Accelerating Intra-Node GPU Communication: A Performance Model for Multi-Path Transfers

Amirhossein Sojoodi
Queens University
Kingston, Ontario, Canada
amir.sojoodi@queensu.ca

Mohammad Akbari
Queen's University
Kingston, Ontario, Canada
13mav1@queensu.ca

Hamed Sharifian
Queen's University
Kingston, Ontario, Canada
hamed.sharifian@queensu.ca

Ali Farazdaghi
Queen's University
Kingston, Ontario, Canada
m.farazdaghi@queensu.ca

Ryan E. Grant
Queen's University
Kingston, Ontario, Canada
ryan.grant@queensu.ca

Ahmad Afsahi
Queen's University
Kingston, Ontario, Canada
ahmad.afsahi@queensu.ca

Abstract

Optimizing GPU-to-GPU communication is a key challenge for improving performance in MPI-based HPC applications, especially when utilizing multiple communication paths. This paper presents a novel performance model for intra-node multi-path GPU communication within the MPI+UCX framework, aimed at determining the optimal configuration for distributing a single Point-to-Point (P2P) communication across multiple paths. By considering factors such as link bandwidth, pipeline overhead, and stream synchronization, the model identifies an efficient path distribution strategy, reducing communication overhead and increasing throughput. Through extensive experiments on various topologies, we demonstrate that our model accurately finds experimentally optimal configurations, achieving significant improvements in performance, with the average of less than 6% error in predicting the optimal configuration for very large messages.

CCS Concepts

• **Software and its engineering → Message passing**.

Keywords

Performance Model, Multi-Path Communication, Multi-GPU, MPI, UCX

ACM Reference Format:

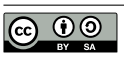
Amirhossein Sojoodi, Mohammad Akbari, Hamed Sharifian, Ali Farazdaghi, Ryan E. Grant, and Ahmad Afsahi. 2025. Accelerating Intra-Node GPU Communication: A Performance Model for Multi-Path Transfers. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3731599.3767392

1 Introduction

In modern High-Performance Computing (HPC) systems, Graphics Processing Unit (GPU) accelerators form the core of computational resources, and efficient intra-node GPU-to-GPU communication is critical for achieving high overall performance [9, 35, 41]. Multi-GPU nodes are now equipped with advanced interconnects (e.g., NVIDIA NVLink/NVSwitch) to meet the demand for low-latency, high-bandwidth data exchange [24]. For example, recent top-ranked supercomputers feature up to 8 GPUs per node connected by multiple NVLink channels with aggregate bandwidths in the terabits-per-second range [9, 39]. Optimizing data transfers in the intra-node context, by fully exploiting all available communication paths, is therefore essential to achieve the full compute potential of modern HPC applications.

Despite the availability of multiple GPU communication paths, utilizing them for intra-node GPU communication is a complex problem. In practice, communication libraries, such as Message Passing Interface (MPI) [27], Unified Communication X (UCX) [43], and NVIDIA Collective Communications Library (NCCL) [26], default to a single link for intra-node GPU communication. Moreover, while some studies propose GPU multi-path communication strategies, they lack analytical modeling for multi-path intra-node transfers, and they perform exhaustive search to find the best configuration for each topology [22, 23, 35–37]. The challenge is further compounded by the fact that the communication paths are heterogeneous, with different latencies and bandwidths, and the need to manage data staging and synchronization across these paths. The importance of communication modeling in HPC is well-established, and significant efforts have been made to model and optimize data transfers, including CPU-GPU transfers, inter-node multi-GPU communication, and various collective operations [4, 11, 16, 19, 30, 34, 38]. Classical latency-bandwidth models (such as the *Hockney*'s model) assume a single transfer path and do not capture concurrent use of NVLink and Peripheral Component Interconnect Express (PCIe) channels [1, 8, 12]. More advanced models, address heterogeneity, or middleware costs but they do not target the specific problem of intra-node multi-path communication [3, 7, 15, 31, 33, 40, 45].

In this paper, we address these gaps by developing a predictive performance model for multi-path intra-node GPU communication. Our model extends the classical *Hockney*'s framework to the multi-channel GPU setting, incorporating the latencies and bandwidths of each NVLink and PCIe link as well as the effects of pipelined, staged transfers. Given hardware parameters and message size, the model analytically computes end-to-end transfer time for candidate



SC Workshops '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767392

multi-path configurations, enabling dynamic selection of the optimal strategy without exhaustive search. We show that this model accurately predicts the best (or near-best) configuration, enabling auto-tuning of intra-node GPU communication.

Furthermore, we integrate our model into the UCX framework, based on the multi-path engine introduced in [35]. We validate our model using a set of tests from OSU Micro-Benchmarks (OMB) [6] on a multi-GPU node with NVLink and PCIe interconnects. Our results show that the model can accurately predict the optimal configuration for intra-node GPU P2P communication, achieving up to 2.9x speedup over single-path methods, with an average of less than 6% error in predicting the optimal configuration for messages larger than 4MB. Moreover, our evaluations suggest that not only does this method improve P2P communication, but it also enhances the performance of collective operations such as `MPI_Allreduce` and `MPI_Alltoall`, by up to 1.4x compared to the single-path versions. Our contributions are as follows:

- We develop a performance model for multi-path intra-node GPU communication, enabling the dynamic configuration of data transfers across multiple paths. To the best of our knowledge, this is the first analytical model for multi-path intra-node GPU communication, predicting data transfer times under pipelined, staged intra-node GPU transfers.
- We prove that the optimal multi-path schedule (minimizing overall transfer time) is achieved when the per-path transfer times are equal (or nearly equal).
- We integrate the performance model into the MPI +UCX communication stack for automatic runtime tuning over the pipeline engine developed in our previous work [35].
- We validate our model using a set of P2P and collective micro-benchmarks on a multi-GPU node with NVLink and PCIe interconnects, achieving a mean error below 6% in predicting the optimal configuration for messages larger than 4MB. We demonstrate the effectiveness of the model-enabled multi-path communication strategy in both P2P and collective operations, achieving up to 2.9x and 1.4x speedup, respectively, over the single-path methods.

The rest of this paper is structured as follows. Section 2 covers relevant background and related work. Section 3 presents our performance model for multi-path intra-node GPU communication, and Section 4 includes its design and integration into the MPI+UCX stack. Section 5 presents experimental results and performance analysis, and Section 6 concludes the paper.

2 Background and Related Work

2.1 GPU-enabled Systems, MPI, and UCX

MPI is one of the most widely used communication libraries in HPC, which provides an extensive set of communication primitives for distributed applications [20]. Configured with UCX and Unified Communication Collectives (UCC), MPI implementations, such as Open MPI and MPICH, provide a performant communication framework for GPU-accelerated applications [21, 27, 32, 42–44].

The UCX library handles intra-node Compute Unified Device Architecture (CUDA) Inter-Process Communication (IPC) by its internal module, named `cuda_ipc`, which is responsible for managing CUDA IPC transfers and caching the CUDA IPC handles translations [32, 43]. This module is designed to perform GPU-to-GPU transfers of MPI processes as one-sided `PUT` or `GET` operations. In this way, UCX can efficiently transfer data between GPUs without involving the CPU, which is particularly beneficial for applications with large data transfers. Although this research is developed using NVIDIA GPU hardware and their CUDA platform, the proposed model can be applied to any other GPU architecture with similar multi-path communication capabilities.

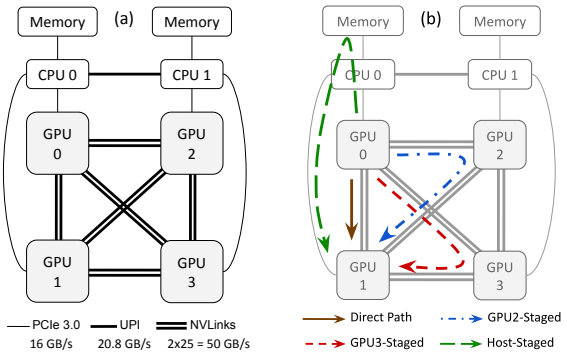


Figure 1: (a) A typical four-GPU node with NVLink per GPU pair, (b) A communication from GPU-0 to GPU-1 is split through multiple paths. (Courtesy of [35])

Figure 1(a) demonstrates a typical four-GPU node with NVLink interconnects. In this configuration, each pair of GPUs has two NVLink sub-links, and the GPUs are connected to the CPUs through PCIe. In this setup, if a communication occurs between GPU-0 and GPU-1, the data transfer can be split into smaller chunks and routed through multiple available paths concurrently, as shown in Figure 1(b). This approach allows for better utilization of the available bandwidth and reduces the overall transfer time when the direct link between the two GPUs is saturated. As will be discussed in Section 5, harnessing host-side available paths (PCIe lanes) may have an adverse effect on performance when they are already used by other processes or threads.

Previous studies demonstrated that when a communication channel (e.g., GPU-0 to GPU-1) reaches its bandwidth limit, the data transfer can be divided into smaller chunks and routed through multiple available paths concurrently to gain considerable improvements [35–37]. However, managing such concurrent communication requires careful handling of data staging, synchronization, and ordering to maintain data integrity and achieve optimal performance. As the number of paths and chunks per path increases, the complexity of managing these aspects also increases, and finding the optimal configuration becomes a challenge.

2.2 Communication Performance Modeling

Communication performance modeling provides analytical models for the evaluation of communication costs based on various platform-specific parameters. Over the past years, numerous models were invented, ranging from homogeneous to heterogeneous clusters. These models aim to predict the communication time accurately, guiding algorithm design, and informing architectural decisions. Some of the notable models include: *Hockney* [12], *LogP* [8], *LogGP* [13], *LogSC* [46], and τ-*Lop* [29].

One of the foundational models in communication performance is the *Hockney*'s linear model [12], which is a good starting point

for understanding communication costs due to its simplicity and effectiveness. Due to its simplicity, it has played an important role in several research directions, such as the evaluation and optimization of collective algorithms, improving collective performance in switched networks, and studying collectives across different topologies in homogeneous/heterogeneous clusters. The *Hockney*'s model is also used, in slightly less related areas to this study, as the network model in simulation tools like SimGrid, and it forms the foundation for scalability analysis in benchmarks like High Performance Linpack (HPL) [30].

2.3 GPU Communication Modeling and Optimization

The invention of GPU accelerators has initiated several research efforts into modeling and optimizing data transfers. Many of these studies initially were focused on CPU-GPU data transfers, often occurring over PCIe links. These transfers are frequently identified as performance bottlenecks, and modeling efforts have aimed to predict PCIe transfer times and develop strategies for overlapping computation and communication [11, 47]. Asynchronous data transfers using mechanisms like CUDA *streams* have been modeled to estimate performance improvements through overlap. Studies have also explored optimal data partitioning and granularity for CPU-GPU transfers [5, 10, 16]. Models like *mHLogGP* have been introduced specifically for CPU/GPU heterogeneous computing environments, abstracting communication and memory access to estimate program running time and guide optimization [17]. Analytical models, sometimes combined with machine learning approaches, have been used for predicting CPU-GPU transfer times, highlighting the importance of accurate predictions [28].

More recently, introduction of technologies like GPUDirect by NVIDIA has enabled direct GPU-to-GPU communication in multi-GPU nodes and clusters, which is faster than PCIe-based transfers [25]. Studies have evaluated GPU interconnect topologies and quantified the benefits of these technologies, through various performance analyses [4, 14]. Several studies have evaluated distributed deep learning frameworks on multi-GPU and multi-node systems, highlighting the impact of communication overhead on training efficiency [15, 34]. Efforts to optimize communication in multi-GPU systems also include model-based approaches for auto-tuning and scheduling in domains like linear algebra (BLAS) and Conjugate Gradient (CG), addressing data transfer bottlenecks [2, 3, 18].

Optimizing communication by leveraging multiple available paths or network interfaces is commonly studied in the context of inter-node communication, in which techniques like multi-HCA-aware designs and rail binding/sharing strategies are proposed to enhance collective communication performance [40]. These approaches aim to improve resource utilization by effectively using multiple network interfaces simultaneously, sharing a similar goal to our work of exploiting all available communication paths. However, these studies focus on inter-node communication, whereas our work targets intra-node communication and the distribution of a P2P transfer. Similarly, libraries like TCCL focus on optimizing collective communication over PCIe-dependent GPU clusters by finding optimal paths for collective algorithms [15], which differs from our focus on distributing a P2P transfer.

While the evaluated studies explore various aspects of multi-GPU communication, including inter-node scenarios and collective operations, studies on analytical performance models specifically designed for optimizing an intra-node GPU-to-GPU transfer by distributing it across multiple, heterogeneous paths (NVLink and PCIe) are less common. Our work directly addresses this gap by providing the first analytical model for multi-path intra-node P2P GPU communication that explicitly considers heterogeneous NVLink and PCIe links for distributing a transfer.

3 Performance Model for Multi-path Intra-node GPU Communication

In this study, we first focus on those intra-node communications that are isolated and not affected by other processes or threads. This assumption helps us to simplify the model and find the highest achievable performance. However, in practice, intra-node GPU interconnects are often shared among multiple processes, which may lead to contention and reduced performance, but as we will discuss later, our approach still accelerates concurrent intra-node communication, including collectives, if there are any under-utilized paths. In this way, the underlying P2P communication is distributed across multiple paths, consequently improving the overall performance of the collective operation. In such cases, the model can still be applied, but the optimal configuration may vary based on the specific workload and contention patterns, and if the communication pattern (determined by the application workload or the collective communication algorithm) can be known ahead of time, unused paths can be extracted and utilized for multi-path communication more effectively.

In this section, we present our performance model based on the *Hockney*'s model. Then, we extend it to account for multi-path communication and derive an optimization problem to find the optimal path configuration.

3.1 Extending the Hockney's Model for Multi-Path Communication

As discussed in Section 2.2, the *Hockney*'s model is widely used for message-passing communication, defined as:

$$T = \alpha + \frac{n}{\beta} \tag{1}$$

where the parameters are defined in Table 1. Considering a system with p distinct available communication paths between two GPUs, we can classify these paths as:

(1) **Direct** GPU-to-GPU path: A direct link connects the source and destination GPUs.
(2) **GPU-Staged** path: Data can be staged through an intermediate GPU before reaching the destination.
(3) **Host-Staged** path: Data can be staged through the host memory before reaching the destination GPU.

While the direct path time can be formulated using Equation (1), the staged paths are a combination of two direct transfers. Each path i has an associated latency α_i and bandwidth β_i. Therefore, the time to transfer a fraction θ of the message n on path i can be specified as (covering both the direct and staged paths):

$$T_i = \alpha_i + \frac{\theta_i n}{\beta_i} + \epsilon_i + \alpha_i' + \frac{\theta_i n}{\beta_i'}, \tag{2}$$

Notation	Description
T	Total communication time
n	Message size
α	Startup Latency
β	Bandwidth
p	Number of paths
T_i	Communication time of path i
α_i, β_i	Model parameters of path i
θ_i	Fraction of message on path i
ϵ_i	Synchronization overhead at staging device i
α'_i, β'_i	Parameters for the second link in staged transfer
Δ_i	$\alpha_i + \alpha'_i + \epsilon_i$
Ω_i	$1/\beta_i + 1/\beta'_i$
ϕ_i	Topology-specific constant for path i
k_i	Number of chunks used along the path i
$T_i(c)$	Time for transfer chunk c on path i

Table 1: Notations used in the model.

where α' and β' represents the parameters of the second link in a staged transfer, and ϵ_i is the overhead of synchronization on the staging device i. Note that:

$$0 \le \theta_i \le 1 \quad \text{and} \quad \sum_{i=1}^{p} \theta_i = 1 \tag{3}$$

Therefore, the overall time required for communication using p paths in parallel can be formulated as:

$$T = \max_{i \in \{1,\dots,p\}} T_i, \text{ where } T_i = \alpha_i + \frac{\theta_i n}{\beta_i} + \epsilon_i + \alpha'_i + \frac{\theta_i n}{\beta'_i} \tag{4}$$

3.2 Finding The Optimal Path Configuration

To find the optimal path configuration, we need to minimize the overall communication time T by adjusting the fraction θ_i of the message size n on each path i, given that we find the other parameters α_i, β_i, and ϵ_i from the system. This can be formulated as a constrained optimization problem:

$$\min_{\theta_i} T = \max_{i \in \{1,\dots,p\}} T_i, \text{ subject to } \sum_{i=1}^{p} \theta_i = 1 \text{ and } 0 \le \theta_i \le 1 \tag{5}$$

This optimization problem can be solved analytically, and we can prove that the optimal solution is when the communication time of all paths are equal, i.e., $T_i = T_j$ for all $i, j \in \{1, \dots, p\}$. To simplify the notation, we assume that every communication path is a direct path, i.e., the communication time of path i can be expressed as:

$$T_i = \alpha_i + \frac{\theta_i n}{\beta_i} \tag{6}$$

This assumption does not affect the generality of the proof, as the same principle applies to staged paths. The only difference is that the time consumption of path i will be a combination of two direct paths, as shown in Equation (2).

Theorem 1. *Let $T_i = \alpha_i + \frac{\theta_i n}{\beta_i}$ be the communication time of path i for a message of size n. For the optimization problem* (5)*, if the optimal solution is such that for any $i, j \in \{1, \dots, p\}$, $T_i \neq T_j$, we have $\alpha_i < \alpha_j + \frac{n\theta_j}{\beta_j}$, then the optimal solution is when the time consumption of each path is equal, i.e., $T_i = T_j$ for all $i, j \in \{1, \dots, p\}$. (Proof is possible by contradiction and is removed due to lack of space.)*

Now, we can find the optimal values of θ_i that minimizes the overall communication time T. According to Theorem 1, the optimal solution is achieved when the communication time across all paths is equal, i.e., $T_i = T_j$ for all $i, j \in \{1, \dots, p\}$. Therefore, we have:

$$\alpha_1 + \frac{\theta_1 n}{\beta_1} = \alpha_i + \frac{\theta_i n}{\beta_i},$$

for all $i \in \{1, \dots, p\}$. This implies that:

$$\theta_i = \frac{\beta_i(\alpha_1 - \alpha_i)}{n} + \frac{\beta_i}{\beta_1}\theta_1. \tag{7}$$

Substituting Equation (7) into the constraint $\sum_{i=1}^{p} \theta_i = 1$, we get:

$$\sum_{i=1}^{p} \left(\frac{\beta_i(\alpha_1 - \alpha_i)}{n} + \frac{\beta_i}{\beta_1}\theta_1 \right) = 1$$

Summing up the terms, we get:

$$\theta_1 = \frac{\beta_1}{\sum_{i=1}^{p} \beta_i} \left(1 - \frac{\alpha_1}{n} \sum_{i=1}^{p} \beta_i + \frac{1}{n} \sum_{i=1}^{p} \alpha_i \beta_i \right)$$

Therefore, the optimal values of θ_i that minimizes the overall communication time T is:

$$\theta_i = \frac{\beta_i}{\sum_{j=1}^{p} \beta_j} \left(1 - \frac{\alpha_i}{n} \sum_{j=1}^{p} \beta_j + \frac{1}{n} \sum_{j=1}^{p} \alpha_j \beta_j \right) \tag{8}$$

Equation (8) indicates that if path i has a higher bandwidth, it is assigned a larger fraction of the total message. This is intuitively correct, as a higher-bandwidth path can transmit more data within the same amount of time. Conversely, if path i exhibits higher latency (i.e., larger α_i), it receives a smaller portion of the message, since its fixed startup cost makes it less efficient for data transfer.

3.3 Incorporating Paths with Staged Transfers

Staged transfers can be either GPU-staged or host-staged. In either case, the time for a staged transfer can be modeled as a combination of two direct transfers. As discussed in Section 3.1, the time for a staged transfer can be modeled in Equation (2). By using the same constraint as before, we can determine the optimal values of θ_i that minimizes the overall communication time T. The only difference is that we need to consider both links and the associated overhead of synchronization on the staging devices. Therefore, for all $i \in \{1, \dots, p\}$, we have:

$$\alpha_1 + \frac{\theta_1 n}{\beta_1} + \epsilon_1 + \alpha_{1'} + \frac{\theta_1 n}{\beta_{1'}} = \alpha_i + \frac{\theta_i n}{\beta_i} + \epsilon_i + \alpha'_i + \frac{\theta_i n}{\beta'_i}$$

This implies that:

$$\theta_i = \frac{1}{n\left(\frac{1}{\beta_i} + \frac{1}{\beta'_i}\right)} \left[\alpha_1 + \alpha_{1'} + \epsilon_1 - (\alpha_i + \alpha'_i + \epsilon_i) + \theta_1 n \left(\frac{1}{\beta_1} + \frac{1}{\beta_{1'}} \right) \right] \tag{9}$$

To simplify the notations, for each i, we define the following terms:

$$\Omega_i = \frac{1}{\beta_i} + \frac{1}{\beta'_i} \text{ and } \Delta_i = \alpha_i + \alpha'_i + \epsilon_i$$

Therefore, we can rewrite Equation (9) as:

$$\theta_i = \frac{\Delta_1 - \Delta_i + \theta_1 n \Omega_1}{n\Omega_i} \tag{10}$$

Once again, by substituting the above equation into the mentioned constraint $\sum_{i=1}^{p} \theta_i = 1$, we get:

$$\sum_{i=1}^{p} \frac{\Delta_1 - \Delta_i + \theta_1 n \Omega_1}{n\Omega_i} = 1$$

Solving for θ_1, we get:

$$\theta_1 = \frac{1}{\Omega_1 \sum_{i=1}^{p} \frac{1}{\Omega_i}} \left(1 - \frac{\Delta_1}{n} \sum_{i=1}^{p} \frac{1}{\Omega_i} + \frac{1}{n} \sum_{i=1}^{p} \frac{\Delta_i}{\Omega_i}\right)$$

Therefore, the optimal values of θ_i that minimizes the overall communication time T including staged transfers are:

$$\theta_i = \frac{1}{\Omega_i \sum_{j=1}^{p} \frac{1}{\Omega_j}} \left(1 - \frac{\Delta_i}{n} \sum_{j=1}^{p} \frac{1}{\Omega_j} + \frac{1}{n} \sum_{j=1}^{p} \frac{\Delta_j}{\Omega_j}\right) \tag{11}$$

Note that Equation (11) generalizes Equation (8) to account for staged transfers. In the special case where path i is a direct path, we have $\Omega_i = 1/\beta_i$ and $\Delta_i = \alpha_i$. Substituting these parameters into Equation (11) leads to Equation (8), confirming it as a specific instance of the more general formulation.

Figure 2(b) illustrates how a message from GPU-0 to GPU-1 can be transferred using four distinct paths: (A) direct path, (B) GPU-staged transfer through GPU-2, (C) GPU-staged transfer through GPU-3, and (D) host-staged transfer. So far, we have discussed staging without pipelining which is path (D) in Figure 2(b).

3.4 Incorporating Pipelining for Staged Transfers

Now, we introduce a pipelining model in which the message is divided into k chunks to enable pipelining. Communication along each staged path proceeds through a repetitive three-step process for all the chunks:

(1) Initiate the transfer of a chunk from the source GPU to the staging location.
(2) Insert a synchronization point to ensure the chunk has arrived at the staging location.
(3) Initiate the transfer of the chunk from the staging location to the destination GPU.

If the path i handles k_i chunks, the time required to transfer a single chunk c through this path is given by:

$$T_i(\mathrm{c}) = \alpha_i + \frac{\theta_i n/k_i}{\beta_i} + \epsilon_i + \alpha_i' + \frac{\theta_i n/k_i}{\beta_i'} \tag{12}$$

With pipelining, the overall communication time is determined by the slower of the two links in the staged path, since both segments operate concurrently and must remain synchronized for continuous data flow. Therefore, we can differentiate between two cases based on the bandwidth of the two links along the path:

$$T_i = \begin{cases} k_i \left(\alpha_i + \frac{\theta_i n/k_i}{\beta_i}\right) + \epsilon_i + \alpha_i' + \frac{\theta_i n/k_i}{\beta_i'}, & \text{if } \beta_i < \beta_i', \\ \alpha_i + \frac{\theta_i n/k_i}{\beta_i} + k_i \left(\epsilon_i + \alpha_i' + \frac{\theta_i n/k_i}{\beta_i'}\right), & \text{if } \beta_i \geq \beta_i'. \end{cases} \tag{13}$$

Paths (B) and (C) from Figure 2(b) depicts the first and the second cases of Equation (13), respectively. To determine the optimal number of chunks to minimize the data transfer, we analyze the total communication time for both scenarios in the following.

Case 1: If the first link (source to the staging device) is the bottleneck, the total communication time is given by:

$$T_i^{\text{Case1}} = k_i \alpha_i + \frac{\theta_i n}{\beta_i} + \epsilon_i + \alpha_i' + \frac{\theta_i n}{k_i \beta_i'}.$$

To minimize T_i^{Case1}, we take its derivative with respect to k_i and set to zero. This approach is commonly used in scenarios with overlapping communication (or computation) [10, 11, 47]:

$$\frac{dT_i^{\text{Case1}}}{dk} = \alpha_i - \frac{\theta_i n}{k_i^2 \beta_i'} = 0.$$

This gives us:

$$k_i = \sqrt{\frac{\theta_i n}{\alpha_i \beta_i'}}. \tag{14}$$

Case 2: When the second link (staging device to the destination) is the bottleneck, the optimal chunk count is:

$$k_i = \sqrt{\frac{\theta_i n}{\beta_i (\epsilon_i + \alpha_i')}}. \tag{15}$$

We can merge the two cases into one equation by defining a new variable $\lambda_i = \lfloor \beta_i'/\beta_i \rfloor$, which is 1 if the first link is the bottleneck, and 0 otherwise. This allows us to express the T_i as:

$$T_i = \lambda_i T_i^{\text{Case1}} + (1 - \lambda_i) T_i^{\text{Case2}}. \tag{16}$$

However, for more simplicity and without loss of generality, we can analyze the two cases separately. For the first case, we can express T_i as (intermediate steps omitted):

$$T_i^{\text{Case1}} = 2\sqrt{\frac{\theta_i n \alpha_i}{\beta_i'}} + \frac{\theta_i n}{\beta_i} + \epsilon_i + \alpha_i'. \tag{17}$$

For comparison, the results for the second case would be:

$$T_i^{\text{Case2}} = 2\sqrt{\frac{\theta_i n (\epsilon_i + \alpha_i')}{\beta_i}} + \frac{\theta_i n}{\beta_i'} + \alpha_i. \tag{18}$$

To find the optimal values for θ_i, we can use the same approach as before. So, we set the communication time of each path to be equal, i.e., $T_1 = T_i$; then, we can isolate θ_i to find its optimal value. However, this process will be much more complex for Equation (17) and Equation (18) than the previous equations, as these ones are not linear anymore. Therefore, We cannot isolate θ_i directly, and numerical methods are required to proceed further.

However, as our goal is to find the optimal values of θ_i in real-time, we need to avoid using numerical methods. Instead, we can approximate Equations (14) and (15) with linear alternatives. Finding the topology-related constants (in the form of $c.f(n)$) for either cases helps us to approximate k_i values. Details of computing these constants are omitted for brevity. Therefore, we can express k_i values for both cases as linear functions by using the topology constants ϕ_i:

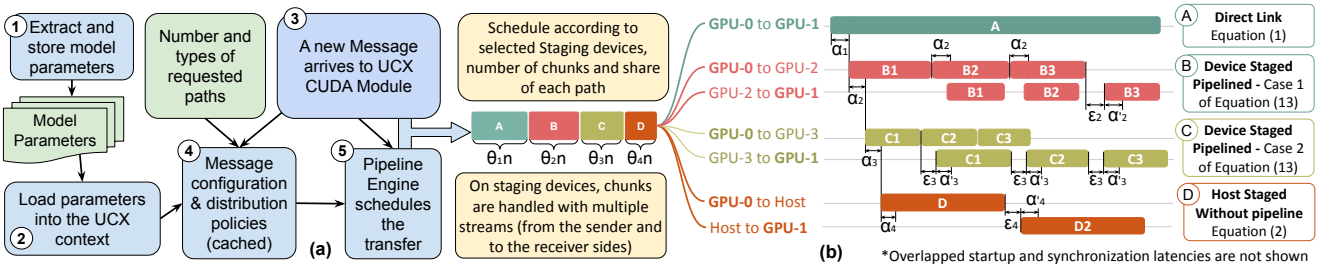


Figure 2: Part (a) depicts the high-level design and implementation of the communication model in the UCX library, and Part (b) shows how a single transfer from GPU-0 to GPU-1 is split into four parts and assigned to different paths, some of which may send data with multiple chunks. Here, the startup and synchronization overheads are exaggerated for clarity.

$$k_i = \begin{cases} \phi_i^1 \cdot \frac{\theta_i n}{\alpha_i \beta_i'}. & \text{if } \beta_i < \beta_i', \\ \phi_i^2 \cdot \frac{\theta_i n}{(\epsilon_i + \alpha_i')\beta_i}. & \text{if } \beta_i \geq \beta_i'. \end{cases} \tag{19}$$

Therefore, we can express T_i as:

$$T_i = \begin{cases} \theta_i n \left(\frac{1}{\beta_i} + \frac{\phi_i^1}{\beta_i'} \right) + \epsilon_i + \alpha_i' + \frac{\alpha_i}{\phi_i^1}, & \text{if } \beta_i < \beta_i', \\ \theta_i n \left(\frac{\phi_i^2}{\beta_i} + \frac{1}{\beta_i'} \right) + \alpha_i + \frac{\epsilon_i + \alpha_i'}{\phi_i^2}, & \text{if } \beta_i \geq \beta_i'. \end{cases} \tag{20}$$

With some simplifications, we can express T_i as:

$$T_i = \theta_i n \Omega_i + \Delta_i, \tag{21}$$

where Ω_i and Δ_i are defined as:

$$\begin{cases} \Omega_i = \frac{1}{\beta_i} + \frac{\phi_i^1}{\beta_i'}, \Delta_i = \epsilon_i + \alpha_i' + \frac{\alpha_i}{\phi_i^1} & \text{if } \beta_i < \beta_i', \\ \Omega_i = \frac{\phi_i^2}{\beta_i} + \frac{1}{\beta_i'}, \Delta_i = \alpha_i + \frac{\epsilon_i + \alpha_i'}{\phi_i^2} & \text{if } \beta_i \geq \beta_i'. \end{cases} \tag{22}$$

Now, with the same approach of equating T_1 and T_i, we can isolate θ_i.

$$\theta_i = \frac{\Delta_1 - \Delta_i + \theta_1 n \Omega_1}{n \Omega_i}. \tag{23}$$

Similar to the Section 3.3, we can substitute the above equation into the constraint $\sum_{i=1}^{p} \theta_i = 1$ and solve for θ_1. Then, we can formulate the general equation for θ_i as the following, which is similar to Equation (11):

$$\theta_i = \frac{1}{\Omega_i \sum_{j=1}^{p} \frac{1}{\Omega_j}} \left(1 - \frac{\Delta_i}{n} \sum_{j=1}^{p} \frac{1}{\Omega_j} + \frac{1}{n} \sum_{j=1}^{p} \frac{\Delta_j}{\Omega_j} \right). \tag{24}$$

4 Design and Implementation of the Proposed Communication Model

To implement the proposed communication model, we have integrated our design into the UCX library. As shown in Figure 2(a), the performance model parameters are extracted once per system topology and stored on each compute node (**Step 1**). At program startup, the model is loaded into the UCX context (**Step 2**). Subsequently, for each data transfer that reaches the CUDA IPC module of UCX, the model is invoked to compute the optimal transfer configuration (**Step 3**). By default, all available paths between the source and destination GPUs are considered. However, this behavior can be modified via environment variables to selectively include or exclude paths. In **Step 4**, the model determines the optimal configuration, including the number of chunks and the size of each chunk per path. Note that any path, except the direct one, may be excluded as a result of the optimization process. Finally in **Step 5**, the computed configuration is forwarded to the pipeline engine, which performs the actual communication scheduling based on previous studies [35].

Algorithm 1 demonstrates the procedure of computing the optimal path configuration, given the source and destination GPUs, the data size, and an initial list of available paths. The algorithm begins by checking whether a cached configuration exists for the specified source-destination pair and the available paths (Lines 4-6). If no cached entry is found, it proceeds to compute the path configuration. For each path, the model calculates the parameters Ω_i and Δ_i using the performance model parameters α, β, and ϵ (Lines 7-18). Based on these computed parameters, and the topology-specific constants, the algorithm then determines the appropriate data size to assign to each path in the rest of the algorithm (Lines 20-30).

An important detail is the sequential nature of transfer initiation across the paths. To account for this, the latency α from previously scheduled paths must be accumulated and included in the computation of Δ_i for subsequent paths. This adjustment is performed in Line 18 of Algorithm 1.

5 Performance Evaluation and Analysis

To evaluate the performance of our framework, we conducted a series of tests to compare these three different configurations:

- **Static Path Distribution**: Refers to a fixed, predetermined configuration across available paths, chosen statically (offline), where the distribution strategy is extracted by exhaustive search, similar to the work in [35].
- **Model-Driven Prediction**: This is our proposed approach, where we report the prediction of the model of the bandwidth, not the actual measurement.
- **Dynamic Path Distribution**: This is the implementation of our model in the runtime library to dynamically compute the model's parameters and to find the best distribution policy.

To assess the performance of our framework, we conducted our evaluations based on the OMB MPI unidirectional (`BW`) and bidirectional (`BIBW`) P2P bandwidth tests [6]. We also performed `MPI_Allreduce` and `MPI_Alltoall` tests to demonstrate the effectiveness of our model-driven multi-path communication strategy in collective operations. In all of the test cases, the runtime overhead

Algorithm 1: Optimal Path Configuration $O(paths\_count)$

Input: $src\_dev\_id, dst\_dev\_id, data\_size, paths[]$
Output: $configs[], shares[]$
Function $populate\_path\_config$:
  if *cached config exists* **then**
    return cached config
  end
  for $i \leftarrow 0$ **to** $paths\_count$ **do**
    if $paths[i] = direct$ **then**
      $paths[i] \leftarrow get\_link(src, dst)$
    else if $paths[i] = host\_staged$ **then**
      $paths[i] \leftarrow get\_link(src, host), get\_link(host, dst)$
    else
      $paths[i] \leftarrow get\_link(src, paths[i]), get\_link(paths[i], dst)$
    end
  end
  for $i \leftarrow 0$ **to** $paths\_count$ **do**
    Compute Ω_i, Δ_i Based on α, β, ϵ
    Add previous α value to Δ_i
    $compute\_topology\_constant()$
    Approximate number of chunks k_i
  end
  $inv\_sum \leftarrow \sum 1/\Omega_i$ and $delta\_sum \leftarrow \sum \Delta_i/\Omega_i$
  for $i \leftarrow 0$ **to** $paths\_count$ **do**
    $\theta_i \leftarrow$ normalized formula using Ω, Δ
    $shares[i] \leftarrow \theta_i \cdot data\_size$
  end
  if $remaining > 0$ **then**
    Give leftover to direct
  end
  store config in cache
  return config, shares
end

of the model-driven framework is negligible for large message sizes (less than 0.1% of the total execution time).

5.1 Experimental Setup

We developed our model-driven framework by extending the UCX library and the multi-path engine introduced in [35]. We utilized Open MPI v5.0.4 configured with UCC v1.3, UCX v1.14.0, and CUDA v12.6 for the experiments, which were performed on two different multi-GPU node configurations:

- **Beluga**: As depicted in Figure 1, Beluga GPU nodes are equipped with four NVIDIA V100s, with two pairs of NVLink-V2 between any two GPUs. The four directly connected GPUs with one of the CPUs are configured as a single Non-Uniform Memory Access (NUMA) node.
- **Narval**: An eight-NUMA node system equipped with four A100 GPUs, with a full mesh topology and four pairs of NVLink-V3 between any two GPUs. As shown in Figure 3, each GPU is configured in a single NUMA node, which consists of a single memory channel and only six cores of one of the CPUs.

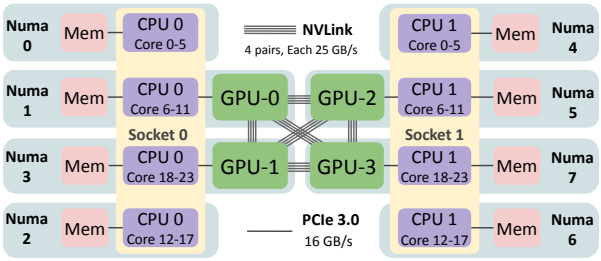


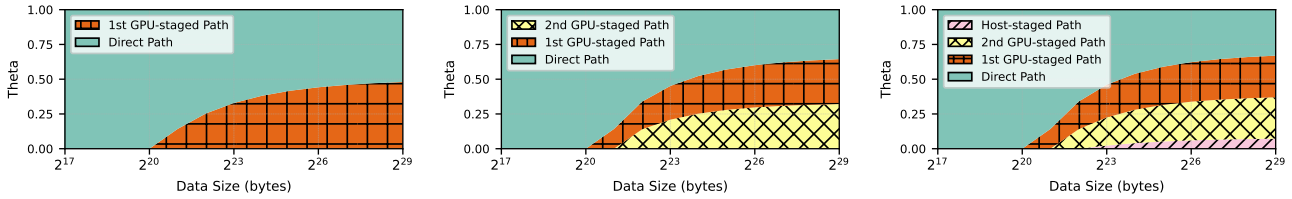
Figure 3: Narval node architecture.

5.2 Unidirectional and Bidirectional Bandwidth Evaluation

We conducted the OMB unidirectional and bidirectional *bandwidth* tests using two different *window* sizes: 1 and 16, for message sizes ranging from 2MB to 512MB. Performance is measured in terms of aggregate bandwidth, and we report prediction error as a percentage deviation from the observed optimal performance. The results are shown in Figure 5 and Figure 6, for unidirectional and bidirectional tests, respectively. In these figures, the label *2_GPUs* refers to a transfer using 2 paths, the direct NVLink plus 1 GPU-staged paths. Similarly, *3_GPUs* refers to the direct and 2 GPU-staged paths, and finally, *3_GPUs_w_host* refers to a transfer with 4 paths: the direct, 2 GPU-staged, and 1 host-staged paths.

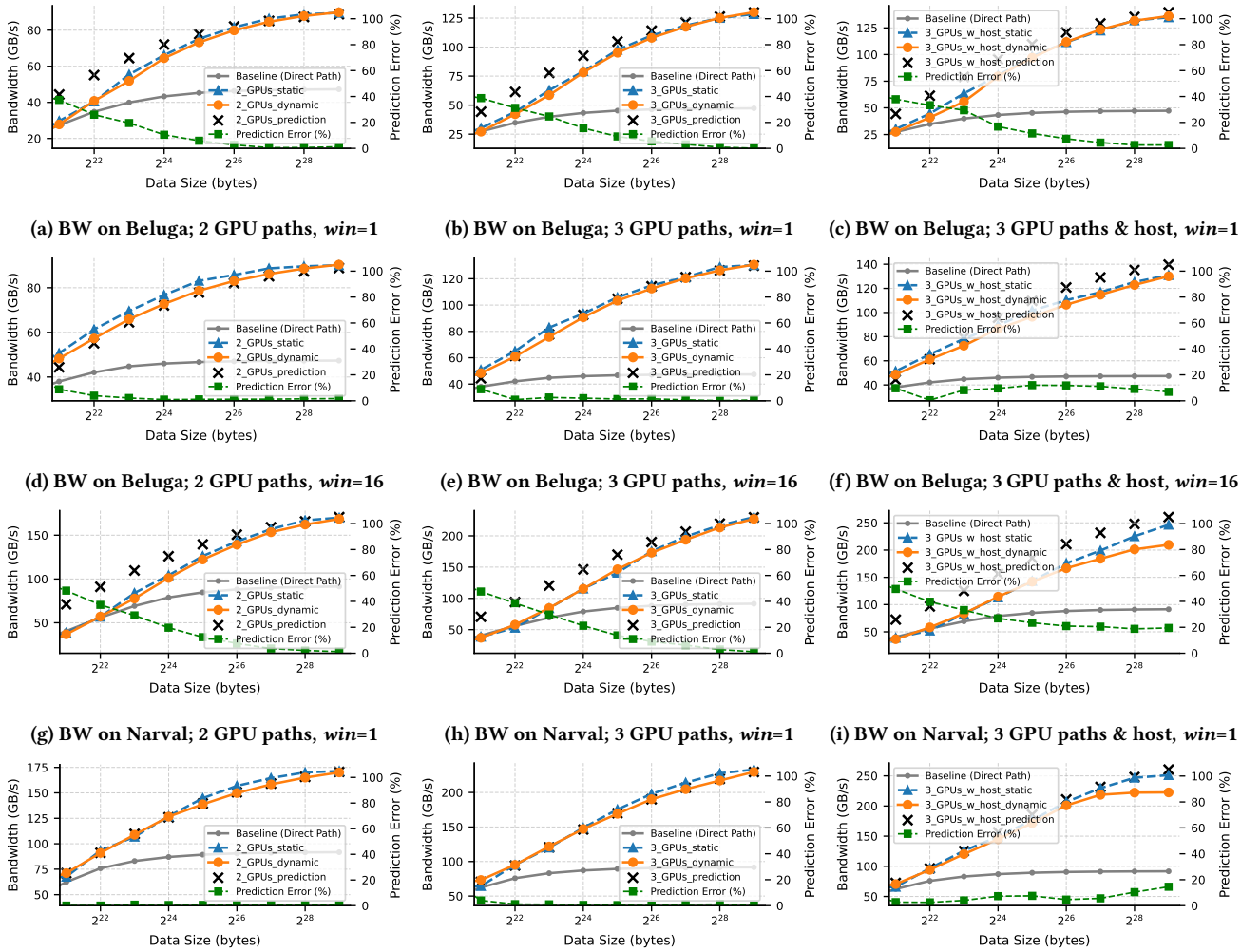
Figure 4 shows the distribution of the θ (message fraction) values across multiple paths for OMB MPI unidirectional bandwidth tests on Beluga. The results reveal how much each path contributes to the overall bandwidth for each message size. Figure 4(c) shows the distribution across four paths: direct, two GPU-staged, and one host-staged. The following observations can be made from the evaluations:

- **Observation 1**: For message sizes greater than 8MB, the model-driven prediction closely matches the observed optimal performance in `BW` test, with an average prediction error of less than 6% across all configurations and clusters. In contrast, the prediction error for `BIBW` tests is higher, averaging around 8% for non-host-staged across both clusters and *window* sizes.
- **Observation 2**: As the number of *windows* increases, the performance gap between the static and dynamic path distributions narrows. Similarly, the prediction errors decrease with higher number of *windows*. This is due to the fact that larger *window* sizes allow for more concurrent transfers, reducing the impact of latency and bandwidth variations (see Figures 5(h) and 5(k)).
- **Observation 3**: The prediction errors are generally higher when the host-staged path is enabled in the configuration for `BW` tests. This issue is particularly present on Narval, where the host-staged path is less efficient due to the NUMA configuration. As can be inferred from Figure 3, host-staged transfers on Narval includes an extra transfer (through Ultra Path Interconnect (UPI) or equivalent), because the memory channels are not shared between the GPUs.
- **Observation 4**: The model-driven predictions are less accurate for smaller message sizes on both clusters, especially for small *window* sizes. This is due to the linear nature of the *Hockney*'s model and the way the asymptotic bandwidth is calculated. This issue is more pronounced for `BW` tests compared to `BIBW` tests.



(a) 2 Paths: direct and 1 GPU-staged (b) 3 Paths: direct and 2 GPU-staged (c) 4 Paths: direct, 2 GPU-, and 1 Host-staged

Figure 4: θ distribution when utilizing different number of paths for OMB MPI unidirectional bandwidth tests on Beluga.



(a) BW on Beluga; 2 GPU paths, *win*=1 (b) BW on Beluga; 3 GPU paths, *win*=1 (c) BW on Beluga; 3 GPU paths & host, *win*=1

(d) BW on Beluga; 2 GPU paths, *win*=16 (e) BW on Beluga; 3 GPU paths, *win*=16 (f) BW on Beluga; 3 GPU paths & host, *win*=16

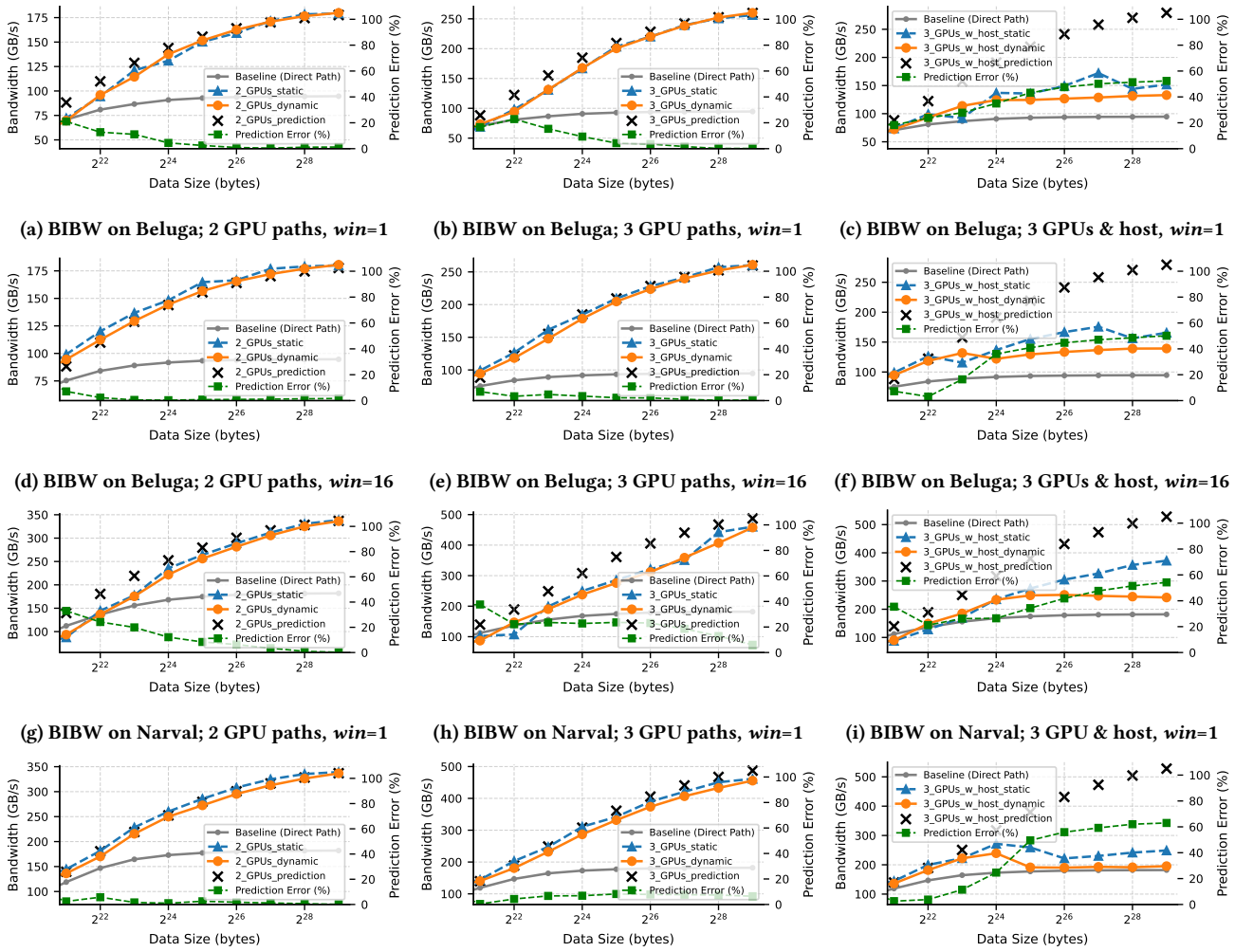
(g) BW on Narval; 2 GPU paths, *win*=1 (h) BW on Narval; 3 GPU paths, *win*=1 (i) BW on Narval; 3 GPU paths & host, *win*=1

(j) BW on Narval; 2 GPU paths, *win*=16 (k) BW on Narval; 3 GPU paths, *win*=16 (l) BW on Narval; 3 GPU paths & host, *win*=16

Figure 5: Unidirectional MPI bandwidth tests on Beluga and Narval. Comparing the predicted and two measured configurations (dynamic- and static-tuned) with the baseline (Direct Path) using various number of paths and *window* sizes.

As Figures 5(a), 5(b), 5(g), and 5(h) show, the model tends to overestimate the performance for smaller message sizes.

- **Observation 5**: The `BIBW` tests demonstrate worse performance with the presence of the host-staged path on both clusters, as shown in Figures 6(c), 6(f), 6(i), and 6(l). This is due to the contention between the bidirectional transfers along the host-staged path, which is not considered in our performance model. This observation suggests that while host staging still increases the overall bandwidth comparing to single-path communication, the contention between the bidirectional



(a) BIBW on Beluga; 2 GPU paths, *win*=1 (b) BIBW on Beluga; 3 GPU paths, *win*=1 (c) BIBW on Beluga; 3 GPUs & host, *win*=1

(d) BIBW on Beluga; 2 GPU paths, *win*=16 (e) BIBW on Beluga; 3 GPU paths, *win*=16 (f) BIBW on Beluga; 3 GPUs & host, *win*=16

(g) BIBW on Narval; 2 GPU paths, *win*=1 (h) BIBW on Narval; 3 GPU paths, *win*=1 (i) BIBW on Narval; 3 GPU & host, *win*=1

(j) BIBW on Narval; 2 GPU paths, *win*=16 (k) BIBW on Narval; 3 GPU paths, *win*=16 (l) BIBW on Narval; 3 GPUs & host, *win*=16

Figure 6: Bidirectional MPI bandwidth tests on Beluga and Narval. Comparing the predicted and two measured configurations (dynamic- and static-tuned) with the baseline (Direct Path) using various number of paths and *window* sizes.

transfers degrades the performance, and the model should be extended to account for host-staged contention, PCIe configuration, and NUMA effects.

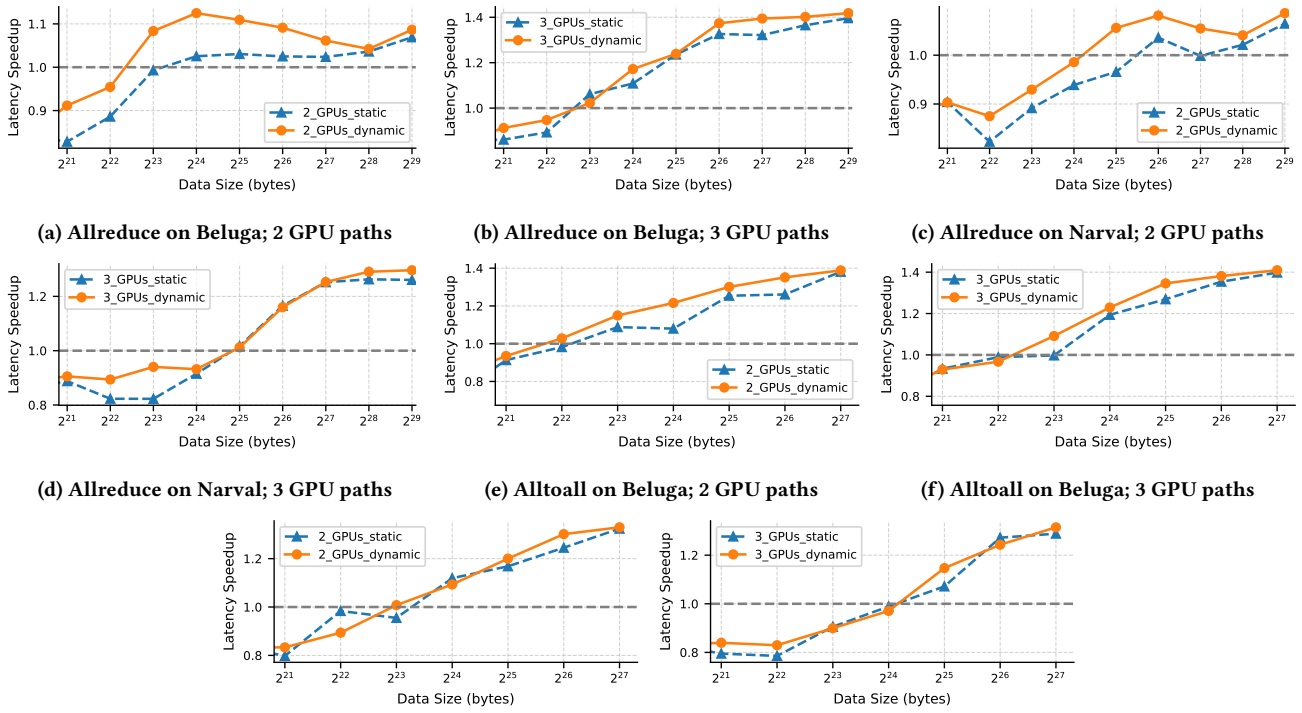
5.3 Collective Communication Evaluation

Although the primary focus of this work is on P2P communication, we also evaluate the performance of collective communication operations, specifically `MPI_Allreduce` and `MPI_Alltoall`. These operations are essential for many parallel applications and can benefit from the intra-node GPU communication optimizations discussed in this work.

We evaluated the performance of the collective operations on both Beluga and Narval clusters, using the same configurations as for the P2P tests. We utilized the UCC library and its Unified Communication Protocols (UCP) module for each collective operation. For `MPI_Allreduce` and large message sizes, UCP uses the *recursive K-nomial scatter-reduce followed by K-nomial allgather* algorithm, and for `MPI_Alltoall`, it uses the *Bruck* algorithm. These algorithms consist of multiple non-blocking P2P communication steps which are handled by UCX under the hood. With this setup, we can enforce the utilization of the UCX `cuda_ipc` module for intra-node communication, where our framework is integrated.

We compared the performance of our model-driven path distribution against the default `MPI+UCC+UCX` stack, with the results presented in Figure 7 (x-axis is the size of message per rank/GPU). As discussed in Section 5.2, host-staging degrades the `BIBW` performance due to contention on the host side. Therefore, for collective operations, we present the results without host-staging configuration. The following observations can be made from the results:

- **Observation 1**: Enabling multi-path transfers improves the performance of both collective operations on both clusters, but the improvements are more significant on Beluga.



(a) Allreduce on Beluga; 2 GPU paths

(b) Allreduce on Beluga; 3 GPU paths

(c) Allreduce on Narval; 2 GPU paths

(d) Allreduce on Narval; 3 GPU paths

(e) Alltoall on Beluga; 2 GPU paths

(f) Alltoall on Beluga; 3 GPU paths

(g) Alltoall on Narval; 2 GPU paths

(h) Alltoall on Narval; 3 GPU paths

Figure 7: Latency speedup of MPI Alltoall and Allreduce tests on Beluga and Narval against MPI+UCC+UCX configuration.

- **Observation 2**: Overall, the results indicate that our model-driven approach outperforms the statically-tuned multi-path configuration for both collective operations on both clusters.
- **Observation 3**: The performance improvement is higher for `MPI_Alltoall` compared to `MPI_Allreduce`, which is due to the computation overhead in `MPI_Allreduce` (Compare Figure 7(a) with Figure 7(e), and Figure 7(c) with Figure 7(g)). However, with three GPU paths, the performance difference between the two collective operations is lower (Compare Figure 7(b) with Figure 7(f)).
- **Observation 4**: For `MPI_Allreduce`, the performance improvement is more pronounced with three GPU paths compared to two GPU paths (Compare Figure 7(a) with Figure 7(b), and Figure 7(c) with Figure 7(d)). However, for `MPI_Alltoall`, the performance improvement with three GPU paths is marginal compared to two GPU paths (Compare Figure 7(e) with Figure 7(f), and Figure 7(g) with Figure 7(h)). This is due to the different communication patterns of the two collective operations and how adding more paths does not lead to significant gains in this case.

6 Conclusion and Future Work

In this work, we designed and evaluated a performance model for multi-path intra-node GPU communication, considering the latencies and bandwidths of each NVLink and PCIe link as well as the effects of pipelined, staged transfers. We showed that the optimal multi-path schedule (minimizing overall transfer time) is achieved when the per-path transfer times are equal (or nearly equal). We demonstrated the effectiveness of the model-enabled multi-path communication strategy in both P2P and collective operations, achieving up to 2.9x and 1.4x speedup, respectively, over the single-path methods. We evaluated the accuracy of our model, achieving less than 6% error in predicting the optimal configuration for messages larger than 4MB in unidirectional transfers.

For future work, we plan to extend our model to support more complex intra-node communication patterns, such as collective operations and multi-node communication. We also aim to explore the possibilities of utilizing other performance models as the basis for our multi-path model, such as *MaxRate* when considering contention on shared links in a loaded network. Finally, we will investigate the application of our model to other hardware architectures and interconnects, like NVSwitch-based systems and AMD GPUs.

Acknowledgments

This research was supported by Digital Research Alliance of Canada. Computations were performed on Narval and Beluga with support from Calcul Québec (calculquebec.ca). We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [ALLRP 5785392022].

References

[1] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1995. LogGP: Incorporating Long Messages into the LogP Model. *J. Parallel and Distrib. Comput.* (1995), 95–105. https://doi.org/10.1145/215399.215427

[2] Petros Anastasiadis, Nikela Papadopoulou, Georgios Goumas, and Nectarios Koziris. 2021. CoCoPeLia: Communication-Computation Overlap Prediction for Efficient Linear Algebra on GPUS. In *Proceedings of IEEE International*

Symposium on Performance Analysis of Systems and Software (ISPASS). 36–47. https://doi.org/10.1109/ISPASS51385.2021.00015

[3] Petros Anastasiadis, Nikela Papadopoulou, Georgios Goumas, Nectarios Koziris, Dennis Hoppe, and Li Zhong. 2023. PARALiA: A Performance Aware Runtime for Auto-tuning Linear Algebra on Heterogeneous Systems. *ACM Transactions on Architecture and Code Optimization* 20, 4 (2023), 1–25. https://doi.org/10.1145/3624569

[4] Amanda Bienz, Luke N. Olson, William D. Gropp, and Shelby Lockhart. 2021. Modeling Data Movement Performance on Heterogeneous Architectures. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7. https://doi.org/10.1109/HPEC49654.2021.9622742 arXiv:2010.10378

[5] Michael Boyer, Jiayuan Meng, and Kalyan Kumaran. 2013. Improving GPU performance prediction with data transfer modeling. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*. IEEE, 1097–1106. https://doi.org/10.1109/IPDPSW.2013.236

[6] Devendar Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda. 2012. OMB-GPU: A micro-benchmark suite for evaluating MPI libraries on GPU clusters. In *Proceedings of the European MPI Users' Group Meeting (EuroMPI)*. 110–120. https://doi.org/10.1007/978-3-642-33518-1_16

[7] Jesús Cámara, Javier Cuenca, Victor Galindo, Arturo Vicente, and Murilo Boratto. 2025. An autotuning approach to select the inter-GPU communication library on heterogeneous systems. *Journal of Supercomputing* 81, 1 (2025), 1–16. https://doi.org/10.1007/s11227-024-06794-3

[8] 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 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, Vol. Part F1296. 1–12. https://doi.org/10.1145/155332.155333

[9] Daniele De Sensi, Lorenzo Pichetti, Flavio Vella, Tiziano De Matteis, Zebin Ren, Luigi Fusco, Matteo Turisini, Daniele Cesarini, Kurt Lust, Animesh Trivedi, Duncan Roweth, Filippo Spiga, Salvatore Di Girolamo, and Torsten Hoefler. 2024. Exploring GPU-to-GPU Communication: Insights into Supercomputer Interconnects. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 1–15. https://doi.org/10.1109/SC41406.2024.00039 arXiv:2408.14090

[10] Frédéric Desprez, Pierre Ramet, and Jean Roman. 1996. Optimal grain size computation for pipelined algorithms. *Lecture Notes in Computer Science* (1996), 165–172. https://doi.org/10.1007/3-540-61626-8_21

[11] Juan Gómez-Luna, José María González-Linares, José Ignacio Benavides, and Nicolás Guil. 2012. Performance models for asynchronous data transfers on consumer Graphics Processing Units. *J. Parallel and Distrib. Comput.* 72, 9 (2012), 1117–1126. https://doi.org/10.1016/j.jpdc.2011.07.011

[12] Roger W. Hockney. 1994. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.* 20, 3 (1994), 389–398. https://doi.org/10.1016/S0167-8191(06)80021-9

[13] T. Hoefler, T. Schneider, and A. Lumsdaine. 2009. LogGP in theory and practice - An in-depth analysis of modern interconnection networks and benchmarking methods for collective operations. *Simulation Modelling Practice and Theory* 17, 9 (2009), 1511–1521. https://doi.org/10.1016/j.simpat.2009.06.007

[14] Harsh Khetawat, Nikhil Jain, Abhinav Bhatele, and Frank Mueller. 2024. Predicting GPUDirect Benefits for HPC Workloads. *Proceedings of Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (2024), 88–97. https://doi.org/10.1109/PDP62718.2024.00020

[15] Heehoon Kim, Junyeol Ryu, and Jaejin Lee. 2024. TCCL: Discovering Better Communication Paths for PCIe GPU Clusters. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Vol. 3. 999–1015. https://doi.org/10.1145/3620666.3651362

[16] Bozhong Liu, Weidong Qiu, Lin Jiang, and Zheng Gong. 2016. Software pipelining for graphic processing unit acceleration: Partition, scheduling and granularity. *International Journal of High Performance Computing Applications* 30, 2 (2016), 169–185. https://doi.org/10.1177/1094342015585845

[17] Gangfeng Liu, Yunlan Wang, Tianhai Zhao, Jianhua Gu, and Dongyang Li. 2012. mHLogGP: A Parallel Computation Model for CPU/GPU. *Network and Parallel Computing* (2012), 217–224. https://doi.org/10.1007/978-3-642-35606-3_25

[18] Shelby Lockhart, Amanda Bienz, William Gropp, and Luke Olson. 2023. Performance Analysis and Optimal Node-aware Communication for Enlarged Conjugate Gradient Methods. *ACM Transactions on Parallel Computing* 10, 1 (2023), 1–25. https://doi.org/10.1145/3580003

[19] Shelby Lockhart, Amanda Bienz, William D. Gropp, and Luke N. Olson. 2023. Characterizing the performance of node-aware strategies for irregular point-to-point communication on heterogeneous architectures. *Parallel Comput.* 116, September 2022 (2023), 1–12. https://doi.org/10.1016/j.parco.2023.103021

[20] MPI Forum. 2025. https://www.mpi-forum.org/ [Accessed: 2025-04-01].

[21] MPICH. 2025. https://www.mpich.org/ [Accessed: 2025-04-01].

[22] Akira Nukada. 2021. Performance Optimization of Allreduce Operation for Multi-GPU Systems. In *Proceedings of the IEEE International Conference on Big Data (Big Data)*. IEEE, 3107–3112.

[23] Akira Nukada. 2022. Performance Optimization of Allreduce Operation for Multi-GPU Systems. In *Proceedings of the International Conference on Big Data (Big Data)*. IEEE, 1–6. https://doi.org/10.1109/bigdata52589.2021.9672073

[24] NVIDIA. 2025. https://www.nvidia.com/ [Accessed: 2025-04-01].

[25] NVIDIA. 2025. GPUDirect Technologies. https://developer.nvidia.com/gpudirect [Accessed: 2025-04-01].

[26] NVIDIA. 2025. NVIDIA Collective Communications Library. https://github.com/NVIDIA/nccl [Accessed: 2025-04-01].

[27] Open MPI. 2025. https://www.open-mpi.org/ [Accessed: 2025-04-01].

[28] Ali Riahi, Abdorreza Savadi, and Mahmoud Naghibzadeh. 2020. Comparison of analytical and ML-based models for predicting CPU–GPU data transfer time. *Computing* 102, 9 (2020), 2099–2116. https://doi.org/10.1007/s00607-019-00780-x

[29] Juan Antonio Rico-Gallego and Juan Carlos Díaz-Martín. 2015. τ-Lop: Modeling performance of shared memory MPI. *Parallel Comput.* 46 (2015), 14–31. https://doi.org/10.1016/j.parco.2015.02.006

[30] Juan A. Rico-Gallego, Juan C. Díaz-Martín, Ravi Reddy Manumachu, and Alexey L. Lastovetsky. 2019. A survey of communication performance models for high-performance computing. *Comput. Surveys* 51, 6 (2019), 1–36. https://doi.org/10.1145/3284358

[31] Whit Schonbein, Scott Levy, Matthew G.F. Dosanjh, W. Pepper Marts, Elizabeth Reid, and Ryan E. Grant. 2023. Modeling and Benchmarking the Potential Benefit of Early-Bird Transmission in Fine-Grained Communication. *Proceedings of the International Conference on Parallel Processing (ICPP)* (2023), 306–316. https://doi.org/10.1145/3605573.3605618

[32] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, and Yiftah Shahar. 2015. UCX : An Open Source Framework for HPC Network APIs and Beyond. In *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 40–43. https://doi.org/10.1109/HOTI.2015.13

[33] Hamed Sharifian, Amirhossein Sojoodi, and Ahmad Afsahi. 2024. A Topology- and Load-Aware Design for Neighborhood Allgather. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*. 1–12. https://doi.org/10.1109/CLUSTER59578.2024.00019

[34] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. 2018. Performance modeling and evaluation of distributed deep learning frameworks on GPUs. *Proceedings of the IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)* (2018), 943–948. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.000-4

[35] Amirhossein Sojoodi, Yıltan Hassan Temucin, and Ahmad Afsahi. 2024. Enhancing Intra-Node GPU-to-GPU Performance in MPI + UCX through Multi-Path Communication. In *Proceedings of the International Workshop on Extreme Heterogeneity Solutions (ExHET)*. 1–6. https://doi.org/10.1145/3642961.3643800

[36] Yıltan Hassan Temucin, Amirhossein Sojoodi, Pedram Alizadeh, and Ahmad Afsahi. 2021. Efficient Multi-Path NVLink / PCIe-Aware UCX based Collective Communication for Deep Learning. In *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI)*. 1–10. https://doi.org/10.1109/HOTI52880.2021.00018

[37] Yıltan Hassan Temucin, Amirhossein Sojoodi, Pedram Alizadeh, Benjamin W Kitor, and Ahmad Afsahi. 2021. Accelerating Deep Learning using Interconnect-Aware UCX Communication for MPI Collectives. *IEEE Micro* (2021), 1–9. https://doi.org/10.1109/MM.2022.3148670

[38] Andreas Thune, Sven-arne Reinemo, Tor Skeie, and Xing Cai. 2023. Detailed modeling of heterogeneous and communication. *IEEE Transactions on Parallel and Distributed Systems* (2023), 1–14. https://doi.org/10.1109/TPDS.2023.3253881

[39] Top500. 2025. https://top500.org/ [Accessed: 2025-04-01].

[40] Tu Tran, Bharath Ramesh, Benjamin Michalowicz, Mustafa Abduljabbar, Hari Subramoni, Aamir Shafi, and Dhabaleswar K. Panda. 2024. Accelerating communication with multi-HCA aware collectives in MPI. *Concurrency and Computation: Practice and Experience* 36, 1 (2024), 1–20. https://doi.org/10.1002/cpe.7879

[41] Didem Unat, Ilyas Turimbetov, Mohammed Kefah, Taha Issa, Flavio Vella, Daniele D E Sensi, and Ismayil Ismayilov. 2024. The Landscape of GPU-Centric Communication. *arXiv* (2024), 1–25. arXiv:arXiv:2409.09874v2

[42] Unified Communication Framework Consortium. 2025. Unified Collective Communication (UCC). https://github.com/openucx/ucc [Accessed: 2025-04-01].

[43] Unified Communication Framework Consortium. 2025. Unified Communication X (UCX). https://openucx.org/ [Accessed: 2025-04-01].

[44] Manjunath Gorentla Venkata, Valentine Petrov, Sergey Lebedev, Devendar Bureddy, Ferrol Aderholdt, Joshua Ladd, Gil Bloch, Mike Dubman, and Gilad Shainer. 2024. Unified Collective Communication (UCC): An Unified Library for CPU , GPU , and DPU Collectives. In *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 37–46. https://doi.org/10.1109/HOTI63208.2024.00018

[45] Jingyuan Wang, Tianhai Zhao, and Yunlan Wang. 2024. Network states-aware collective communication optimization. *Cluster Computing* (2024), 1–19. https://doi.org/10.1007/s10586-024-04330-9

[46] Ziheng Wang, Heng Chen, Xiaoshe Dong, Weilin Cai, and Xingjun Zhang. 2022. LogSC: Model-based one-sided communication performance estimation. *Future Generation Computer Systems* 132 (2022), 25–39.

https://doi.org/10.1016/j.future.2022.02.004

[47] B. Van Werkhoven, J. Maassen, F. J. Seinstra, and H. E. Bal. 2014. Performance models for CPU-GPU data transfers. In *Proceedings of the International Symposium on Cluster, Cloud, and Grid Computing (CCGrid)*. IEEE, 11–20. https://doi.org/10.1109/CCGrid.2014.16