Predicting GPUDirect Benefits for HPC Workloads

Harsh Khetawat*, Nikhil Jain‡, Abhinav Bhatele†, Frank Mueller*

*Department of Computer Science, North Carolina State University
‡NVIDIA, Inc.
†Department of Computer Science, University of Maryland, College Park
Email: hkhetaw@ncsu.edu, nikhijain@nvidia.com, bhatele@cs.umd.edu, fmuelle@ncsu.edu

***Abstract*—Graphics processing units (GPUs) are becoming increasingly popular in modern HPC systems. Hardware for data movement to and from GPUs such as NVLink and GPUDirect has reduced latencies, increased throughput, and eliminated redundant copies. In this work, we use discrete event simulations to explore the impact of different communication paradigms on the messaging performance of scientific applications running on multi-GPU nodes. First, we extend an existing simulation framework to model data movement on GPU-based clusters. Second, we implement support for the simulation of communication paradigms such as GPUDirect with point-to-point messages and collectives. Finally, we study the impact of different parameters on communication performance such as the number of GPUs per node and GPUDirect. We validate the framework and then simulate traces from GPU-enabled applications on a fat-tree based cluster. Simulation results uncover strengths but also weaknesses of GPUDirect depending on the application and their usage of communication primitives.**

***Index Terms*—performance prediction, simulator, GPGPU, heterogeneous architectures**

I. Introduction

The use of general-purpose graphics processing units (GPGPUs) in modern high performance computing (HPC) systems is becoming increasingly popular, with seven of the ten fastest machines on the Top500 list using GPUs to speed up computation in November 2022 [1]. The continued improvement in GPU communication technologies has made GPUs first-class computation devices with the ability to directly exchange messages with one another, both within and across compute nodes. Hardware for data movement to and from GPUs such as NVLink and enhancements in system software such as GPUDirect has reduced latencies, increased throughput, and eliminated redundant copies during data movement between different system components.

While GPUs on HPC systems have dramatically increased a node's computing capabilities, the interconnect bandwidth per flop/s has not increased at the same rate even on high-end systems such as Summit and Sierra. The objective of this work is to explore the impact of using a large HPC system with multi-GPU nodes and modern GPU communication paradigms such as NVLink and GPUDirect on the messaging performance of scientific applications. Moving data directly between GPUs using technologies such as GPUDirect requires modification to the application code. To assess the performance impact of making such changes before the developer makes extensive modifications, we use parallel discrete event simulations (PDES) to study such what-if scenarios.

Most current network simulators treat the compute node as a black box and only model network communication to and from the node. In order to simulate direct communication between GPUs, GPUs need to be treated as first class objects in the network simulation. We use the TraceR-CODES [2] simulation framework that replays MPI execution traces using PDES to predict the communication performance of parallel codes. Explicitly simulating communication between the GPU and the network requires modifications to the traces used for simulation and also to the network simulation framework. We extend the Score-P OTF2 library [3] to annotate MPI calls with locations of the MPI send/receive buffers, which allows us to identify whether the buffers are in GPU memory or host memory. This allows us to predict the performance of the code if the buffers were read from GPU memory instead of host memory for instance.

We also extend TraceR-CODES to explicitly model GPUs, and communication between GPUs or between the host and GPUs with both point-to-point and collective MPI communication. We add functionality to replay any MPI messages that send data from CPU buffers as though they were sending data from GPU memory instead, using GPUDirect communication. We utilize these what-if capabilities to evaluate how application performance differs when using different communication paradigms, and identify the main factors contributing to GPUDirect benefits today. We also identify communication patterns largely unaffected by GPUDirect today, which can help in procurement, application tuning, and future enhancements of GPUDirect. Several scientific applications are yet to take advantage of GPUDirect. For those, our results provide guidance in determining the potential impact of GPUDirect before investing time into code refactoring.

II. Background

We provide a brief overview of modern GPU-based architectures and the TraceR-CODES simulation framework used for performance predictions in this work.

a) GPU-based Node Architectures: Over the last decade, GPGPUs have been used to drive scientific computations and have led to tremendous improvements in performance and energy efficiency, particularly for numerical kernels [4]. The development of frameworks and languages, such as

NVIDIA's Compute Unified Device Architecture (CUDA) [5] and the Open Compute Language (OpenCL) [6], have further contributed to establishing GPUs as first-class computational devices in HPC systems. Recently installed supercomputers such as Sierra and Summit rely predominantly on GPGPUs for their peak flop/s. The logical design of a node on the Sierra system is shown in Figure 1, featuring four high-end GPUs connected to two Power 9 CPUs via NVLink. The CPUs have access to non-volatile memory (NVMe) serving as burst buffers. Summit has six GPUs per node in a similar design.

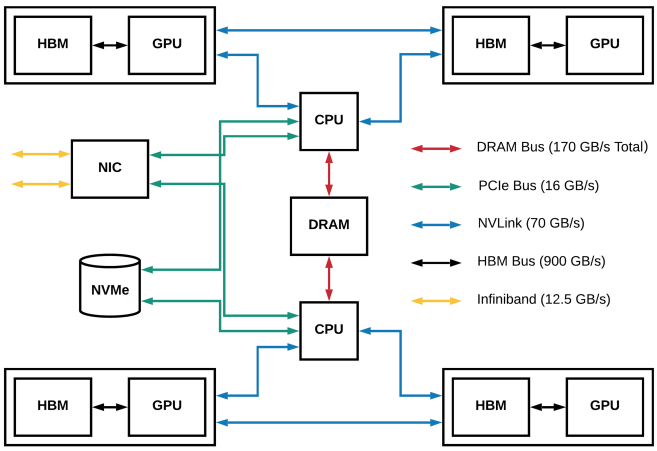


Fig. 1. Organization of a compute node on the Sierra supercomputer at Lawrence Livermore National Lab (LLNL).

The trend of delegating more and more computation to GPUs has spurred the development of modern interconnects such as NVLink [7] and communication methods such as GPUDirect [8]. While previous generations of GPUs were connected over slower Peripheral Component Interconnect Express (PCI-E) interfaces (green lines in Figure 1), newer GPUs with NVLink (depicted using blue lines) significantly increase throughput between GPUs, and GPUs and CPUs on a node. Prior to GPUDirect, data subject to communication between GPUs on the same node or between GPUs on different nodes required multiple copies, i.e., from GPU high-bandwidth memory (HBM) via NVLink and over the memory bus (red) to the host DRAM on the sender. We refer to this mode as **HostCopy** throughout the paper. GPUDirect avoids these extra copies by enabling data to be sent directly to the NIC from the GPU device's DRAM via NVLink and PCI-E. The GPUDirect transfers are still initiated by the CPU but the memory copies between the CPU DRAM and GPU HBM can be avoided.

b) Parallel Discrete Event Simulation with TraceR-CODES: Parallel Discrete Event Simulation (PDES) uses a sequence of events to model a system with each event resulting in a change in the state of the system.

The TraceR-CODES (https://github.com/hpcgroup/TraceR) simulation framework can be used to predict the performance of production applications by performing packet-level simulations of communication in parallel workloads running in HPC environments. CODES provides models for several network topologies such as fat-tree, dragonfly, torus etc. Rensselaer's Optimistic Simulation System (ROSS) is the discrete-event simulation engine used by TraceR-CODES. ROSS models each component in the system (MPI processes, switches, etc.) as a Logical Process (LP) that communicates with other LPs using time-stamped messages. We go into detail about some of the components of TraceR.

CPU LPs: Each CPU LP in TraceR acts as an MPI rank for an application. A list of timestamped MPI operations, usually read from a trace file, is associated with the CPU LP. These LPs communicate with each other through the network model using events during simulation. They also model compute loops by waiting for the appropriate time read from the trace.

MPI Model: The MPI model in TraceR is responsible for modeling the MPI layer of an application. It dictates the protocol to utilize for point-to-point MPI messages (explained in Section III), the algorithms for collective communication, and the expected behavior for synchronous and asynchronous sends and receives.

OTF2 Traces and Reader: Simulations using TraceR-CODES require capturing MPI execution traces for each parallel application. In order to collect these traces, we use the Score-P library that generates traces in the OTF2 format. The OTF2 Reader in TraceR takes the application trace file as input and associates operations in the trace file to each of the MPI ranks (CPU LPs). For each MPI operation it records the necessary information in order to replay the operation during simulation: type of MPI operation, the destination/source for MPI Send/Receive messages, the root for collective communication, the size of the payload, etc.

III. Simulating Communication & GPUs

In the current implementation of TraceR, GPUs and their communication are not modeled explicitly. For this study, to be able to model the impact of NVLink and GPUDirect technologies, we had to add the notion of independent GPU LPs to the simulation framework. Since TraceR uses OTF2 traces to simulate applications, we also need to extend Score-P and OTF2 traces to record additional information that can distinguish GPUDirect MPI calls from regular HostCopy MPI calls. Below, we describe the changes to Score-P and the OTF2 traces, the model of the GPU LP and the associated operations and communications.

a) Score-P and OTF2: We extend the Score-P MPI backend to query the CUDA runtime for the location of the buffer pointer passed to the MPI call and then annotate the OTF2 trace file with information about the location of this buffer. We annotate each MPI event with a flag denoting whether the buffer was in CPU memory or GPU memory. On the simulator end, we enhance the OTF2 reader in TraceR. During initialization of the simulation, the tasks for each server LP are read from the trace file, flagging each operation as either MPI call or a GPUDirect call using the annotations.

Further, we enhance TraceR-CODES to support replay of regular MPI HostCopy calls as GPUDirect calls, which enables simulation of GPUDirect using traces collected for a non-GPUDirect version of the code.

b) The GPU Logical Process: We augment the server LP, which represents an MPI process in our simulation, with an additional GPU LP. There is a one-to-one mapping between a server LP and a GPU LP, which gives each MPI process exclusive control over its GPU. This means that in the current implementation, each MPI rank can only control one GPU. So, if there are n GPUs on a node, we simulate n MPI processes on it. As the simulation proceeds, server LPs are assigned events by the simulator from the execution traces during simulation initialization. The GPU LPs wait to receive events from other LPs to execute their corresponding tasks.

During the simulation initialization process, TraceR reads the respective tasks that need to be assigned to each process or server LP. The GPU LPs are not assigned tasks during initialization but receive events from either their associated server LP or other GPU LPs, both inter-node and intra-node. The time between these communication triggered events is used to model either an idle GPU or a GPU engaging in active computation. The parameters added to TraceR-CODES to support GPU LPs and modeling of their associated communication are shown in Table I.

TABLE I
NEW TRACER-CODES PARAMETERS FOR GPUS

Parameter	Description	Value
gpu_copy_enabled	Enable HostCopy simulation	1 (for HostCopy) 0 (for GPUDirect)
gpudirect_enabled	Enable GPUDirect simulation of regular MPI calls	1 (for GPUDirect) 0 (for HostCopy)
gpu_copy_delay	Latency for cudaMemcpy	20000ns
gpu_copy_per_byte	Copy cost per byte for cudaMemcpy	0.07ns
gpudirect_delay	Cost of GPUDirect memory pinning	4000ns

c) Simulating HostCopy: The simulation of HostCopy communication uses the packet-level modeling already implemented in TraceR albeit with the added time to copy the message buffer from the host DRAM to the GPU, or vice versa. An MPI operation is recorded as a HostCopy operation when the OTF2 reader in TraceR determines that the location of the message buffer is the CPU. We add parameters to denote the latency and per-byte cost of cudaMemcpy calls (Table I). In the simulation, this delay is added to the messaging time to facilitate comparison with GPUDirect performance. Apart from this delay, the simulation proceeds as if simulating regular MPI communication.

d) Simulating GPUDirect: MPI implementations use two different protocols for send/receive operations based on the size of the message: eager and rendezvous. The eager protocol allows a sender to send the message to the receiver without having to receive an acknowledgment from the receiver. The eager protocol is used for smaller message sizes (decided by the EAGER_LIMIT parameter in MPI). The rendezvous protocol is used for larger message sizes and requires the sender to receive a notification from the receiver that the corresponding receive has been posted and a buffer is available for the incoming message.

We extend TraceR to add various GPU LP-related events to enable simulating GPUDirect communication. These events support coordination between the GPU LPs and their associated CPU LPs:

- GPU_SEND: This event is used by the sender server LP to inform its associated GPU LP that it should initiate a GPUDirect send.
- GPU_RECV: The GPU_RECV event is used by the sender GPU LP to inform the receiver GPU LP of an incoming GPUDirect message. This event also carries the MPI payload.
- GPU_SEND_DONE: This event is used by the sender GPU LP to inform the sender CPU LP that GPUDirect communication has been completed.
- GPU_RECV_DONE: The receiver GPU LP uses this event to inform the receiver server LP that a GPUDirect message has been received.

A timeline of the simulation of the GPUDirect rendezvous protocol as implemented in TraceR-CODES is shown in Figure 2. The sender server LP sends a Request to Send (RTS) to the receiver server LP. Once a corresponding receive has been posted, the receiver informs the sender with a Clear to Send (CTS) message. The sender server LP then informs its associated GPU LP to initiate a GPUDirect send to the receiving GPU LP. Once the message has been transmitted to the receiver GPU LP, the sender and receiver GPU LPs inform their respective server LPs using GPU_RECV_DONE and GPU_SEND_DONE events, respectively.

In addition to simulating point-to-point GPUDirect communication, we also implement the modeling of GPUDirect for some of the most popular MPI collectives encountered in HPC applications, namely MPI_Broadcast, MPI_Reduce, MPI_Allreduce, MPI_Allgather and MPI_Alltoall. The implementation of these collectives has been facilitated by using control messages between the CPU and GPU as well as GPUDirect communication between the GPUs. We use tree-based implementations for modeling GPUDirect collective communication. Our algorithms are based on the implementation of collectives in TraceR for standard MPI collectives, which dominate the benchmarks studied.

We add a parameter to represent the cost of pinning the GPU memory address with the GPU DMA engine in preparation of sending the message (gpudirect_delay in Table I). GPU memory is pinned in BAR using the CUDA kernel driver [9]. This introduces an overhead especially for small messages. We simulate NVLink communication using the intra-node bandwidth and node copy queues (NCQ) parameters in TraceR-CODES. The former specifies the bandwidth of the intra-node

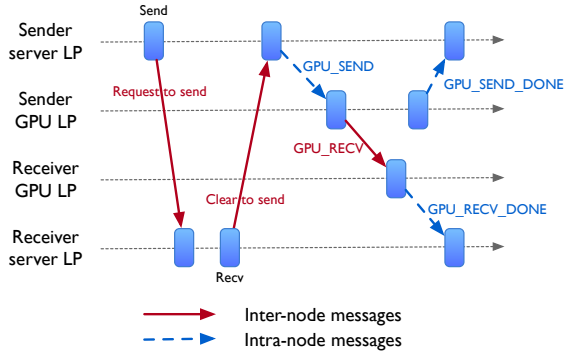


Fig. 2. GPUDirect messaging in TraceR-CODES.

bus to be used in the simulation, in this case NVLink. The latter indicates the number of queues to be used for the intra-node communication.

e) Validation: We validate our simulation models by timing an MPI_Send/Recv between a pair of nodes on Lassen for both the HostCopy and GPUDirect scenarios for a range of message sizes. We time the MPI calls for 100 iterations and validate against the average time of the latter 90 iterations (ignoring the first 10 due to caching effects). We then simulate the traces of the Send/Recv pairs for each message size with both our GPUDirect and HostCopy models. Figure 3 depicts the results of the validation experiments. Both plots use log scales on their y-axes denoting elapsed time and plot the size of an MPI message on the x-axis. We can see from the validation results that the time for Send/Recv pairs increases linearly with the size of the message. For both GPUDirect and HostCopy, we observe that our models are quite accurate with the average error for GPUDirect is less than 16% for messages that are of size 32KB or greater and for HostCopy is less than 13%. While we see a larger error for GPUDirect for small messages, they do not have a significant impact on our simulations since SW4lite and Minisweep (see Sect. IV) exclusively use large messages (680KB to 11.5MB for SW4lite and 4MB to 8MB for Minisweep). Messages of size less than 32KB account for only about 4% and 0.6% of the messaging payload for AmgX and Lulesh, respectively.

IV. Experimental Design

We first describe the system and network for simulation studies and the four applications subject to trace generation.

a) Simulated GPU-based System: We set up a GPU cluster with 810 nodes and 45 leaf-level switches connected by a 3-level tapered fat-tree network EDR Infiniband [10]. This is resembling the Lassen supercomputer [11], with a 1.5:1 tapering on the fat-tree. While we collect traces on the Lassen supercomputer as well as model the simulated system based on Lassen, we run our simulations on the Quartz system at LLNL [12]. Quartz is a ≈3000 node machine with Intel Xeon CPUs and a Cornelis Omni-Path network.

Node Configurations: We keep the number of nodes in the system constant but vary the number of GPUs per node – 1, 2, 4 and 6 GPUs/node. Each MPI process gets exclusive use of its GPU. Therefore, we simulate 1, 2, 4 and 6 processes/node configurations for the different numbers of GPUs/node. This allows us to compare the performance on different GPU-based systems such as Sierra (4 GPUs/node), Summit (6 GPUs/node) and Piz Daint (1 GPU/node). We set the intra bandwidth parameter to the bandwidth of NVLink in Lassen and Sierra, which is 75 GB/s. We also set the GPUDirect delay, CUDA host copy delay and CUDA copy per byte parameters based on experiments on the Lassen supercomputer.

Simulation Considerations: We base our simulations on the following design decisions:

- Communication protocol: While HostCopy simulations use both the eager and rendezvous protocols based on message size, GPUDirect simulations use only the rendezvous protocol. This reflects the Open MPI implementation, which does not use the eager protocol under GPUDirect [13]. Our experiments focus on bulk MPI messaging since the vast majority of HPC applications use this paradigm for communication.
- Exclusive GPU use: Each MPI process in our simulation has exclusive access to a GPU. While multiple processes can share a GPU, in practice, most HPC applications exclusively allocate a GPU to a process.
- Process-to-node mapping: We use a linear mapping to allocate processes to nodes, where processes are assigned to nodes in MPI rank order.

b) Applications: We collect traces from four proxy applications for the simulation experiments. All applications are run in weak scaling mode, and we only instrument the main computation loop in each application to generate traces. For each application, we collect traces on Lassen using 4 MPI processes per node with an exclusive GPU. We chose applications specifically to cover a wide variety of patterns with point-to-point and collective messages as well as small and large message sizes, all of them using MPI and GPUs (with CUDA) for computation. We describe the proxy applications and their input parameters below.

AmgX [14] is NVIDIA's version of a Distributed Algebraic Multigrid Solver Library. It is a GPU-accelerated solver for sparse linear systems. For our trace collection, we use the 7-point Poisson example application in AmgX. We collect traces for 32, 64, 128 and 256 processes while weakly scaling the application to maintain the same problem size per process.

LULESH [15] is a shock hydrodynamics proxy application developed at LLNL. The number of MPI processes it can use is constrained by n^3, where $n \in N$. Given this constraint we collect traces for 27, 64, 125 and 216 processes with weak scaling. We collect traces for a structured mesh size of 144^3 running for 100 iterations repeatedly for 10 times.

Minisweep [16] is a deterministic S^n radiation transport miniapp developed at Oak Ridge National Laboratory. Similar to AmgX, we perform weak scaling of Minisweep on Lassen and collect traces for execution sizes of 32, 64, 128 and 256 processes over 10 iterations.

SW4lite is a proxy application for SW4 [17], a code for

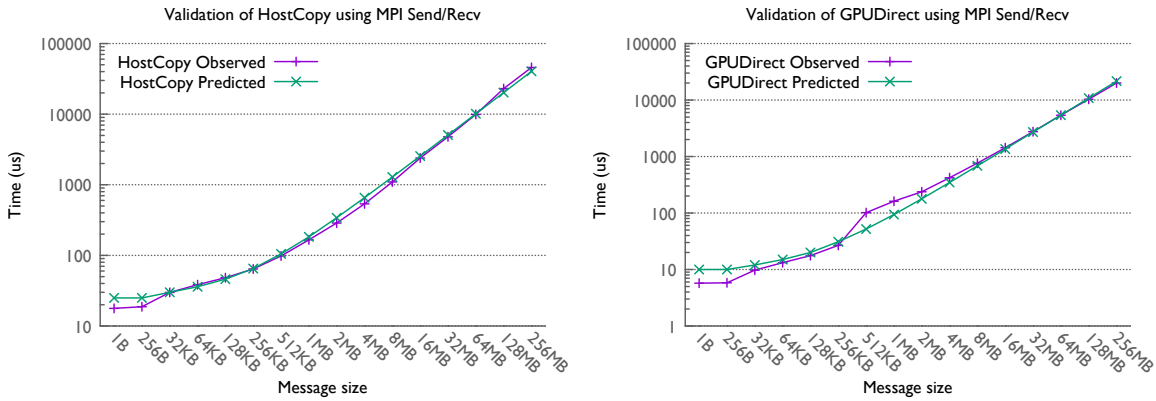


Fig. 3. Validation of HostCopy and GPUDirect simulations using TraceR-CODES.

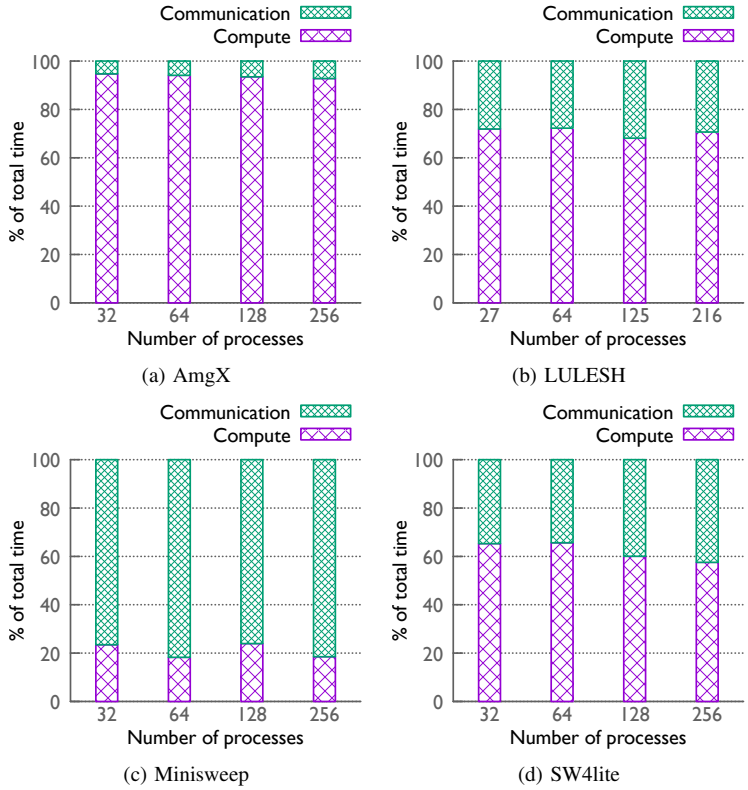


Fig. 4. Time spent in computation vs. communication (MPI routines) over different number of processes.

solving seismic wave equations in Cartesian coordinates. We capture traces for SW4lite for 100 iterations and weakly scale the application for 32, 64, 128 and 256 processes.

c) Communication Behavior: We use Vampir [18] to analyze the collected application traces to ascertain the amount of time each application spends in computation vs. the time it spends in MPI routines. This analysis provides a preliminary estimate of how much of the application runtime can be affected by either GPUDirect, or the different node configurations. We aggregate the time spent in all MPI communication functions to get the total MPI time of the application. Figure 4 shows the results from this analysis.

AmgX (Figure 4(a)) spends a small amount of time in MPI routines ranging from 5.3% for 32 ranks to 7.2% for 256 ranks indicating that even large improvements in MPI time will not significantly improve the overall performance of the application. Each MPI rank communicates with 4 other ranks and exchanges the same number of messages.

For Minisweep (Figure 4(c)), the overall execution time is dominated by the MPI routines. MPI time ranges from 76.0% for 128 ranks to 81.7% for 64 ranks. Minisweep can greatly benefit from improvements in communication time as each rank communicates with 2-26 other ranks.

For applications such as LULESH (Figure 4(b)) and

SW4lite (Figure 4(d)) we observe that a moderate amount of time is spent in MPI routines. LULESH spends between 28.1% (for 64 ranks) to 31.9% (for 125 ranks) of its execution time inside MPI routines. Similarly, SW4lite spends between 34.3% (for 64 ranks) and 42.5% (for 256 ranks) in MPI routines. The benefit accorded to the overall execution by reducing messaging time for these applications will largely depend on the quantum of improvement in messaging time. Both these applications have each MPI rank communicating with 2-4 other ranks with the same number of messages. Slight variations in communication/computation ratio are caused by variations in the state of the system when the traces were collected and due to the communication characteristics of the application for different rank sizes. This static analysis can be used to determine if it is worthwhile to invest in improving the MPI performance of the application either using GPUDirect or by changing the number of ranks on each node.

V. Results

This section presents the results obtained from simulating executions via the traces for the node configurations described previously. For each application, we also simulate the effect of implementing GPUDirect. We assess the performance of GPUDirect and HostCopy by comparing the time each application spends in MPI routines (for GPUDirect) and MPI routines + CUDA memcpy calls (for HostCopy) for both the weak scaling and the strong scaling scenarios. Finally, we discuss the impact of the node configuration on each application by varying the number of ranks (GPUs) per node.

Performance improvements of GPUDirect benefit inter-node communication in large due to omission of cudaMemcpy calls since any MPI communication uses the same interconnect irrespective of HostCopy or GPUDirect.

a) Weak Scaling Scenario: Figure 5 depicts speedups in communication time for GPUDirect over HostCopy per application under weak scaling.

Observation 1: *Some applications benefit more than others from GPUDirect communication. Lulesh has only up to a 3% improvement in communication performance while SW4lite shows up to a 75% improvement in communication performance with GPUDirect.*

There is a substantial improvement in communication time for SW4lite (Figure 5d) when using GPUDirect in lieu of HostCopy, with a 15.8%-74.4% speedup of the former over the latter. For AmgX (Figure 5a), we observe that GPUDirect is faster by 16.2%-17.6% compared to HostCopy. For Minisweep (Figure 5c), this difference ranges from 0.1% for 128 ranks to 7.9% for 64 ranks, and for LULESH (Figure 5b) the difference is small (less than 3%).

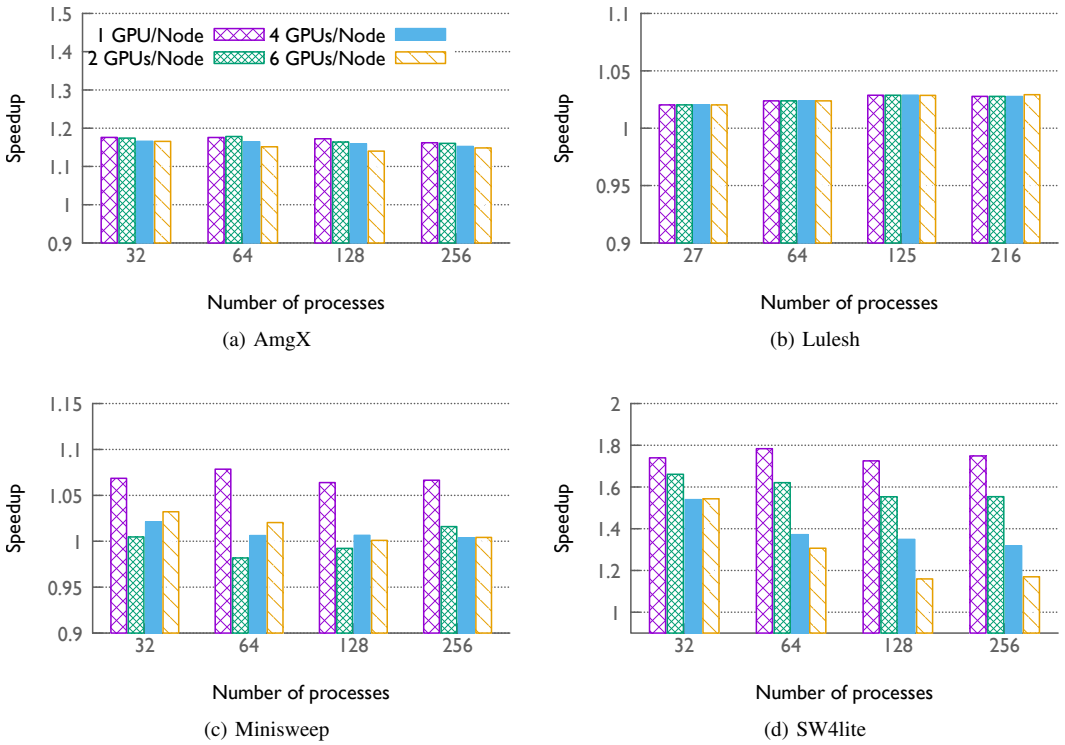
Observation 2: *The impact of reduced communication on overall application runtime under GPUDirect varies significantly depending on communication to computation ratio. Assessing both this ratio as well as the improvement due to GPUDirect helps in determining viable candidates for a GPUDirect implementation. Our experiments show that AmgX with only 5% communication is not a viable candidate while SW4lite with 40% communication is ideal.*

SW4lite spends up to about 42% of its execution time in MPI calls with up to a 74% improvement in communication performance when GPUDirect is used. Such applications can be expected to exhibit significantly improved execution times with GPUDirect. Although there seems potential, later results indicate hidden costs (e.g., due to DMA) that outweigh the benefits, i.e., message size alone is not a sufficient condition. Minisweep (Figure 4c) spends up to about 81% of its execution time in communication. For such applications, the modest improvement under GPUDirect (up to 7.9%) can have a significant impact on the overall execution time of the application. In contrast, AmgX spends only about 5.3%-7.2% of its time in communication (Figure 4a). Even though this results in up to a 18% improvement in GPUDirect performance over HostCopy, the small time AmgX spends in communication limits the overall benefit in execution time under GPUDirect. Finally, LULESH spends a non-trivial amount of time in MPI (up to 32%), but small improvements in communication performance makes it unsuitable for retrofitting with GPUDirect.

Observation 3: *The improvement in GPUDirect performance for an application depends on the characteristics of the MPI messages. The size of MPI messages as well as collective vs. point-to-point communication characteristics have an impact on GPUDirect performance.*

The benefit of using GPUDirect over HostCopy is most evident when MPI message sizes are large. In particular, Minisweep utilizes predominantly large MPI messages of either 4MB or 8MB. Similarly, SW4lite uses message sizes of about 64KB (50% of MPI messages) or about 12MB (the remaining 50% of messages). For Minisweep we see that GPUDirect performance can be worse than HostCopy in certain cases (e.g., Figure 5c for 2 GPUs at 64 and 128 processes) even though Minisweep utilizes large message sizes. This is because (as is evident from Fig 4c) Minisweep spends a vast majority of its time in communication and is bound by the intra and inter node bandwidths. Furthermore, using GPUDirect adds additional overhead of pinning GPU memory addresses to the GPU DMA engine, sometimes resulting in inferior performance to HostCopy. AmgX has a large distribution of message sizes: About 8% of messages are greater than 500KB, 17% have sizes of 100KB-500KB, and 71% range from 1KB-100KB. Experiments further indicate that AmgX benefits from GPUDirect collectives, dominated by *MPI_Allreduce*. For smaller message sizes, the performance improvement is modest. The size of MPI messages ranges between 24B to 7KB for LULESH accounting for more than 44% of MPI messages, and an equal number (44%) of MPI messages are in the 7-512KB range. Only about 11% of MPI messages are around 1MB. This results in only a negligible improvement in GPUDirect performance. While other factors also play a role, results show that message size is a significant factor for benefits from GPUDirect.

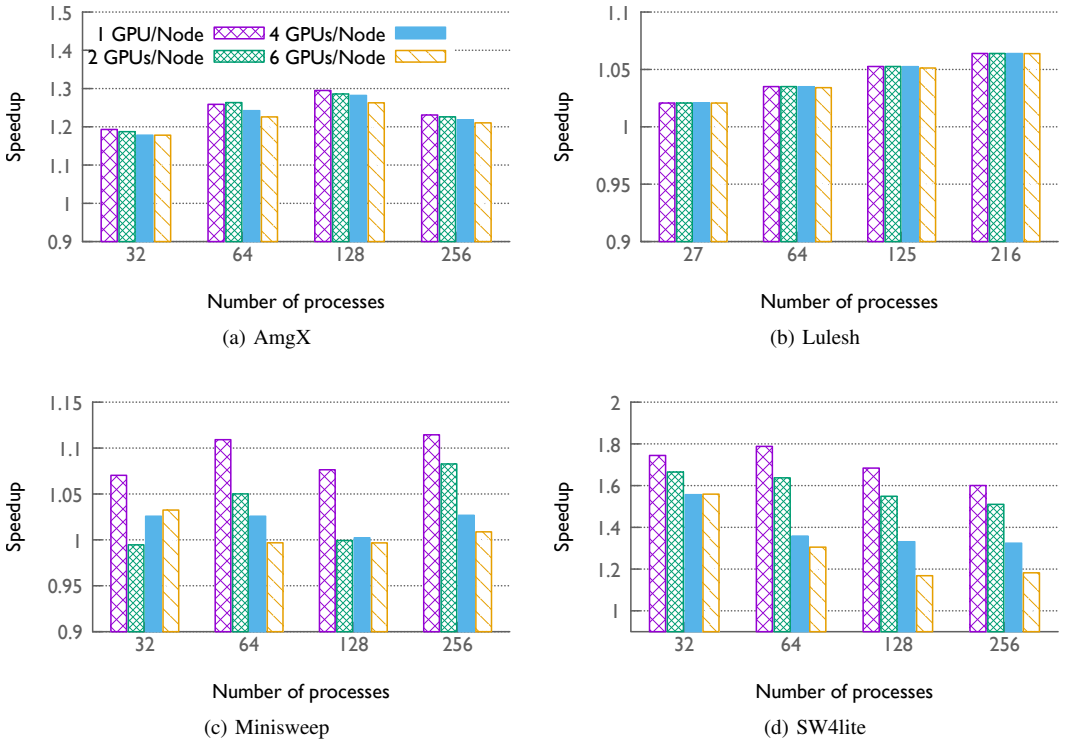
b) Strong Scaling Scenario: We conducted experiments using strong scaling traces collected for each application and



(a) AmgX (b) Lulesh

(c) Minisweep (d) SW4lite

Fig. 5. Weak Scaling: Predicted speedup in comm. time for GPUDirect over HostCopy (baseline), see text for std-dev.



(a) AmgX (b) Lulesh

(c) Minisweep (d) SW4lite

Fig. 6. Strong Scaling: Predicted speedup in comm. time for GPUDirect over HostCopy (baseline), see text for std-dev.

rank size. We also conducted experiments while strong scaling the applications from 32-256 ranks (Figure 6). We see the speedup in communication time achieved with GPUDirect (instead of HostCopy) for each configuration. Results indicate improvements of GPUDirect similar to that in the weak scaling experiments across all applications. For AmgX, GPUDirect results in improvements of 18%-29% over HostCopy, and for SW4lite improvements range from 18%-77%.

Marginal improvements exist under GPUDirect for Minisweep with only a 11% improvement over HostCopy in the best case, just as for weak scaling. Experiments with LULESH also show that GPUDirect is only $approx$2%-6% faster than HostCopy during strong scaling.

c) Effect of Node Configuration: We investigated the effect of node configurations on the communication performance of applications. HPC centers procure machines with varying performance characteristics, the Sierra and Lassen supercomputers at LLNL have four GPUs per node while Summit at ORNL has 6 GPUs per node. Piz Daint at CSCS has one GPU per node. Increasing the number of GPUs per node increases the peak performance of the machine while keeping the number of nodes constant, thus reducing the network and server costs. GPUs on different nodes communicate over the inter-node network, which is Infiniband for Lassen, Sierra and Summit. GPUs on the same node communicate over the intra-node networks, e.g., NVLink.

While Figure 5 shows speedup numbers for communication, we also need to consider absolute time to understand the impact of the number of GPUs. Figure 7 depicts the execution time of Minisweep across various node configurations while weakly scaling. The overall execution time can be split into computation and communication time. Because we are weakly scaling, the computation time remains about the same across different rank sizes. Increasing the number of GPUs (ranks) on a single node reduces the number of nodes required for an execution with the same number of ranks. Adding more GPUs opens up more opportunities for intra-node communication (over NVLink) since there are more ranks on the same node. In our simulations, we allocate each rank to its own GPU.

Observation 4: *Adding more GPUs (and subsequently more ranks) decreases the performance of applications. Furthermore, larger application runs are impacted more than smaller ones. The communication performance of Lulesh is not impacted by adding more GPUs to the node while Minisweep experiences a communication degradation with more GPUs, namely up to 242% for 6 GPUs/node.*

The experiments show that while the high-speed intra-node network is used, the overall communication runtime of an application increases when more ranks are added to the same node. SW4lite and Minisweep see the largest performance penalty when more ranks are allocated to the same node in terms of speedup (Figure 5) and absolute time (Figure 7). For SW4lite, the total communication time increases by 23.9%-39.8% for 32 ranks and up to 118%-227% for the 256 ranks when increasing the number of ranks from one per node to 6 per node whereas the overall execution time increases by 9.1% for 32 ranks to 50.4% for 256 ranks. This is because SW4lite spends only about 40% of its time on communication. Similarly, Minisweep suffers a performance penalty of 47.6%-52.6% for 32 ranks and up to 222%-242% for 256 ranks when going from one rank per node to six ranks per node. This also has a severe effect on the overall execution time since Minisweep spends a vast majority of its execution time on communication. The execution time increases by 36.9% for 32 ranks and up to 169.7% for 256 ranks. For AmgX, the communication performance decrease is smaller, ranging from 2.4%-3.3% for 32 ranks and from 6.2%-7.4% for 256 ranks when comparing one rank per node to six ranks per node. Finally, the communication performance for LULESH remains roughly the same regardless of the number of ranks allocated to each node. For 27, 64 and 125 ranks, the communication performance remains the same, and for 216 ranks, an increase in communication time of only about a 1% is observed when modulating the number of ranks per node from one to six. The standard deviation in the communication time for different ranks is quite low: 0.34-0.42 for AmgX, 0.09-0.11 for Lulesh, 0.01-0.02 for Minisweep, and 0.07-0.12 for SW4lite. This indicates that the traces do not experience high jitter; otherwise, the standard deviation would be significantly higher.

The rise in communication cost while increasing the number of ranks per node is due to higher contention for the shared NIC on the node. As the number of ranks per node increases, more ranks are engaging in MPI communication with ranks outside the node. This causes contention on the node-local NIC. While there is sufficient bandwidth available on the intra-node interconnect, the increased requirement for inter-node communication causes a bottleneck at the node boundary.

Under strong scaling, the communication performance of applications degrades as more GPUs are added to the node, just as for weak scaling before. For AmgX, communication time increases modestly by 3.2% for 32 ranks and up to 10% for 64 ranks while the increase in communication time for Lulesh is negligible ($< 1\%$). Yet, SW4lite and Minisweep have significantly reduced communication performance as more GPUs are added to the node, increasing communication by 39.5%-181% for SW4lite and by 55.7%-225.8% for Minisweep.

Observation 5: *Adding more compute resources to a node requires careful consideration of the trade-offs, the expected application workload and optimized rank-to-node allocation schemes. Adding more GPUs to the node requires improvements in network bandwidth if communication performance is to be maintained.*

HPC centers can increase the peak compute performance of the system by adding more GPUs to each node while keeping the server and networking costs in check. Consequently, operational expenditure can also be curtailed. This strategy is becoming common in HPC centers, i.e., they increase the number of GPUs on each node to four (Sierra at LLNL) or even six GPUs per node (Summit at ORNL). But interconnects have not kept pace with the increase in compute performance of modern CPUs and GPUs creating bottlenecks at the shared NIC. HPC centers should conduct

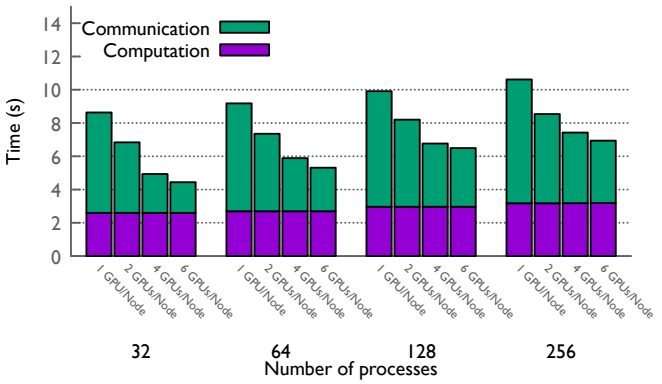


Fig. 7. Minisweep communication/computation times per node configurations while scaling the network bandwidth with number of GPUs/node.

studies of the trade-offs between cost and performance. We conducted simulation experiments with Minisweep by increasing the network bandwidth linearly with the number of GPUs on the node (shown in Figure 7). Our results show that the decrease in communication performance is avoided with interconnects having larger bandwidths. Here the intra-node NVLink interconnect starts to become the bottleneck. HPC centers need to evaluate the applications that dominate utilization on their system. Some applications (e.g., AmgX and LULESH) either show little or no performance penalty when more compute resources are added to the node. Others, like SW4lite and Minisweep, show significantly decreased performance. Furthermore, we observe across applications that choosing 1-2 GPUs per node has no significant effect on performance, but a further increase to four or six GPUs per node can severely degrade performance.

Finally, an application-conscious optimization of the allocation of ranks to nodes can increase the use of the intra-node interconnect if local communication dominates, which reduces the pressure on the shared NIC. This requires prior analysis of the MPI messaging characteristics for each application. The proportion of intra-node communication can be increased by allocating more communicating ranks to the same node, particularly for small groups of frequently communicating ranks with less communication between such groups.

VI. Related Work

Prior work has assessed the performance of different GPU interconnect topologies. Li et al. [19] evaluate the performance of interconnect topologies such as NVLink, PCIe, NV-SLI, NVSwitch and GPUDirect on 6 high-end servers and HPC platforms. They observe that intra-node multi-GPU communication is dependent on the choice of GPU combinations. Potluri et al. [20] study the performance of GPUDirect on PCIe-connected GPUs in a multi-node HPC environment. The authors show that using GPUDirect increases inter-GPU bandwidth significantly while decreasing communication latency.

Jiao et al. [21] characterize GPU applications based on performance and energy efficiency. They study applications with varying compute and memory intensity to conclude that performance and efficiency of GPUs in an HPC context depends on the computational patterns employed by the application. Pennycook et al. [22] utilize the NAS LU benchmark to compare existing HPC clusters and future large-scale systems. The authors use the LU performance model to extrapolate the performance of the benchmark to future large-scale distributed GPU clusters. Choi et al. [23] model the end-to-end performance of GPU HPC applications using PDES while Arafa et al. [24] focus on the prediction of computational performance of GPU applications. Chapuis et al. [25] use PDES to predict the GPU performance by using a model that is a combination of cycle-accurate GPU models and more coarse-grained analytical models. Their work focuses on the compute performance of individual GPUs and not on modern GPU communication paradigms or the resulting MPI performance. Groves et al. [26] develop a prediction model for NVSHMEM instead of our focus on GPUDirect. In contrast to all this prior work, ours contributes a PDES framework to comprehensively study the impact of multi-node GPU applications on HPC messaging performance *with modern GPU communication paradigms* using GPUDirect, which is novel.

VII. Conclusion

This work enhances the TraceR simulation framework with novel capabilities to treat GPUs as a first-class computation device, enabling us for the first time to model multi-GPU nodes without actually having to port application code to GPUDirect. It complements TraceR with support for GPU-aware MPI communication, both for point-to-point messages and collectives in order to accurately simulate HPC systems with multi-GPU nodes. It also extends the trace collection utility, Score-P, with a feature to annotate MPI operations so as to indicate the use of GPUDirect without a need to refactor application code for GPUDirect. The framework gains the ability to support what-if analysis of modern communication paradigms such as GPUDirect. The performance of GPUDirect, in a what-if scenario, is then compared against the traditional method of copying data through host memory. These novel capabilities also allow us to study the impact of the number of GPUs per node on application and network performance. Using this framework, the communication performance of four important HPC applications is evaluated. Results indicate that applications experience asymmetric benefits from using GPUDirect, i.e., some will benefit significantly while others do not. Given our what-if analysis, this allows applications programmers to selectively decide which applications to refactor for GPUDirect. Furthermore, HPC centers are facing an important tradeoff between maximized performance vs. minimized capital and operational expenditure, i.e., a choice between adding more GPUs to fewer nodes vs. deploying more nodes with fewer GPUs — a decision that also depends on the application mix, as our results show. Finally, the simulator can be combined with performance models like the roofline or analytical models to further improve prediction performance.

ACKNOWLEDGMENTS

This work supported in part by NSF awards CISE-2316201, CISE-2217020, a subcontract from LLNL and by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The work used the resources from the OLCF.

REFERENCES

[1] "Top500 hpc systems," https://www.top500.org/lists/top500/2022/11/, Nov. 2022.

[2] B. Acun, N. Jain, A. Bhatele, M. Mubarak, C. D. Carothers, and L. V. Kale, "Preliminary evaluation of a parallel trace replay tool for hpc network simulations," in *Workshop on Parallel and Distributed Agent-Based Simulations*, Aug. 2015.

[3] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. E. Nagel, and F. Wolf, "Open trace format 2: The next generation of scalable trace formats and support libraries." in *PARCO*, vol. 22, 2011, pp. 481–490.

[4] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[5] Nvidia, "Compute unified device architecture programming guide," *Nvidia*, 2007.

[6] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[7] D. Foley and J. Danskin, "Ultra-performance pascal gpu and nvlink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[8] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and P. S. Crozier, "The development of mellanox/nvidia gpudirect over infiniband—a new model for gpu to gpu communications," *Computer Science-Research and Development*, vol. 26, no. 3-4, pp. 267–273, 2011.

[9] "Gpudirect," https://docs.nvidia.com/cuda/gpudirect-rdma/index.html, 2023.

[10] E. A. León, I. Karlin, A. Bhatele, S. H. Langer, C. Chambreau, L. H. Howell, T. D'Hooge, and M. L. Leininger, "Characterizing parallel scientific applications on commodity clusters: An empirical study of a tapered fat-tree," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2016, pp. 909–920.

[11] LLNL, "Lassen," https://hpc.llnl.gov/hardware/platforms/lassen, Lawrence Livermore National Laboratory, 2023.

[12] ——, "Quartz," https://hpc.llnl.gov/hardware/platforms/Quartz, Lawrence Livermore National Laboratory, 2023.

[13] OpenMPI, "Openmpi: Running cuda-aware open mpi," https://www.open-mpi.org/faq/?category=runcuda, 2023.

[14] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh *et al.*, "Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods," *SIAM Journal on Scientific Computing*, vol. 37, no. 5, pp. S602–S626, 2015.

[15] I. Karlin, J. McGraw, J. Keasler, and B. Still, "Tuning the lulesh mini-app for current and future hardware," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2013.

[16] C. Baker, G. Davidson, T. M. Evans, S. Hamilton, J. Jarrell, and W. Joubert, "High performance radiation transport simulations: preparing for titan," in *SC'12: International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–10.

[17] N. Petersson and B. Sjogreen, "Sw4, version 2.0 [software], computational infrastructure for geodynamics, 2017, doi: 10.5281/zenodo. 1045297."

[18] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The vampir performance analysis toolset," in *Tools for high performance computing*. Springer, 2008, pp. 139–155.

[19] 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.

[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*. IEEE, 2013, pp. 80–89.

[21] Y. Jiao, H. Lin, P. Balaji, and W.-c. Feng, "Power and performance characterization of computational kernels on the gpu," in *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*. IEEE, 2010, pp. 221–228.

[22] S. J. Pennycook, S. D. Hammond, S. A. Jarvis, and G. R. Mudalige, "Performance analysis of a hybrid mpi/cuda implementation of the naslu benchmark," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 23–29, 2011.

[23] J. Choi, D. F. Richards, L. V. Kale, and A. Bhatele, "End-to-end performance modeling of distributed gpu applications," in *34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[24] Y. Arafa, A.-H. Badawy, A. ElWazir, A. Barai, A. Eker, G. Chennupati, N. Santhi, and S. Eidenbenz, "Hybrid, scalable, trace-driven performance modeling of gpgpus," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[25] G. Chapuis, S. Eidenbenz, and N. Santhi, "Gpu performance prediction through parallel discrete event simulation and common sense," in *9th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2016, pp. 204–211.

[26] T. Groves, B. Brock, Y. Chen, K. Z. Ibrahim, L. Oliker, N. J. Wright, S. Williams, and K. Yelick, "Performance trade-offs in gpu communication: A study of host and device-initiated approaches," in *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2020, pp. 126–137.