Evaluating the Viability of LogGP for Modeling MPI Performance with Non-contiguous Datatypes on Modern Architectures

Nicholas Bacon
Department of Computer Science
University of New Mexico
Albuquerque, New Mexicoddd, USA
nbacon@unm.edu

Patrick G. Bridges
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, USA
patrickb@unm.edu

Scott Levy
Center for Computational Research
Sandia National Laboratories
Albuquerque, New Mexico, USA
sllevy@sandia.gov

Kurt Ferreira
Center for Computational Research
Sandia National Laboratories
Albuquerque, New Mexico, USA
kbferre@sandia.gov

Amanda Bienz
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, USA
bienz@unm.edu

ABSTRACT

Modern architectures and communication systems software include complex hardware, communication abstractions, and optimizations that make their performance difficult to measure, model, and understand. This paper examines the ability of modified versions of the existing Netgauge communication performance measurement tool and LogGOPS performance model to accurately characterize communication behavior of modern hardware, MPI abstractions, and implementations. This includes analyzing their ability to model both GPU-aware communication in different MPI implementations and quantifying the performance characteristics of different approaches to non-contiguous data communication on modern GPU systems. This paper also applies these techniques to quantify the performance of different implementations and optimization approaches to non-contiguous data communication on a variety of systems, demonstrating that modern communication system design approaches can result in widely-varying and difficult-to-predict performance variation, even within the same hardware/communication software combination.

ACM Reference Format:
Nicholas Bacon, Patrick G. Bridges, Scott Levy, Kurt Ferreira, and Amanda Bienz. 2023. Evaluating the Viability of LogGP for Modeling MPI Performance with Non-contiguous Datatypes on Modern Architectures. In *Proceedings of EuroMPI2023: the 30th European MPI Users' Group Meeting (EUROMPI '23), September 11–13, 2023, Bristol, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3615318.3615326

1 INTRODUCTION

Complex hardware architectures, communication abstractions, and system software optimizations can make the performance of modern high-performance communication systems difficult to understand and predict. Communication performance variation between and within systems and implementations discourages programmers from using more sophisticated abstractions, instead causing them to fall back to the lowest common denominator programming abstractions. In the case of MPI, this generally results in programmers defaulting to point-to-point sends and receives using contiguous buffers instead of more sophisticated abstractions.

It can be difficult to accurately quantify and model the performance of even conceptually simple communication abstractions on modern high-performance computing systems. The MPI datatype abstraction, for example, was designed to ease communication and improve performance when sending, receiving, and manipulating (e.g., reducing) non-contiguous data, and has been an element of the MPI standard from its inception. Unfortunately, datatype performance on modern GPU-based systems is often poor [7], complex to optimize [18, 11, 21], and can vary widely system-to-system, implementation-to-implementation, and even call-to-call based on the optimizations implemented [21].

This paper examines the ability of the well-known `LogGOPS` communication performance model [9] and Netgauge measurement tool [8] to model, measure, and analyze the performance of modern GPU-based HPC systems, particularly when using more complex communication abstractions such as MPI derived datatypes. In doing so, it describes the following contributions:

- An analysis of the suitability of `LogGOPS`-based models to quantify the performance of approaches to communicating data, including non-contiguous data, in modern GPU-based high-performance systems;
- An approach and related set of open-source tools for measuring these model parameters when transferring contiguous and non-contiguous data;
- An evaluation of the model's ability to capture key performance characteristics and predict the performance of contiguous and non-contiguous communication on multiple modern hardware platforms and MPI implementations; and,
- An empirical comparison of the performance of different non-contiguous data communication implementations on multiple systems using this model.

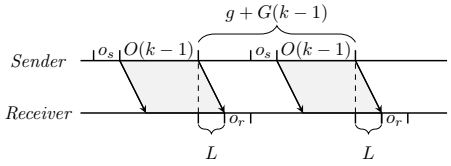
The remainder of this paper is organized as follows. Section 2 begins with a discussion of the state-of-the-art techniques for modeling

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

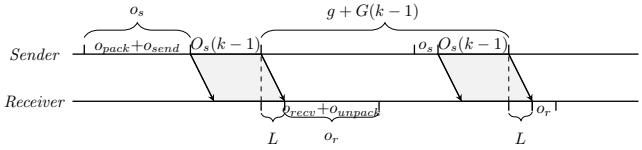
EuroMPI'23, September 2023, Bristol, UK

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0913-5/23/09…$15.00
https://doi.org/10.1145/3615318.3615326



(a) Original LogGOPS model



(b) LogGOPS model w/ packing & unpacking message data

Figure 1: Example of LogGOPS model for the transmission of two back-to-back k-byte messages

and measuring the performance of high-performance communication systems. Section 3 then follows with a discussion of the challenges in modeling communication performance on modern communication middleware and architectures and the proposed approach for considering LogGPS, LogGOPS, and Netgauge modeling and measurement in these systems. Section 4 describes the experimental setup used to evaluate this modeling and measurement approach on modern systems, and Section 5 presents and discusses the results of these experiments. Finally, Section 6 discusses related research in this area, and Section 7 summarizes paper results and discusses potential directions for future research.

2 BACKGROUND

2.1 The LogGOPS model

The LogP [6] family of models (*see e.g.*, LogGP [1], LogGPS [10]) are a group of abstract models of inter-process communication in distributed systems. LogGOPS [9] is a member of this family that extends the LogGPS model by adding the cost of per-byte communication overheads (O). The complete LogGOPS model calculates communication performance as a linear function of the following eight variables: (i) k, the message size, measured in bytes; (ii) L, the latency of sending a message between processes; (iii) o, the per-message processor cost of sending a message; (iv) g, the per-message cost of initiating a network send; (v) G, the per-byte cost of initiating a send; (vi) O, the per-byte processor cost of sending a message; (vii) P, the number of processes in the system; and (viii) S, the threshold for using synchronized sends (e.g., the rendezvous protocol in MPI).

The original LogP model used a single value (o) to represent both send and receive overheads. However, as a practical matter, send and receive overheads may not always be the same [14]. Therefore, in this paper, we follow the established practice of distinguishing between the send overhead (o_s) and the receive overhead (o_r).

Figure 1a shows a simple example of sending two back-to-back k-byte messages between a *Sender* and *Receiver*. In networks that allow communication-computation overlap, the network and the CPU can progress independently. The G and g terms are used to determine the network time required for a send and the O and o_s terms are used to determine the processor time required for a send. The time required to complete a send operation is the maximum of the network time and the processor time (i.e., the point at which both the network and the processor have completed the work necessary for a send). We discuss our modifications to this model in Section 3.1.

2.2 Netgauge

Netgauge [8] is a tool for measuring and characterizing performance in high-performance networks. It supports measurement on different networks using several communication patterns. For the purposes of the analysis in this paper, Netgauge supports the ability to measure LogGP parameters using MPI.

2.3 MPI Derived Datatypes

The MPI standard [13, Table 3.2] includes several predefined types for use in the MPI API. These types roughly correspond to standard C data types (e.g., `MPI_CHAR`). The MPI standard [13, Chapter 5] also provides the ability for programmers to use these basic types to construct more sophisticated types. In addition to allowing programmers to group multiple basic types into new types, derived datatypes also allow programmers to define an associated memory layout (e.g., `MPI_TYPE_CREATE_CONTIGUOUS` for data that is contiguous in memory, `MPI_TYPE_CREATE_VECTOR` for data that is strided in memory). Given that derived datatypes may be non-contiguous in memory, send and receive operations using derived types generally perform additional operations to pack and unpack derived datatypes into (and out of) contiguous memory buffers as they are sent and received over the network.

3 APPROACH

Our primary modeling goal for modern systems is to quantify the additional costs associated with GPU and non-contiguous data communication, as there are many different approaches for handling these cases. For example, GPU data can be packed and unpacked in host memory or they can be packed in GPU memory (e.g., by launching appropriate pack and unpack kernels on the device). Additionally, CUDA-aware MPI enables send and receive operations to reference message buffers in host or device memory. Different approaches may also use different optimization strategies (e.g., pipelined transfers or scatter/gather operations). Each of these choices implicates a different sets of performance characteristics. In the remainder of this section, we describe our approach to modeling these costs with a straightforward extension of the LogGOPS network communication model using model parameters that we measure using a modified version of the Netgauge tools.

3.1 Applying LogGOPS to GPU communication and Non-Contiguous Data

Figure 1b shows a simple example of sending two back-to-back k-byte messages using our simple extension of the LogGOPS model. The principal difference between this model and the original LogGOPS model (*see* Section 2.1) is that, unlike the original model, we explicitly account for the costs associated with moving data between host and device memory and assembling non-contiguous data into contiguous message buffers. To capture the impact of these costs,

we model the per-message overheads (o_s and o_r) and per-byte overhead (O_s and O_r) to include: (i) the time required for sending messages to (o_{send}), and receiving messages from (o_{recv}), the network; and (ii) the costs associated with preparing non-contiguous data for transmission (o_{pack}) and the costs associated with processing non-contiguous data after reception (o_{unpack}). The costs include datatype packing or unpacking (including launching kernels to pack or unpack data directly in device memory), copying data between host and device memory, creating scatter-gather lists, or other similar per-message or per-byte costs associated with every send. We seek primarily to use this model to understand how GPUs and MPI datatypes impact per-message and per-byte processor overheads. For the remainder of this paper, we refer to the activities associated with managing non-contiguous data in GPU memory as packing and unpacking. However, we recognize they can include other overheads associated with sending from, and receiving to, GPU memory. We also note that these overheads can vary for different data transfers, depending for example on the layout of data being sent or received. For notational simplicity, we also assume for the remainder of the paper that o_s, o_r, and O_s include the costs associated with packing and unpacking non-contiguous data in GPU memory. That is, o_{pack} and o_{unpack} are not calculated explicitly but rather are conceptual costs added to o_s and o_r by the kernel running on the GPU.

Netgauge only provides estimates of the LogGPS parameters. As a result, its measurements combine per-message and per-byte overheads. Its estimates of per-message send and receive overhead change as a function of the message buffer size. In this section, we refer to these combined measurements as $\hat{o}_s$ and $\hat{o}_s$. To obtain the per-byte overhead parameter, we perform a linear fit of the per-message measurements from Netgauge. We use the coefficients of the linear fit of the send overhead to model the per-byte send overhead (O_s) as the slope and the per-message send overhead (o_s) as the y-intercept.

In the LogGOPS model, o_r is typically assumed to be constant. However, we have empirically shown that the value of o_r reported by Netgauge is a function of message size. Therefore, in our LogGOPS model, we also perform a linear fit of the receive overhead reported by Netgauge. We use the coefficients of the linear fit of the receive overhead and model the per-byte receive overhead (O_r) as the slope and the per-message receive overhead (o_r) as the y-intercept. The combination of these two coefficients allows us to estimate the total receive overhead based on the size of the received message buffer. Given this background, we use the LogGPS model to estimate the time required to complete a ping-pong operation with a k-byte buffer ($t_{ping\text{-}pong}$) as:

$$t_{ping\text{-}pong} = 2(\max(\hat{o}_s, g) + Gk + L + \hat{o}_r)$$

where $\hat{o}_s = \hat{o}_{send} + \hat{o}_{pack}$, and $\hat{o}_r = \hat{o}_{recv} + \hat{o}_{unpack}$ are the per-message size estimates of send and receive overheads from Netgauge, respectively.

Similarly, given the linear fit of the overhead estimates from Netgauge, we can use the LogGOPS model to estimate $t_{ping\text{-}pong}$ for a k-byte buffer as:

$$t_{ping\text{-}pong} = 2(\max(o_s + O_s k, g + Gk) + L + O_r k + o_r)$$

where O_s and o_s are the coefficients of the linear fit of $\hat{o}_s$, and O_r and o_r are the coefficients of the linear fit of $\hat{o}_r$.

3.2 Using Netgauge to Model Derived Datatypes in GPU Memory

To measure LogGOPS parameters on modern systems, we use multiple runs of a version of Netgauge that we modified to send and receive messages composed of a configurable datatype in either CPU or GPU memory. Our modifications to Netgauge include changes to take into account the *size* of the datatype being used; specifically, this modified version of Netgauge assumes that the costs of sending an MPI datatype is dependent on its *MPI size*, not its *MPI extent*. In the work described in this paper, we focus on MPI derived datatypes constructed with `MPI_Type_vector` using configurable block count, block size, and block stride parameters.

In addition, we also modified Netgauge to use a configurable *delay* parameter when estimating the overhead (o) and inter-message gap (g) parameters. Because these two parameters act in parallel during message transmission, Netgauge adds this delay to some round trips to measure the two quantities separately. The original version of Netgauge used a delay parameter that was much too short to account for the high overheads that some MPI implementations (e.g. Spectrum) encountered when sending large buffers of derived datatypes. Finally, we have made this modified version of NetGauge publicly available [2].

Overall, our measurement approach is based on the that described in the original Netgauge and LogGOPSim papers:

(1) Choose a target memory system (i.e. host or device memory), datatype, and range of bytes transmitted to evaluate.
(2) Estimate a delay parameter sufficient to accurately distinguish between inter-message gap and per-message overhead; we generally use twice the expected time to send the largest message size being tested with the chosen datatype. We do not currently attempt to dynamically choose this delay parameter.
(3) Compute the base LogGOPS parameters for communication with a simple primitive type (e.g., `MPI_FLOAT`) with a contiguous datatype (i.e. a single element MPI vector type with $stride == blocklength$) using the standard approach for LogGOPS measurement [9].
    (a) Run Netgauge with a primitive datatype and counts to obtain basic LogGPS network communication parameters for the memory being used and to identify where an increase in the buffer size results in a protocol switch (e.g., eager to rendezvous).
    (b) For each protocol section, perform a linear fit of the original LogGPS $\hat{o}_r$ and $\hat{o}_s$ parameters to compute the base o_r, O_r, o_s, and O_s parameters of the LogGOPS model.

Note that the LogGPS parameters include a different overhead value for each $(datatype, bytes\_sent)$ tuple, while the LogGOPS parameters are estimated piecewise for each protocol switch section of the model. Our approach also assumes that changes to the packing approach that impact performance occur at the *same* boundaries as network protocol switches.

4 EXPERIMENTAL SETUP

To evaluate the challenges and approach described in the previous section, we chose a set of systems, MPI implementations, and MPI derived datatypes to evaluate in the following manner:

(1) Estimate the LogGPS and LogGOPS model parameters of CUDA-aware MPI communication of GPU buffers with different MPI datatypes on these systems using our modified version of Netgauge described in the previous section;
(2) Directly measure the MPI ping-pong round trip communication time for the same systems, MPI implementations, and datatypes.
(3) Compare the relative error between the measured median round trip time and the predicted round trip time computed using the formulas presented in Section 3. We use the arithmetic mean when averaging relative errors of different sizes for comparison between datatypes or MPIs.

The remainder of this section provides additional details on the experimental setup, including the systems and MPI implementations tested, the datatypes evaluated, and how round trip communication times are measured.

4.1 System Configurations

We collected data for multiple MPI implementations on both a high-end supercomputer system and a mid-range HPC cluster:

LLNL Lassen: Lawrence Livermore National Laboratory's Lassen supercomputer uses IBM POWER9 3.80 GHz CPUs, NVIDIA V100 GPUS, and Mellanox EDR InfiniBand network interface cards. CPUs, GPUs, and NICs are interconnected using bidirectional 150GB/s NVLINK bus links. Lassen nodes are configured to provide `gdrcopy` support [15] for direct NIC/GPU communication. Our modified version of Netgauge was compiled with GCC 8.3.1 and CUDA 11.1.1. We used the following system-provided MPI implementations on this system:

- **Lassen/Spectrum**: Spectrum MPI module version 2020.08.19, the IBM-provided MPI implementation based on Open-MPI.
- **Lassen/MVAPICH2**: MVAPICH2-GDR module version 2021.05.29-cuda-11.1.1, the LLNL-provided version of MVAPICH [17].

SNL Glinda: Sandia National Laboratories' Glinda cluster is composed of compute nodes built around AMD EPYC 2.80 GHz CPUs, NVIDIA A100 GPUS, and NVIDIA Mellanox ConnectX-6 2xHDR InfiniBand network interface cards. CPUs, GPUs, and NICs are connected via PCIe. `gdrcopy` support is not provided on these nodes. Our modified version of Netgauge was compiled with GCC 9.3.0 and CUDA 11.1.0. We used the following system-provided MPI implementations on this system:

- **Glinda/OpenMPI4**
- **Glinda/OpenMPI4+TEMPI**: OpenMPI 4.1.4 was built with the TEMPI datatype engine (git commit hash `9e623f0` from github.com/cwpearson/TEMPI) to improve GPU datatype handling. The provided `measure-system` binary was used to calibrate packing performance tradeoffs prior to testing.

4.2 Datatype Evaluation Strategy

Our datatype evaluation focuses on testing one of the simpler MPI derived datatypes, `MPI_Type_vector`. This datatype is frequently used to specify blocks of strided data, for example sub-portions of arrays. MPI vector types are specified with a base `MPI_Datatype` and a $(blockcount, elementcount, stride)$ tuple that specifies, respectively, the number of blocks of contiguous elements; the number of contiguous elements in each block, and the number of base type instances between the start of each block. For example, a $(1, 1, 1)$ vector is simply the underlying base type, while a $(2, 2, 4)$ vector contains two blocks of two base type elements with two unused elements at the end of each block.

The experiments described in this paper use either a contiguous buffer of primitive datatypes, specifically `MPI_FLOAT`, or vector datatypes with `MPI_FLOAT` as the base type. In particular, we use vector configurations varying from $(1, 1, 4)$ to $(4, 4, 4)$ with a fixed stride of 4 `MPI_FLOAT`s. The sizes of the resulting datatypes ranged from 4 bytes (i.e., $(1, 1, 4)$) to 64 bytes (i.e., $(4, 4, 4)$). This choice of datatypes tests both sparse and contiguous datatypes, and both single block and multi-block datatypes. For brevity, the results presented in the next section focus on eight `MPI_Type_vector` datatypes: $(1, 1, 4)$, $(1, 2, 4)$, $(1, 3, 4)$, $(1, 4, 4)$, $(4, 1, 4)$, $(4, 2, 4)$, $(4, 3, 4)$, $(4, 4, 4)$. This range allows us to test how well the LogGPS models capture performance when varying send/recv count, datatype block count, and sparsity.

4.3 Ping-pong Performance Measurement

We measure the ping-pong time for each datatype with power-of-two `MPI_Send`/`MPI_Recv` element counts ranging from 1 (2^0) element to 262,144 (2^{18}) elements. In each test, the given number of datatypes is exchanged as a sequential ping-pong (i.e., the same number of bytes is sent and received in each direction) between two nodes 100 times, and the total amount of time for this data exchange is measured and used to compute the average round trip time for the trial. We run 10 trials for each datatype with each of the four system configurations described in Section 4.1. Each of these trials is repeated 10 times, and we report the median round trip time of the set of trials as the measured value. For analysis, test results are grouped by the total number of bytes sent, allowing for the comparison between datatypes that have dissimilar layouts but result in the transfer of the same amount of data.

4.4 LogGPS Performance Measurement

We measure the LogGPS parameters using our modified version of Netgauge. We use the same datatype and element count range as the ping-pong performance measurements. We do a linear fit on the raw LogGPS $\hat{o}_r$ and $\hat{o}_s$ parameters to compute the o_r, O_r, o_s, and O_s parameters of the LogGOPS model. Estimated round trip communication times are then estimated from the measured LogGPS and LogGOPS parameters using the formulas presented in Section 3.1.

5 RESULTS AND ANALYSIS

To evaluate the usefulness of the approach described in the previous section for quantifying and analyzing the performance of MPI

implementations on modern systems, we sought to answer three specific research questions:

- How effectively do the LogGPS and LogGOPS models quantify communication performance of MPI implementations on modern GPU systems when using simple primitive datatypes?
- How effectively do the LogGPS and LogGOPS models quantify the performance of communication using MPI derived datatypes?
- How do the LogGPS and LogGOPS parameters for different MPI implementations change across a range of datatypes and message sizes?

We evaluated these questions on the systems, MPI implementations, and MPI datatypes and message sizes described in the previous section.

5.1 LogGPS modeling of GPU communication performance

In this subsection, we analyze how accurately the LogGPS and LogGOPS models capture key performance features of the MPI implementations and systems described in Section 4.1 with primitive MPI datatypes. LogP-derived models have a relatively simple network model where communication is modeled as a sequence of messages moving through a single network path. Because of this, it is not obvious that these models can reasonably predict even the general trends of modern network communication, much less precisely predict round trip times.

Figure 2, shows the full range of model predictions of four of the MPI implementation using both the LogGPS and LogGOPS and the relative prediction error at each data size. The models measured using Netgauge capture some key features of MPI performance, particularly for mid-sized messages. However, they also tend to consistently over-predict ping-pong communication times, particularly for very large and very small messages. This is particularly true in both Lassen test cases, which consistently have a relative error of 60% or higher. The complex network topology, GPUDirect network access, and highly-tuned nature of MVAPICH2 mean these cases may not be a good match for the assumptions implicit in LogP-based communication models. Despite this, however, these models still capture general communication cost trends, which is often more important in communication modeling than absolute or relative error.

5.2 LogGPS and LogGOPS Modeling of GPU Datatype Communication Performance

In this subsection, we analyze how accurately the LogGPS and LogGOPS models captured key performance features of the MPI implementations and systems described in Section 4.1 with different MPI datatypes. Based on the results of the previous subsection, we focus primarily on how well the LogGPS and LogGOPS models perform when modeling abstractions with higher CPU overheads. While LogGOPS struggles to capture the performance of complex modern networks, it is a separate question as to whether it can reasonably model the communication overheads associated with communication that uses MPI derived datatypes.

Figures 3 and 4 show the performance of Glinda/OpenMPI4, Glinda/OpenMPI4+TEMPI, Lassen/MVAPICH2, and Lassen/Spectrum on two different vector datatypes: 4-block vector datatypes with either 4 contiguous floats per block of 32 bytes (i.e., (4, 4, 4)) or 1 sparse float per block of 32 bytes (i.e, (4, 1, 4)). In general and as with primitive datatypes, the models generally track measured communication performance but overestimate ping-pong times. However, the model is *more accurate* for communication using more expensive sparse datatypes where datatype packing/unpacking costs dominate network communication costs. Also, note the poor absolute performance of Spectrum on multi-block sparse datatypes; this is a known problem with these implementations that other researchers have also observed [18].

Finally, Figure 5 shows the arithmetic means of the LogGPS relative prediction error for each of the datatypes we tested on each system/MPI combination. First, in all cases except Spectrum MPI, accuracy is relatively poor because the MVAPICH2 and OpenMPI 4.1.4 datatype engines do not introduce significant additional overheads. However, the relative error is generally smaller for 4-block datatypes than 1-block datatypes, especially for sparse 4-block datatypes (i.e., (4, 1, 4), (4, 2, 4), and (4, 3, 4)) with MVAPICH2 and Spectrum MPI. In particular, LogGPS is much more accurate for Spectrum MPI when modeling the performance of the costly multi-block non-contiguous datatypes, but exhibits the same systematic relative errors of all LogGPS models on modern systems in other cases. LogGOPS is also more accurate for MVAPICH2 with multi-block sparse datatypes for which packing/unpacking costs are more significant than communication costs.

In general, these data show that LogGPS and LogGOPS modeling is *more* accurate when datatype packing and unpacking costs are high compared to network communication costs. In addition, the model also successfully captures the general shape and trends of GPU datatype communication. As a result, we conclude that our modified Netgauge-measured LogGPS parameters appear to: (1) accurately model packing and unpacking costs; and (2) continue to systematically overestimate network communication costs similar to the original Netgauge. As a result, Netgauge and LogGPS are very useful for quantifying communication computational overheads *despite the fact* that they have significant challenges accurately quantifying communication network costs on modern systems.

5.3 Analyzing Datatype Engine Performance

As described in the previous subsection, the LogGPS and LogGOPS parameters measured by our modified Netgauge version can accurately quantify datatype engine behavior of modern MPI implementations. In this subsection, we use this capability to analyze and compare the performance of the MPI and system configurations described in Section 4.1.

Figure 6 shows the computed receive overhead per byte measured in microseconds per byte for MPIs running on the LLNL Lassen and SNL Glinda systems with both single block and multi-block vector datatypes. Note the use of a logarithmic scale on the y-axis in this chart to show the full range of the data and the extent to which it varies both between datatypes for a given MPI implementation and between MPI implementations.

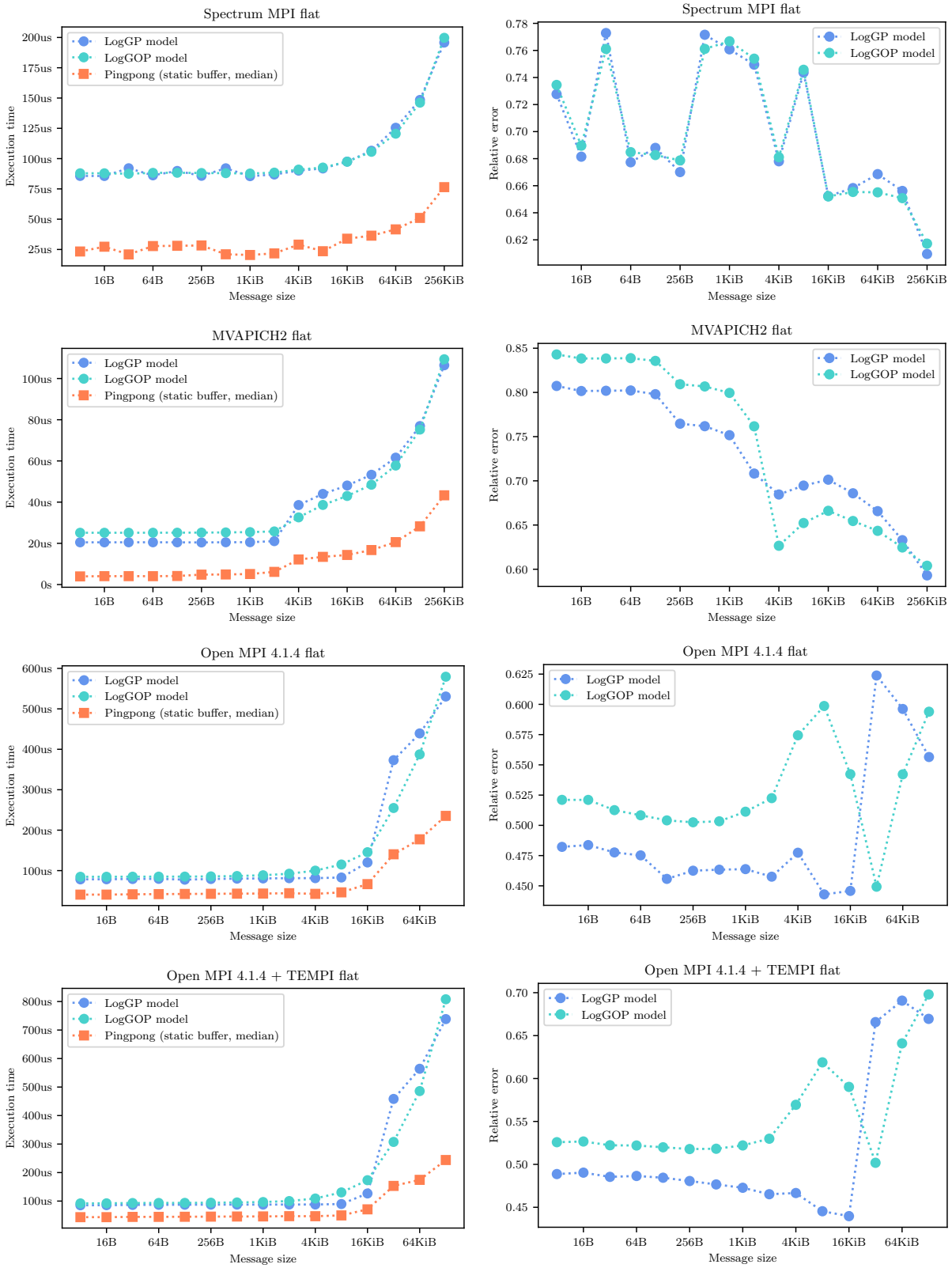


Figure 2: LogGPS and LogGOPS modeling contiguous `MPI_FLOAT` Ping-pong Performance.

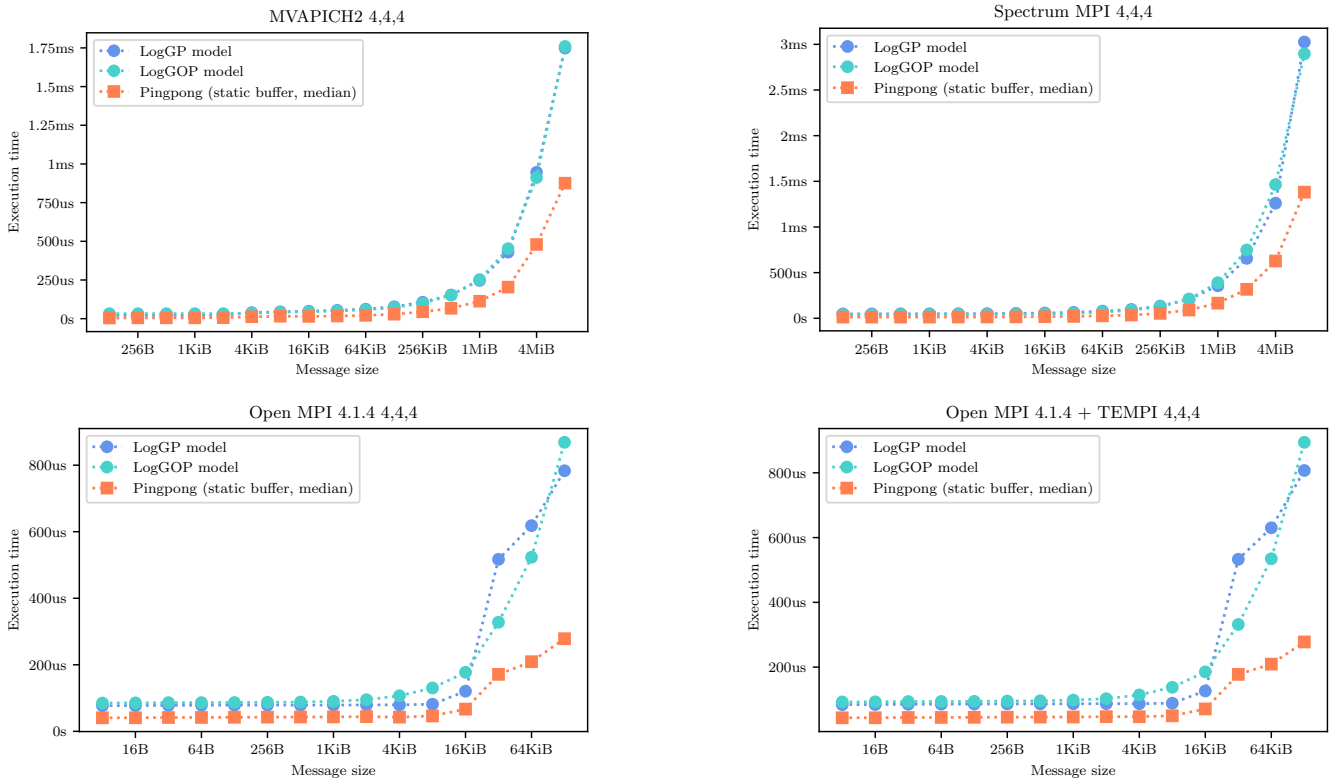


Figure 3: Modeled versus measured ping-pong datatype performance of a 4-block contiguous vector datatype

As shown in the figure, the results for Spectrum MPI vary by three orders of magnitude between single-block datatypes and multi-block datatypes when receiving sparse data. This accounts for the high ping-pong times in Figure 4. MVAPICH2 also shows increased overhead in per-byte overhead for multi-block sparse datatype reception (approximately one order of magnitude), though not to the degree shown by Spectrum MPI. In contrast, the results for OpenMPI and OpenMPI+TEMPI vary less between single block and multi-block sparse and contiguous dataypes. Send overhead per byte data, *see* Figure 7, show similar results.

The data in Figures 6 and 7 shows differences between the way that these MPI implementations handle datatypes. For the 1-block datatypes (left subfigures in Figures 6 and 7), the per-byte overheads are relatively constant between the dense datatype $(1, 4, 4)$ and the sparse datatypes, $(1, 1, 4)$, $(1, 2, 4)$, and $(1, 3, 4)$. This suggests that these MPI implementations are not exploiting that $(1, 4, 4)$ specifies a contiguous layout in memory (i.e., no packing or unpacking is required). In contrast, for the 4-block datatypes (right subfigures in Figures 6 and 7), the per-byte overheads for MVAPICH2 and Spectrum MPI are significantly lower for the dense datatype $(4, 4, 4)$ than for the sparse datatypes, $(4, 1, 4)$, $(4, 2, 4)$, and $(4, 3, 4)$, suggesting that these implementations are able to exploit the fact that $(4, 4, 4)$ specifies a layout that is contiguous in memory. The data for the two Open MPI configurations show that, like the 1-block datatypes, the per-byte overheads are relatively constant between the dense datatype and the sparse datatypes.

6 RELATED WORK

6.1 Modern Datatype Abstractions

MPI datatypes [13, Chapter 5] are the most well-known and commonly-used datatype abstractions, but other HPC communication systems support similar features. For example, GASNet-EX [12] provides support for user-defined data types in reduction operations. Similarly, UCX [16] provides support for generic data types that enable integration with MPI datatype engines.

6.2 Datatype Performance Modeling

Sun et al. [20] use the Memory-LogP model to understand the overhead of transferring non-contiguous data types. Based on their analysis, they develop an approach based on the MPI Profiling interface in which they use heuristics to identify data types in `MPI_Send` operations where optimization is possible.

6.3 Datatype Performance Optimization

Several researchers have implemented new approaches for optimizing datatype performance in modern communication systems. TEMPI [18], for example, is an MPI interposer library that converts non-contiguous data types in GPU memory into contiguous data to improve communication performance. It includes fine-grained measurement-based optimization to tune packing. Similarly, Yaksa [11] is a datatype engine that helps users and communication libraries more effectively manage and manipulate datatypes

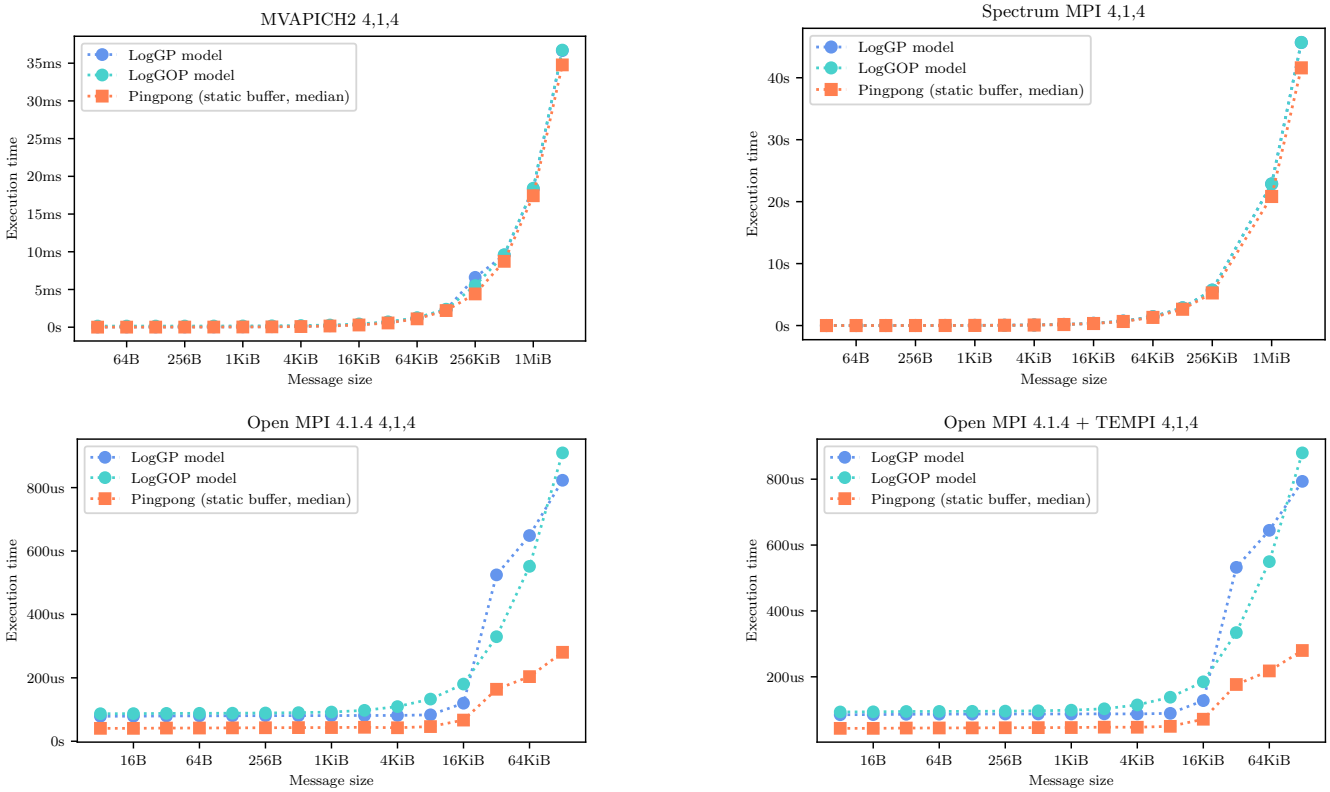
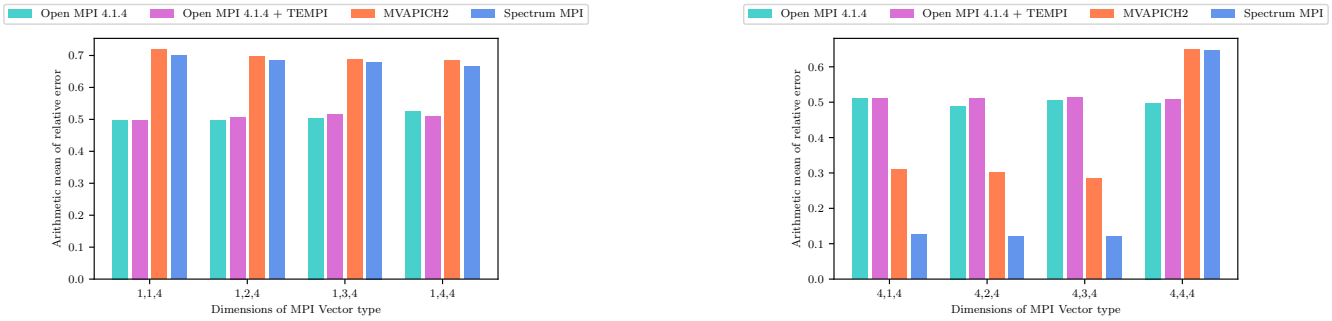


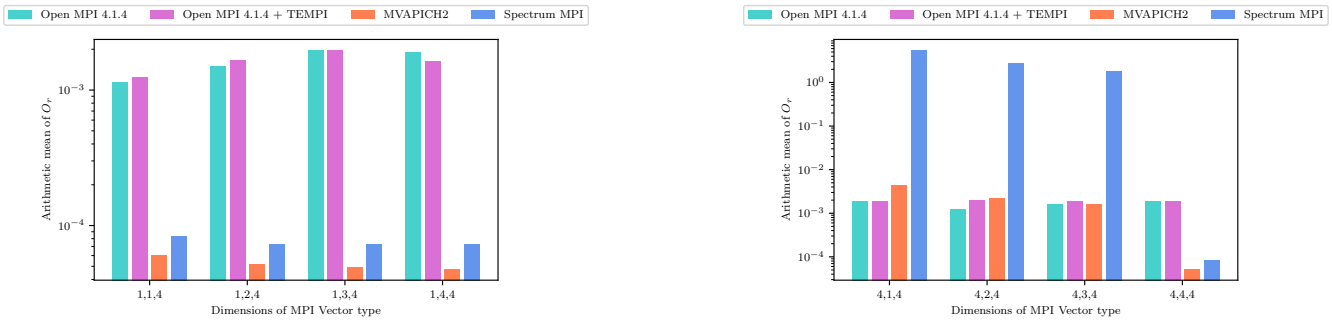
Figure 4: Modeled versus measured ping-pong datatype performance of a 4-block sparse vector datatype



(a) 1-block datatypes **(b) 4-block datatypes**

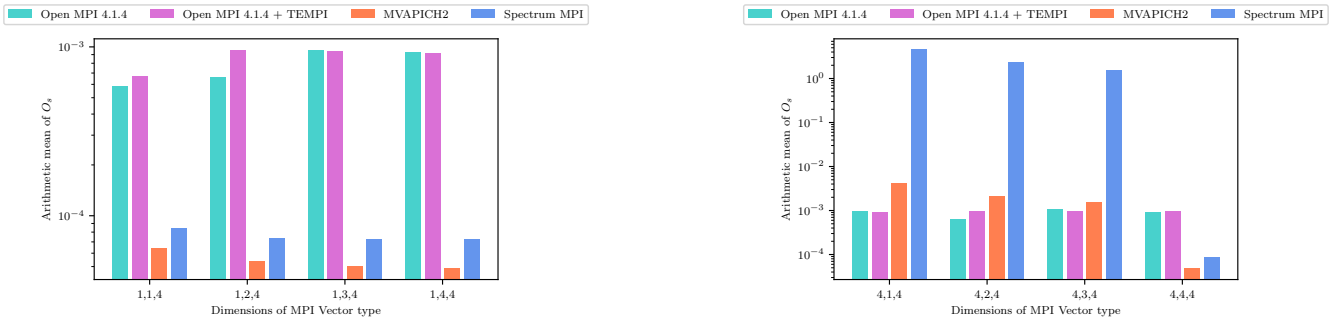
Figure 5: Arithmetic mean of relative error for `LogGP` model with one block and four block vector datatypes

in non-contiguous memory. Yaksa provides support for both CPUs and GPUs and has been integrated into MPICH. Finally, a wide range of datatype optimization approaches have been proposed for MVAPICH2. For example, Wang et. al [23] describe an approach that offloads packing and unpacking operations to the GPU and uses pipelining to overlap communication operations. Similarly, HAND [19] is a framework that uses CUDA GPU kernels to pack and unpack non-contiguous data types in GPU memory to improve communication performance and to reduce the need for hand-tuned, application-specific packing kernels. Finally, Suresh et al. [21] examine the potential benefit of using the scatter-gather functionality of Infiniband HCAs to manage transfers of MPI datatypes that are non-contiguous in GPU memory. function to improve performance. Similar approaches have also been examined for non-contiguous transfers in the GASNet-EX communication system [4].



(a) 1-block datatypes **(b) 4-block datatypes**

Figure 6: LogGOPS estimated receive overhead per byte (O_r) for varying vector datatypes, MPI implementation, and systems.



(a) 1-block datatypes **(b) 4-block datatypes**

Figure 7: LogGOPS estimated send overhead per byte (O_s) for varying vector datatypes, MPI implementation, and systems.

6.4 Data Transfer Models

Finally, various models have been developed to study the performance of specific elements of GPU communication systems. Van-Werkhoven et al. [22] use LogGP to model the time required to transfer data between CPU and GPU memory via the PCIe bus, and Boyer et al. [5] used a simple latency plus bandwidth model of transfers of data between CPUs and GPUs over the PCIe bus. More recently, Bienz et al. [3] used a latency plus bandwidth model to measure the performance of multiple GPU communication datapaths in modern multi-GPU systems. Unlike the work described in this paper, none of these approaches attempt to model end-to-end GPU communication performance or assess the suitability of their models for end-to-end performance analysis.

7 CONCLUSION

The results presented in this paper show the challenges and opportunities faced in understanding, measuring, and modeling the performance of communication in modern HPC middleware and hardware systems. In particular, these results demonstrate: (1) the limitations LogP-based models for predicting communication performance on modern hardware, and (2) their continuing suitability for quantifying the computational overheads associated with communication abstraction implementations. In addition, these results quantify and highlight the large range of performance in the implementations of key communication abstractions on modern systems.

A wide range of directions exists for future work building on these results. First, the results in this paper focus only on the point-to-point communication portions of the LogGPS and LogGOPS models; studies examining both the qualitative and quantitative accuracy of this model for collective (e.g., broadcast) communication on modern systems that includes hardware collective support would be an interesting direction for future work. Second, the presented results highlight the need for new communication performance models that more accurately capture the communication characteristics of modern hardware systems. Finally, studies modeling the performance characteristics of additional common or emerging communication abstractions, for example, GPU kernel-triggered or stream-triggered communication, are also potentially interesting.

ACKNOWLEDGMENTS

This research was supported in part by an award from the National Science Foundation under grant number OAC-2103510, and by the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under the Predictive Science Academic Alliance Program (PSAAP-III), Award DE-NA0003966.

This article has been authored by an employee of National Technology & Engineering Solutions of Sandia,LLC under Contract

No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes.The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan.

REFERENCES

[1] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1995. 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* (Santa Barbara, California, USA) *(SPAA '95)*. Association for Computing Machinery, New York, NY, USA, 95–105.

[2] Nicholas Bacon. 2023. GPU Datatype Enhanced Netgauge. https://github.com/CUP-ECS/datatypes-logGP

[3] Amanda Bienz, Luke N. Olson, William D. Gropp, and Shelby Lockhart. 2021. Modeling Data Movement Performance on Heterogeneous Architectures. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7.

[4] Dan Bonachea and Paul H Hargrove. 2019. GASNet-EX: A high-performance, portable communication library for exascale. In *Languages and Compilers for Parallel Computing: 31st International Workshop, Salt Lake City, UT, USA, October 9–11, 2018, Revised Selected Papers 31*. Springer, 138–158.

[5] Michael Boyer, Jiayuan Meng, and Kalyan Kumaran. 2013. Improving GPU performance prediction with data transfer modeling. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 1097–1106.

[6] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. 1993. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 1–12.

[7] Keira Haskins, Patrick Bridges, Kurt Ferreira, and Scott Levy. 2021. *A Benchmark to Understand Communication Performance in Hybrid MPI and GPU Applications.* Technical Report. Sandia National Laboratory, Albuquerque, NM.

[8] Torsten Hoefler, Torsten Mehlan, Andrew Lumsdaine, and Wolfgang Rehm. 2007. Netgauge: A Network Performance Measurement Framework. In *Proceedings of High Performance Computing and Communications, HPCC'07* (Houston, USA), Vol. 4782. Springer, 659–671.

[9] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. 2010. LogGOPSim: simulating large-scale applications in the LogGOPS model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. 597–604.

[10] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. 2001. LogGPS: a parallel computational model for synchronization analysis. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*. 133–142.

[11] Argonne National Laboratory. 2020. Yaksa : High-performance Noncontiguous Data Management. https://www.yaksa.org/.

[12] Lawrence Berkeley National Laboratory. 2023. GASNet-EX API Description. https://gasnet.lbl.gov/docs/GASNet-EX.txt

[13] Message Passing Interface Forum. 2021. MPI: A Message-Passing Interface Standard Version 4.0. https://www.mpi-forum.org/docs/

[14] Csaba Andras Moritz. 1998. Cost Modeling and Analysis: Towards Optimal Resource Utilization in Parallel Computer Systems. *Ph. D. Thesis, Royal Institute of Technology* (1998).

[15] NVIDIA. 2022. Faster memory transfers between CPU and GPU with GDRCopy. https://developer.nvidia.com/gdrcopy

[16] OpenUCX. 2023. Data type routines. https://openucx.readthedocs.io/en/master/api.html#data-type-routines

[17] Dhabaleswar K Panda, Karen Tomko, Karl Schulz, and Amitava Majumdar. 2013. The MVAPICH project: Evolution and sustainability of an open source production quality MPI library for HPC. In *Workshop on Sustainable Software for Science: Practice and Experiences, held in conjunction with Int'l Conference on Supercomputing (WSSPE)*.

[18] Carl Pearson, Kun Wu, I-Hsin Chung, Jinjun Xiong, and Wen-Mei Hwu. 2021. TEMPI: An interposed MPI library with a canonical representation of CUDA-aware datatypes. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*. 95–106.

[19] Rong Shi, Xiaoyi Lu, Sreeram Potluri, Khaled Hamidouche, Jie Zhang, and Dhabaleswar K Panda. 2014. Hand: A hybrid approach to accelerate non-contiguous data movement using MPI datatypes on GPU clusters. In *2014 43rd International Conference on Parallel Processing*. IEEE, 221–230.

[20] Xian-He Sun et al. 2003. Improving the performance of MPI derived datatypes by optimizing memory-access cost. In *2003 Proceedings IEEE International Conference on Cluster Computing*. IEEE, 412–419.

[21] Kaushik Kandadi Suresh, Kawthar Shafie Khorassani, Chen Chun Chen, Bharath Ramesh, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni, and Dhabaleswar K Panda. 2022. Network Assisted Non-Contiguous Transfers for GPU-Aware MPI Libraries. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 13–20.

[22] Ben Van Werkhoven, Jason Maassen, Frank J Seinstra, and Henri E Bal. 2014. Performance models for CPU-GPU data transfers. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 11–20.

[23] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Xiangyong Ouyang, Sayantan Sur, and Dhabaleswar K Panda. 2011. Optimized non-contiguous MPI datatype communication for GPU clusters: Design, implementation and evaluation with MVAPICH2. In *2011 IEEE International Conference on Cluster Computing*. IEEE, 308–316.