ns increase when traveling across a single IB switch and an additional 250ns cost for accessing data in

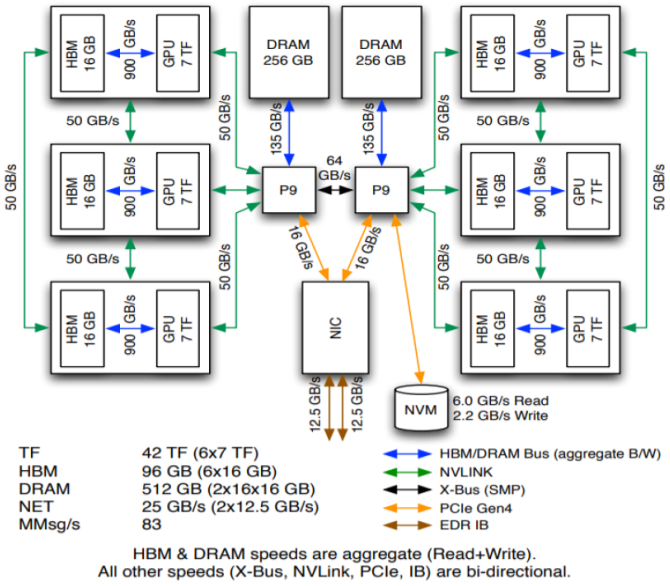

Fig. 1. Summit node layout, original image from work by Oral et. al [15]

	L	o	$O(m)$	g	$G(m)$	S
NV HI Put	530	229	$0.074 * m$	5	$m/12.5$	NA
NV HI Get.	530	247	$0.067 * m$	5	$m/12.5$	NA
NV DI Put	530	219	NA	5	$m/12.5$	4,380
NV DI Get	530	237	NA	5	$m/12.5$	4,570

TABLE I
MODEL PARAMETERS FOR HOST AND DEVICE INITIATED NVSHMEM ON SUMMIT (SINGLE MESSAGES LESS THAN 8KIB IN SIZE, UNIT NS.).

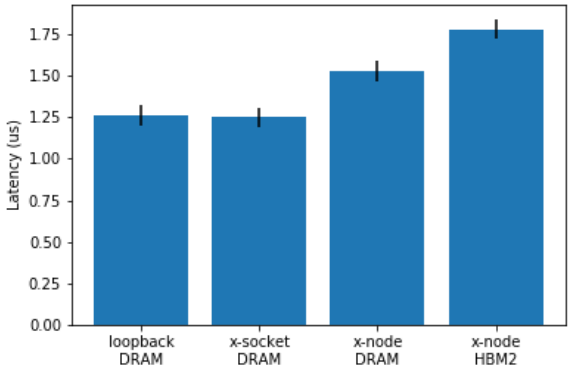

Fig. 2. Deriving l values: 8-byte OFED `ib_read_lat` tests by varying the distance of the initiator and target: (1) loopback device, (2) inter-socket, (3) inter-node to target DRAM, (4) inter-node to target HBM2. Minimum, Median and Maximum values shown.

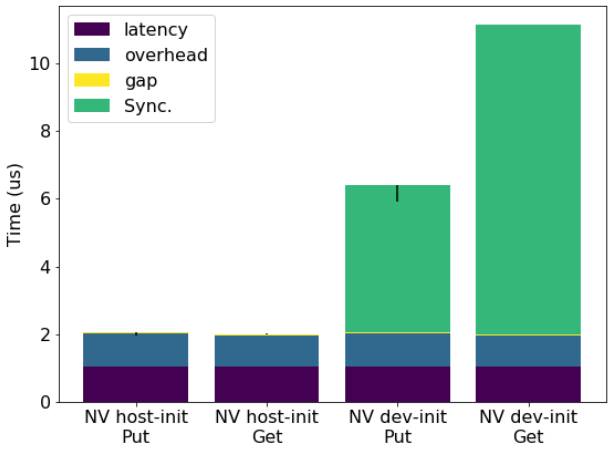

Fig. 3. Modeled and observed NVSHMEM performance for 8B host-initiated and device-initated latency benchmarks. Model cost is further divided into l,o,g,S parameters. Error bars show difference in total benchmark compared to total modeled time. All results are inter-node and use GPU HBM2 memory.

GPU memory. This does not allow us to determine the PCIe latency, since it is included in each measurement. However, this error term is comparatively minor when compared to overall communication times and does not significantly impact the prediction of the model in Section V.

B. Calculating Gap

Our values for gap are derived from the theoretical limits of the hardware as specified by Mellanox for ConnectX-5 EDR NICs. For small messages we are limited by the maximum message rate of 200 million messages per second (Mpps) which is equivalent to a gap of 5ns. This gives us the value g. As we increase message size we are eventually limited by the bandwidth of the NIC (12.5 GBps). This gives us our formula for $G = m/12.5$ns. We validate these values in Section IV-E.

C. Calculating Overhead and Synchronization

Having established values for L, g and G, we calculate overhead as the remaining value from our experiments. Other approaches have measured o by inserting delays between messages, which are increased until the delay and overhead match the cost of the gap [17]. However, on modern systems gap is significantly smaller and overhead may be spread across multiple processor units, making the approach problematic. With this in mind, we first perform a set of experiments that derive the o, O, and S of the NVSHMEM API and compare it with the performance. First we establish models for NVSHMEM (NV) device (dev.) and host initiated latency benchmarks and then run the benchmarks using small 8-byte messages to derive o and S:

$$NVdev.get = 2(l + G + S) + 4(o + g) \quad (1)$$

$$NVdev.put = S + 2(l + G) + 4(o + g) \quad (2)$$

$$NVhost.get = 2(l + O + G) + 4(o + g) \quad (3)$$

$$NVhost.put = O + 2(l + G) + 4(o + g) \quad (4)$$

These equations represent the nuanced differences between device initiated and host initiated approaches as well as *put* versus *get* operations.

Both *puts* and *gets* transfer the payload once, however, *get* includes the full roundtrip time to retrieve the data from remote memory, whereas *put* operations may return acknowledgement once the target system has guaranteed delivery of the payload. Depending on the system this may occur on the NIC, allowing for a reduction in CPU overheads O. In the equations above, only device initiated approaches incur the cost S (twice for *get* and once for *put*).

We first evaluate performance for 8B messages. By using small messages our equations can be simplified by removing O, G. By substituting our values for l, g, we can then solve for o (219 and 237ns. in Eq. 3 and Eq. 4, respectively). For simplification, we assume a symmetric model for o and S (i.e. it is the same on both initiator and target side). The data comparing NVSHMEM host-initated *get* vs. *put* (Fig. 3, suggests a nearly identical cost for o since *put* is within 70ns. (3.5%) of *get* performance. Last we substitute our values for o and determine S equals 4.6us. in Eq. 1.

a) Host vs Device Initiated NVSHMEM: In Fig 3 we compare the performance of CPU-initiated NVSHMEM versus GPU-initiated NVSHMEM. For 8-byte messages we see that the cost of S incurred by the GPU initiated path is substantial, 72% to 82% of total time.

As discussed in Section II, in the NVSHMEM 0.3.0 implementation, the GPU-initiated communication operations over Infiniband are implemented with the help of progress threads running on the CPU. When a GPU thread issues a communication operation, it must also first issue a global memory fence to flush all modified state from the L2 cache to GPU global HBM2 memory. Only after it finishes this global memory fence can it perform an enqueue operation (consisting of one atomic fetch-and-add and four global GPU memory writes) to the queue associated with its CPU proxy thread. Once this write has been completed, the CPU progress thread can dequeue the metadata written by the GPU thread and prepare and send the corresponding Infiniband request to the NIC.

In order to ensure completion of communication operations, we can use the `nvshmem_quiet()` operation. In this function, the calling GPU threads poll a completion counter which is updated by CPU progress thread once the Completion Queue Entry (an IBV data structure) is generated. Finally, a system level memory fence must be issued to ensure all updated data is visible to all threads across the GPU as well as any clients interacting with the GPU using PCIe or NVLink. If multiple non-blocking GPU-initiated NVSHMEM communication calls are issued simultaneously, then NVSHMEM is able to use only one memory fence to ensure consistency for all the communication requests.

However, with CPU-initiated communication, a CPU thread may directly prepare and send a command to the NIC, which avoids the overhead associated with both the progress thread's queue and the memory fence. However, CPU-initiated com-

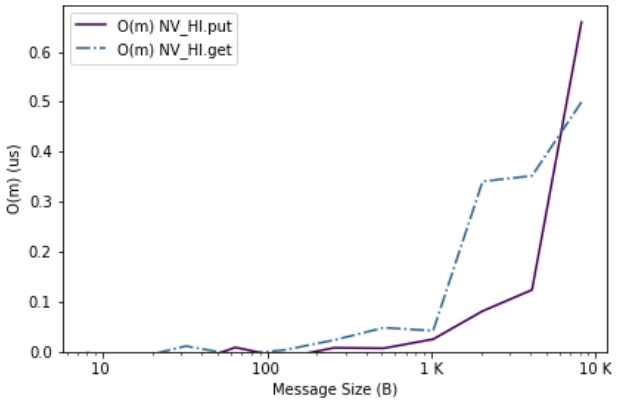

Fig. 4. Overhead with respect to message size after subtracting latency and gap values for host initiated NVSHMEM *put* and *get*. X-axis is log-scale. Least squares regression is performed to derive values for O in Tab. I

munication may incur additional hardware overheads due to the need to issue multiple kernel calls.

D. Single-message (under 8KiB) overheads

To evaluate the cost of single message, it is necessary to examine how O scales with respect to message size (m). To calculate this, we run the NVSHMEM host and device benchmarks with increasing message size. We then subtract the previously derived values for L, o, g and G before performing a least squared linear regression.

a) Overlap of parameters: As message size increases messages begin to be chopped into smaller units for transmission. This enables the overlap between O and G. For Summit we begin to see this reflected in the model for m greater than 8KiB such that the observed impact of O decreases as G becomes larger. For this reason, we evaluate single-message overheads for messages up to 8KiB in Fig 4. Similarly, for the device initiated approaches there is overlap between O and S, such that we assign the value of NA to O for device initiated rows in Tab I.

Least squares linear regression gives a value of O equals $0.74 * m$ ns and $0.67 * m$ ns for NV HI *put* and *get*, with a R^2 values of 0.92 and 0.86, respectively. This implies a modest portion of the variation in performance is unexplained by O (8-14%). This does not drastically impact model accuracy, since the importance of O, is diminished for both small and large messages, where o or G make up the bulk of cost. This data is displayed in Fig. 4. We use this data to populate the values of o and $O(m)$ in Table I.

E. Pipelined message overheads

Most communication libraries are able to take advantage of multiple messages sent back-to-back through optimizations such as combining/piggybacking requests onto one another. To evaluate this we utilize bandwidth benchmarks to measure throughput for varying message sizes. As we enqueue a large number of messages we are able to hide the repeated round-trip latency costs. We can then calculate time spent per byte

	L	o	$O(m)$	g	$G(m)$	S
IBV DRAM/Host	NA	NA	$290 * m^{-1.15}$	5	$m/12.5$	NA
NV Host Init.	NA	NA	$624 * m^{-1.09}$	5	$m/12.5$	NA
NV Dev. Init.	NA	NA	$1411 * m^{-0.781}$	5	$m/12.5$	NA

TABLE II
MODEL PARAMETERS AS MEASURED FOR NVSHMEM AND VERBS ON SUMMIT WITH MESSAGE PIPELINING (UNIT NS.).

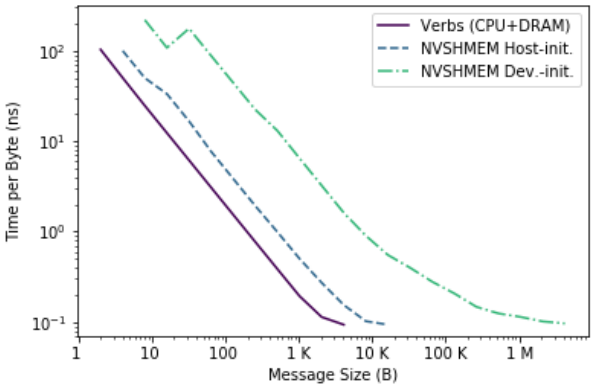

Fig. 5. Amortized time per byte sent for varying message size (derived from pipelined get bandwidth). Showing how overhead per byte (O) is reduced as we enqueue a large number of messages for (1) verbs accessing target DRAM (2) NVSHMEM Device Initiated and (3) NVSHMEM Host Initiated.

transmitted. Our experiments use the NVSHMEM perftest bandwidth benchmarks (`bw`) and `shmem_get_bw` for host and device initiated, respectively. In each benchmark two separate nodes send 10,000 messages back to back for each message size. This is plotted in Fig 5. In the plot, we see a protocol transition at 32 byte message sizes for NVSHMEM host-initiated. This is the result of small messages payloads being able to piggyback on control flow between the device and host. As message size increases, the data shows overhead per byte is reduced across all approaches until we hit the limitations of gap (G). By its definition in SectionII, gap is the minimum time required between messages and cannot be reduced through pipelining. The results show that while overhead can be substantially decreased, both host driven approaches see the benefits at much smaller message size. Specifically, the verbs based approach minimizes overhead at message sizes as small as 4KiB and host initiated NVSHMEM minimizes overhead at approximately 16KiB.

After subtracting for the costs of g and G we perform least-squares linear regression of the functions in the form of a power law ($Y = Ax^B$). The derived values are in Tab. II.

V. MODEL VALIDATION

We focus on validating our model using performance benchmarks resembling real application workloads. Our validation focuses on the NVSHMEM CPU-initiated GPU communication, since this outperformed device initiated in earlier experiments and it is the primary focus on application performance projections in Section VI.

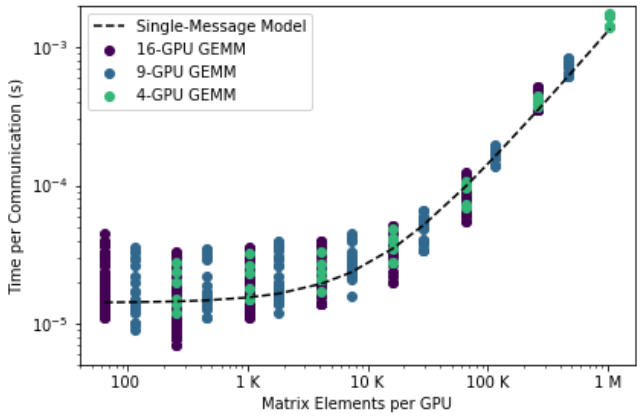

Fig. 6. Single-message model validation for varied matrix size and GPU count. Showing measured communication times across multiple iterations for varying matrix sizes. Though the difference between best case and worst case performance is on the order of tens of microseconds, the model does a good job predicting mean performance. Comparing single-message model with communication timings recorded in 4-GPU (R^2=0.94), 9-GPU (R^2=0.94) and 16-GPU (R^2=0.96) GEMM workloads (non-overlapping communication with computation). Results follow the trends of the single message model.

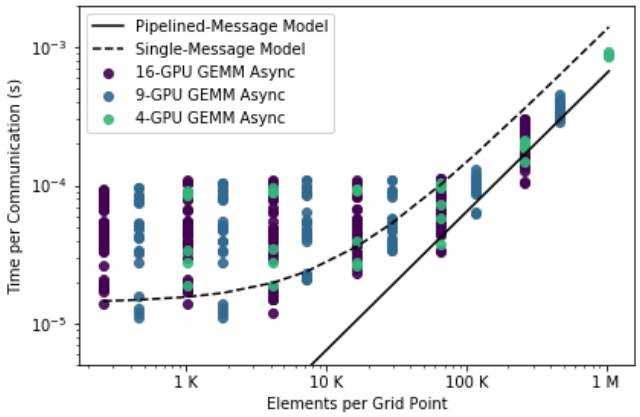

Fig. 7. Model validation using asynchronous GEMM benchmark for varied matrix size and GPU count. Showing measured communication times across multiple iterations for varying matrix sizes. Elements per Grid Point is defined as the total matrix size divided by the number of GPUs. Comparing both single-message and pipelined-message model with communication/computation timings recorded in 4-GPU, 9-GPU and 16-GPU GEMM workloads. Best-case results follow the trends of the single message model for messages smaller than 65KiB. As the message size increases, the timings more closely follow the pipelined model.

1) Distributed matrix multiply: To validate our model we utilize a distributed Generalized Matrix Multiply (GEMM) routine, where a matrix of n elements is divided into a $P \times P$ tile grid (1 tile per GPU) we record the time it takes to perform two of the $\sqrt{P}$ successive get operations, each containing n/P elements. We then compare our model against this timing for validation. For very small element counts, latency-bound communication stresses the small message part of our model, while large element counts issue large bandwidth-bound transfers stress the large message part of our model. The workloads consist of various matrix sizes (32×32 to 2048×2048) distributed across 4, 9 or 16 GPUs (1 GPU per node to ensure inter-node communication) and displays the communication times recorded across different iterations. For experiments utilizing the smallest matrix size the problem is entirely communication bound. In contrast a matrix size of 2048 by 2048 split across 4 GPUs achieve 12.2% of theoretical FLOPS. Therefore we would expect most applications tuned for GPUs to utilize larger matrix sizes or increase utilization through multiple streams. At which point they will have saturated bandwidth. However, the purpose of this section is validation of our model against a more complex communication and computation pattern.

a) Overlapping overhead and gap: As we use our derived model parameters to estimate communication times, we use the common assumption that overhead and gap are able to overlap for pipelined messages. With this in mind our model calculations incorporate the cost of whichever is larger. This means that for small messages O dominates and as message size increases this transitions is bound by G.

b) Simple GEMM validation: We begin with Fig 6, a simple GEMM workload where we utilize synchronous communication to avoid communication-computation overlap. This is to highlight the communication model and not artifacts of computation. We find that our single-message model is a good match for the non-overlapping GEMM communications achieving a R^2 0.95 to 0.97.

c) Asynchronous GEMM validation: Most GEMM implementations will leverage asynchronous communications and computation to provide overlap and reduce the impact of communication. In Fig 7, we evaluate the accuracy of our models against this more complex scenario. By measuring overlapping communication and computation, we see an increase in the variability of recorded timings (over an order of magnitude for small messages). This is expected as we are adding another source of variability in the compute kernel as well as competing for scheduling resources. However, examining the best case measured performance, we see that observed performance follows the single-message model closely until 65K elements. At this point we see a shift towards the pipelined messaging model. In a few cases the pipelined model overestimates the time required to perform the communication. The difference between the observed data and the pipelined model is likely representative of the error accumulated by underestimating G in Section IV. These results show the importance of utilizing two models, one for single messages and another for pipelined messaging as message sizes grow and applications transition in their behavior.

d) Variability: From the results of our validation, there is a range of variability observed in production that is outside the capabilities of a simple LogP model to explain, but this would be interesting to explore in future work. Comparing the best performance to the worst performance sees a roughly fixed difference on the order of tens of microseconds. Reasons for delays include caching effects and system scheduling as

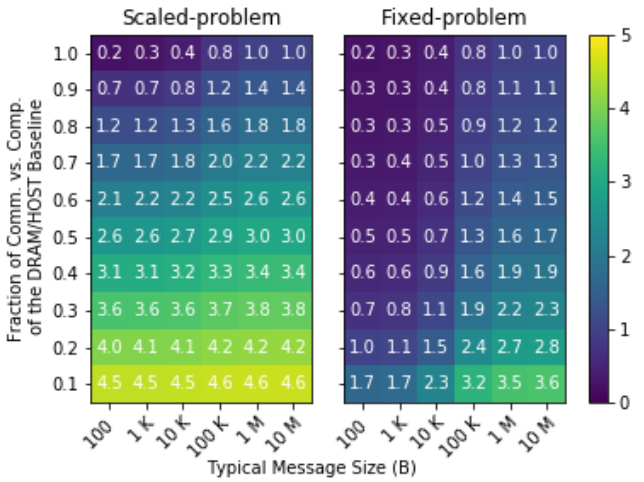

Fig. 8. Impact of Scaling Problem Size on Application Performance when Porting to GPU (for the Single Message Model). Numbers in grid are factor of speedup or slowdown. Being able to increase the amount of computation greatly improves the benefit of the GPU. In the Fixed-problem scenario (right) average performance improvement on the GPU is only 1.2×, whereas the Scaled-problem (left) sees an average speedup of 2.6× (averaged across all message sizes and fraction of communication/computation).

mentioned in II. Our applications used for validation contain a moderate ability to pipeline messages, which contributes to the best case performance observed. The impact of this phenomenon is more significant as computation and communication overlap in Fig 7. Despite the variation of performance the LogP-based model does a good job of predicting typical performance.

VI. Implications for Application Performance

Using the model parameters from Table I, we provide an estimate of performance speedup for applications moving from the CPU to the GPU. We consider three additional criteria for this assessment, (1) the ratio of CPU/GPU computational power, (2) message size, (3) and ratio of communication/computation, such that communication plus computation sums to one. For our CPU to GPU performance projections, we assume the performance scales with the peak flops ratios - 2 TFLOPS for the CPU and 10 TFLOPS for the GPU. Unless otherwise specified, we assume a 5us kernel launch and synchronization time for CPU-initiated communication.

a) Impact of scaling the workload: We consider two scaling scenarios. In the first scenario, workload size remains fixed when ported from CPU to GPU. Because the time to perform the computation decreases and the communication operations take longer compared to the DRAM/Host implementation (Table I, this results in less performance benefit. We refer to this scenario as *Fixed-problem*. The second scenario assumes that the problem size scales along with the increased computing power of the GPU. This scenario generally shows greater speedup benefits for GPUs, and we refer to it as *Scaled-problem*. In both cases the baseline performance is a traditional CPU workload that does not utilize the GPU at all.

In Fig 8 we show the difference that problem scaling makes when porting to the GPU. The figure shows projected speedup for both fixed and scaled problem size across varying message sizes and ratios of communication/computation. Any value less than one projects a slowdown on the GPU when compared to the CPU approach. This data was generated using the single-message model. In summary, applications that are not able to increase the volume of computation on the GPU will be limited by communication bottlenecks following Amdhal's Law. These workloads will be forced to increase message size, enable message batching, or find ways of overlapping communication and computation to be productive on GPUs.

While we show the impact of problem scaling in Fig. 8, for the remainder of this paper we show projected applications speedup assuming a Fixed-problem. Furthermore, to simplify the presentation of Figs. 9-11, we only show data for the scenario where the original CPU-based application spends 25% of time in communication and the remaining 75% of time in computation.

b) Computation and Communication Overlap: Many applications are able to overlap some amount of communication and computation, thereby amortizing communication costs. Both CPU-initiated and GPU-initiated techniques are capable of communication-computation overlap. For GPU-initiated communication the same kernel can continue to perform computation while the CPU progress thread forwards the Infiniband request to the NIC. For CPU-initiated approaches overlap is provided via multiple independent streams (e.g. `cudaMemcpyAsync` in CUDA). If an application overlaps communication/computation this changes the fraction in Fig. 8 accordingly. For example an application that takes 10 units of time to complete without any overlap and spends 60% of time in communication, is rewritten so that 50% of communication overlaps. The application now takes 7 units of time to complete and communication is 43% of that runtime. One caveat, illustrated by Fig. 7, is that variability tends to increase with overlapping computation operations.

c) Increasing message size and pipelining: Fig. 9 estimates the application benefit of increasing message size for the single-message and pipelined models for a fixed problem size and host-initiated communication. As shown in previous sections, the efficiency of communication increases significantly with message size. This has a noticeable impact on single-message applications that are able to send messages greater than 10KB. In cases where applications are unable to increase the message size, but are able to batch messages, the pipelined model shows a greater range of effectiveness. For the scenario shown, the pipelined model achieves the maximum possible speedup of 2.5× using just 100B messages, whereas the single model approach must send messages larger than 1MB to approach the same benefit.

d) Reducing kernel synchronization times: For future GPU based systems one of the goals of vendors is to reduce kernel synchronization and launch times. To evaluate the impact this has on application performance, we evaluate our Single message model using an assumed 5us and 2us

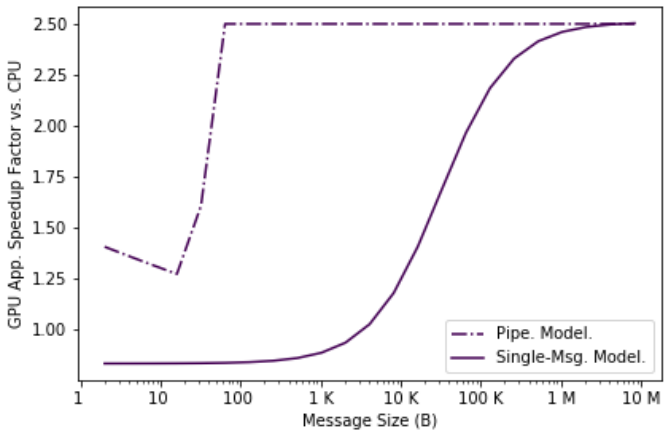

Fig. 9. Pipelining Communication Application Benefit. Speedup porting fixed-problem to GPU (25% comm., 75% comp. on CPU). Two additional techniques for increasing performance for applications porting to GPUs are (1) increasing message size and (2) batching messages to take advantage of pipelining. If the application sends a small number of large messages (>100KB), GPUs provide good value. If message sizes are smaller, pipelined messaging increases efficiency and provides a much larger range of effective message sizes.

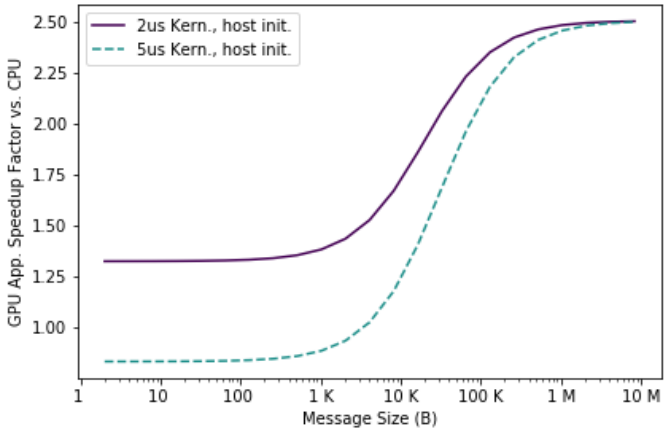

Fig. 10. Benefit of Reduced Kernel Synch. Times (5 us. vs. 2 us.) for Application Ported to GPU (25% comm. on CPU with Single-message Model). Reducing kernel overheads from 5us to 2us is projected to have the biggest impact for small message (1.49×) and tapers off for large messages.

kernel launch time. Fig. 10 provides an estimate of the relative performance improvement. Again we assume a 25% communication to 75% computation ratio for the original CPU implementation with fixed problem size. The benefit of reducing kernel synchronization times has the biggest impact on small message communications, going from a slowdown of 0.8× to a speedup near 1.3×. However, as message size increases, the significance of reducing kernel synchronization times diminishes.

e) CPU vs. GPU-initiated communication: In Fig. 11 we compare the impact of CPU versus GPU-initiated communication in NVSHMEM. We again assume a 25% communication to 75% computation ratio for the original CPU implementation

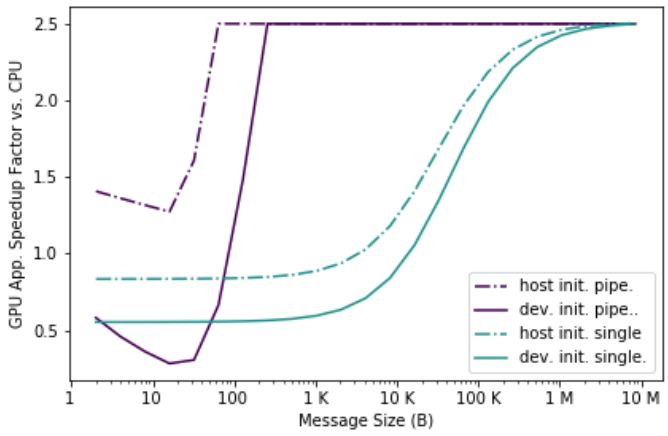

Fig. 11. Host vs. Device-initiated Projected Speedup for Application Ported to GPU (25% comm. on CPU). Host initiated approaches maintains a performance benefit (1.26×) until messages near 512KB in size. For very large messages device initiated improves application performance by 1.09×.

with a fixed problem size. For the pipelined model we see a brief window (1-100B) where there are significant benefits to the host initiated approach before both approaches saturate available bandwidth (limited by g and G). The trade-offs are more nuanced for the single-message model. For messages less than 512KB, applications should use the host-initiated approach. For very large message sizes the device initiated approach provides some minor application speedups.

*f) **Priorities for performance**:* The results of our modeling work suggests that when optimizing an application for GPUs, developers should prioritize the following:

1) Scaling up the problem size on the GPU relative to CPU. However, due to limited GPU memory sizes relative to DRAM this potentially requires greater data reuse.
2) Sending larger messages to increase efficiency
3) Batching/pipelining messages when possible
4) Reducing kernel launch and synchronization times
5) CPU-initiated messaging currently provides the best performance, but under some circumstances device initiated messaging may be viable to reduce the number of kernel synchronization events (item 4).

The reason why CPU-initiated vs. GPU-initiated communication ranked least important among our priorities is because of the high cost of providing memory consistency for GPU memory. This cost is incurred regardless of the approach and dominates the overall communication overheads [18]. If this aspect of performance was improved, then the importance of increasing message size, and pipelining messages would be reduced.

VII. Related Work

Parallel machine models, *e.g.* PRAM [19], LogP [1], logGP [4], loPC [5], LogGPS [6], or logGOPS [8], have been instrumental in developing parallel algorithms both at the application and communication runtime levels. The development of these model is intended to provide a simple yet effective

abstraction for evaluating evolving architectural designs and to ease the assessment of algorithmic variants.

The advent of heterogeneous systems, especially with the use of hardware accelerators, brings back to the forefront the modeling question of these complex systems. Moving data between accelerator memories has been a significant bottleneck in distributed computing environments [20, 21]. Unlike earlier systems that rely mainly on CPU-initiated mechanisms [20], moving data residing on accelerator memories has recently involved novel mechanisms, including device-initiated [3, 12, 22–24] and hardware transparent migration using unified memory models [25, 26].

A data transfer initiated from an accelerator device could possibly traverse multiple hardware technologies [27], including NVlink, PCIe, QPI, X-Bus, IBV, each technology with its performance attributes, ordering constraints, overheads, making performance prediction a cumbersome task.

Striking a balance between accuracy and clarity is a challenge in developing a performance model. As such, we relied on a well-understood model, LogGOP [8], to consolidate the impact of multiple technologies that the data traverse during their journey between the memories of accelerator devices.

Most MPI implementations for distributed GPU programming focus on host-initiated techniques [28], which has been simplified with the introduction of unified virtual addressing [29]. RDMA-based programming frameworks, with their simple semantics and low overheads, enable efficient device-initiated distributed GPU programming. NVSHMEM [12] leverages the relaxed memory semantics of the SHMEM [30] programming abstraction to provide efficient inter-GPU communication. Numerous studies [3, 12, 22–24] studied the performance issues associated with scaling NVSHMEM without developing a performance model to guide algorithm development. In contrast, our study aims at not only guiding algorithm development but also identifying hardware bottlenecks with the greatest impact on the performance.

VIII. Future Work and Conclusions

a) Recent improvements to NVSHMEM: Since this work began, NVSHMEM performance has steadily improved. In the recent 1.0 version get operations are notably faster, with a reduction of 1.84 us for device initiated *get* operations compared to the results obtained in version 0.3. Furthermore, there are further improvements in development for small message shmem_g/get latencies which reduce roundtrip latencies to 6 us. These improvements are enabled by avoiding an additional consistency operation once the data is received by the requester.

b) Future work: Our model sets the foundations for what we expect in GPU communication for current architectures. However, we leave several areas for future work. First, we would like to examine the impact of multiple GPUs per node. While, many systems dedicate a NIC per GPU, the impact of sharing NIC resources is something we will explore in the future. Secondly, collective communications are an area that deserves further study. Collectives may be able to take advantage of hardware acceleration within the NIC and switches of the network. Furthermore, collectives must be tuned to reduce congestion as they generate more complex communication patterns. This also is related to predicting the impact of variability in communication, which we leave for future work.

c) Forward-looking projections: GPU vendors are actively working on techniques to reduce kernel launch times. In Fig 10 we show the impact of reducing kernel synchronization costs to 2us. Our data suggests that this would allow GPUs provide a benefit even for applications that spend 80% of time in communication. However we believe the biggest priority should be, using larger messages, pipelining and reducing the cost of memory consistency operations, which will enable a larger set of applications to port to GPUs successfully.

d) Summary of contributions: In this report we assessed the parameter values for two methods of GPU communication. Our findings suggest that for current systems, host controlled communication is generally preferred, especially if kernel launch overheads can be kept below 8us. Our results support work by Hamidouche and LeBeane [18] that a majority of communication time is spent synchronizing memory in preparation for communication rather than communication over the network itself. We prioritize the design considerations application developers should consider when porting to GPUs. We show that applications can expect to see speedups on next-generation GPU systems through a variety of techniques. Although GPU-communication has pitfalls, our models help navigate around them to deliver performance.

Acknowledgements

We would like to thank the Christopher Zimmer at Oak Ridge Leadership Computing for his help mapping the Summit network topology and Sarp Oral for use of the Summit node diagram. We would additionally like to thank Sreeram Potluri and Akhil Langer of Nvidia for their help in understanding the GPU architectures and how they relate to our observed performance. This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken, "Logp: Towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN*

symposium on Principles and practice of parallel programming, 1993, pp. 1–12.

[2] GPU-Direct RDMA, https://docs.nvidia.com/cuda/gpudirect-rdma/index.html, 2020.

[3] A. Goswami, A. Langer, and S. Potluri, "S9677 - nvshmem: A partitioned global address space library for nvidia gpu clusters," in *Nvidia GPU Technology Conference*. Nvidia, 2019.

[4] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: incorporating long messages into the LogP model–one step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95. New York, NY, USA: ACM, 1995, pp. 95–105. [Online]. Available: http://doi.acm.org/10.1145/215399.215427

[5] M. I. Frank, A. Agarwal, and M. K. Vernon, "Lopc: modeling contention in parallel algorithms," *ACM SIGPLAN Notices*, vol. 32, no. 7, pp. 276–287, 1997.

[6] F. Ino, N. Fujimoto, and K. Hagihara, "LogGPS: a parallel computational model for synchronization analysis," in *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, ser. PPoPP '01. New York, NY, USA: ACM, 2001, pp. 133–142. [Online]. Available: http://doi.acm.org/10.1145/379539.379592

[7] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm, "LogfP - a model for small messages in InfiniBand," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, april 2006, p. 6 pp.

[8] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim: simulating large-scale applications in the LogGOPS model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 597–604. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851564

[9] T. Groves, S. K. Gutierrez, and D. Arnold, "A LogP extension for modeling tree aggregation networks," in *2015 HPCMASPA in association with IEEE International Conference on Cluster Computing*, Sept 2015, pp. 666–673.

[10] T. Hoefler, A. Lichei, and W. Rehm, "Low-overhead loggp parameter assessment for modern interconnection networks," in *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.

[11] OFED, https://github.com/linux-rdma/perftest, 2020.

[12] S. Potluri, A. Goswami, D. Rossetti, C. Newburn, M. G. Venkata, and N. Imam, "Gpu-centric communication on nvidia gpu clusters with infiniband: A case study with openshmem," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 253–262.

[13] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 94–110, 2019.

[14] Mellanox, https://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_EN_Card.pdf, 2020.

[15] S. Oral, S. S. Vazhkudai, F. Wang, C. Zimmer, C. Brumgard, J. Hanley, G. Markomanolis, R. Miller, D. Leverman, S. Atchley *et al.*, "End-to-end i/o portfolio for the summit supercomputing ecosystem," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–14.

[16] Oak Ridge Leadership Computing Facility, https://www.olcf.ornl.gov/wp-content/uploads/2018/06/Summit_bythenumbers_FIN-1.pdf, 2020.

[17] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, and K. Yelick, "An evaluation of current high-performance networks," in *Proceedings International Parallel and Distributed Processing Symposium.* IEEE, 2003, pp. 10–pp.

[18] K. Hamidouche and M. LeBeane, "Gpu initiated openshmem: correct and efficient intra-kernel networking for dgpus," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 336–347.

[19] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, ser. STOC '78. New York, NY, USA: Association for Computing Machinery, 1978, p. 114–118. [Online]. Available: https://doi.org/10.1145/800133.804339

[20] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, "Efficient inter-node mpi communication using gpudirect rdma for infiniband clusters with nvidia gpus," in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 80–89.

[21] M. LeBeane, K. Hamidouche, B. Benton, M. Breternitz, S. K. Reinhardt, and L. K. John, "Gpu triggered networking for intra-kernel communications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126950

[22] M. S. Orr, S. Che, B. M. Beckmann, M. Oskin, S. K. Reinhardt, and D. A. Wood, "Gravel: Fine-grain gpu-initiated network messages," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126914

[23] M. Silberstein, S. Kim, S. Huh, X. Zhang, Y. Hu, A. Wated, and E. Witchel, "Gpunet: Networking abstractions for gpu programs," *ACM Trans. Comput. Syst.*, vol. 34, no. 3, Sep. 2016. [Online]. Available: https://doi.org/10.1145/2963098

[24] T. Gysi, J. Bär, and T. Hoefler, "dcuda: Hardware supported overlap of computation and communication," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 609–620.

[25] N. Sakharnykh, "Beyond GPU Memory Limits with Unified Memory on Pascal," 2017.

[26] M. Knap and P. Czarnul, "Performance evaluation of unified memory with prefetching and oversubscription for selected parallel cuda applications on nvidia pascal and volta gpus," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7625–7645, 2019. [Online]. Available: https://doi.org/10.1007/s11227-019-02966-8

[27] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 94–110, 2020.

[28] A. M. Aji, L. S. Panwar, F. Ji, M. Chabbi, K. Murthy, P. Balaji, K. R. Bisset, J. Dinan, W.-c. Feng, J. Mellor-Crummey, X. Ma, and R. Thakur, "On the efficacy of gpu-integrated mpi for scientific applications," in *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 191–202. [Online]. Available: https://doi.org/10.1145/2462902.2462915

[29] K. V. Manian, A. A. Ammar, A. Ruhela, C.-H. Chu, H. Subramoni, and D. K. Panda, "Characterizing cuda unified memory (um)-aware mpi designs on modern gpu architectures," in *Proceedings of the 12th Workshop on General Purpose Processing Using GPUs*, ser. GPGPU '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 43–52. [Online]. Available: https://doi.org/10.1145/3300053.3319419

[30] S. W. Poole, O. Hernandez, J. A. Kuehn, G. M. Shipman, A. Curtis, and K. Feind, *OpenSHMEM - Toward a Unified RMA Model*. Boston, MA: Springer US, 2011, pp. 1379–1391. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_490

ARTIFACT DESCRIPTION/ARTIFACT EVALUATION

We use two sets of benchmarks to derive our LogP model parameters: OFED and NVSHMEM Perftests.

OFED: For the cases where it is desirable to reduce software and CPU overheads as much as possible, werely on the OFED performance tests. These are written directly using the low level Infiniband Verbs (IBV) API to minimize software overhead. In the read_lattests, work requests are posted one at a time with a "delta" value measuring the time between consecutive post events. For any message k ithat is posted, the message k_{i+1} is not posted until the completion queue signifies success with IBV_WC_SUCCESS, reporting half round trip time (RTT/2). All OFED Perftests utilize CPU-initiated communication, but it includes CPU-to-CPU as well as GPU-to-GPU benchmarks, which allows us to compare latency and bandwidth as the data path changes fromthe CPU's DRAM memory to the GPU's HBM2 memory.)

NVSHMEM Perftest: The NVSHMEM communication library provides lightweight communication operations for accessing GPU memory. NVSHMEM is written using the IBV interface. NVSHMEM provides two separate control paths for communication. One is invoked directly by the CPU (e.g.*_get_on_stream) and the other is invoked directly within a GPU kernel (e.g.*_get_nbi_block). As discussed in Section II, the GPU-initiated interface currently relies on progress threads on the CPU to help with sending the necessary Infiniband requests to the NIC. We compare the get and put-based NVSHMEM perftests included with NVSHMEM 0.3.0. A similar device initiated control path is not available in any MPI implementation to date. There is an option in the benchmark to discard the first k iterations (this is common in many communication benchmarks to avoid reflecting cache performance). We found that if the warm-up phase is disabled, get latency could be as high as 8 us. For our parameter assessment we enable the default warm-up of 20 iterations. We note that NVSHMEM supports intra-node messages between GPUs sent over PCIe or NVLink. However this work is focuses on device vs. host-initiated inter-node communication.

URL/DOI List:
https://gitlab.com/h4u5/gpu-comm,
3988559446a4c17125654d173cece6f71ef57a5a
https://github.com/berkeley-container-library/bcl,
2ac6da10c96375e2750bb36239f86ee2cb32785a

Relevant hardware details:
Summit System OLCF, IBM Power9, OS version 4.14.0-115.21.2.el7a.ppc64le, gcc-6.4.0, NVSHMEM-0.3, spectrum-mpi-10.3.1.2, cuda-10.1.168

Paper Modifications:
Infiniband perftest
https://github.com/linux-rdma/perftest,
6369e620429197f7cc0b6bfcb9734fe70f0b92f0

The following variables were specified:
GRB_SPARSE_MATRIX_FORMAT=1
NVSHMEM_SYMMETRIC_SIZE=9GB