

UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Towards Topology-Aware Communication in Multi-Accelerator Servers

Permalink

<https://escholarship.org/uc/item/5h30s19t>

Author

Ranganath, Kiran

Publication Date

2022

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NonCommercial-ShareAlike License, available at <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Towards Topology-Aware Communication in Multi-Accelerator Servers

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Kiran Ranganath

September 2022

Dissertation Committee:

Dr. Daniel Wong, Chairperson
Dr. Nael Abu-Ghazaleh
Dr. Laxmi Bhuyan

Copyright by
Kiran Ranganath
2022

The Dissertation of Kiran Ranganath is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

This dissertation marks a significant milestone in my life. Reaching this milestone was a journey of learning, grit, and compassion. I am confident the knowledge and experiences earned during my Ph.D. have enabled me to achieve many milestones in the future.

I express my sincerest gratitude to my advisor, Prof. Daniel Wong. I am forever indebted to Prof. Wong for allowing me to pursue Doctoral studies under his tutelage. He is always an infinite source of knowledge, wisdom, and inspiration. He granted me the freedom to pursue research projects and provided all the guidance, feedback, and encouragement to refine ideas and overcome obstacles. I also thank the members of my dissertation committee, Prof. Nael Abu-Ghazaleh and Prof. Laxmi N. Bhuyan. Their valuable feedback and support during my qualifying exam and dissertation defense have helped shape this dissertation.

A significant part of this dissertation was possible with the support of the High-Performance Computing Group at Pacific Northwest National Laboratory. I am fortunate to have been mentored by Dr. Shuaiwen Leon Song, Dr. Joshua Suetterlein, and Dr. Joseph Manzano at the lab. They provided the opportunity to work on cutting-edge research projects that fundamentally shaped this dissertation. They were always there not only as mentors but as collaborators working by my side on several projects.

I am fortunate to have had the most amazing friends, lab-mates, and colleagues, including, but not limited to, Marcus Chow, Dr. AmirAli Abdolrashidi, Dr. Devashree Tripathy, Ali Jahanshahi, Mohammed Reza Rezvani, Ziyang (Max) Jia, and Nafis (Niloy)

Mustakin. I will cherish the time I spent with you all forever. I also thank all the teachers, professors, colleagues, and friends, for their support.

Finally and most importantly, I thank my family for the love, sacrifices, and continuous support in all my endeavors.

This dissertation contains content published in the following publications:

- Kiran Ranganath, AmirAli Abdolrashidi, Shuaiwen Leon Song, and Daniel Wong. “Speeding up collective communications through inter-GPU re-routing.” *IEEE Computer Architecture Letters* 18, no. 2 (2019): 128-131.
- Kiran Ranganath, Joshua D. Suetterlein, Joseph B. Manzano, Shuaiwen Leon Song, and Daniel Wong. “Mapa: Multi-accelerator pattern allocation policy for multi-tenant GPU servers.” In *Proceedings of the International Conference for High-Performance Computing, Networking, Storage and Analysis*, pp. 1-14. 2021.

To my mother.

ABSTRACT OF THE DISSERTATION

Towards Topology-Aware Communication in Multi-Accelerator Servers

by

Kiran Ranganath

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, September 2022
Dr. Daniel Wong, Chairperson

Multi-accelerator servers are increasingly deployed in shared multi-tenant environments (such as in cloud data centers) to meet the demands of large-scale compute-intensive workloads. In addition, these accelerators are increasingly interconnected in complex topologies, and workloads exhibit a wider variety of inter-accelerator communication patterns. However, existing allocation policies are ill-suited for these emerging use cases. Specifically, when the multi-accelerator workloads are commonly fragmented, leading to reduced bandwidth and increased latency for inter-accelerator communication. This dissertation proposes several methods to minimize the impact of *fragmentation* on multi-accelerator workloads.

In the first approach, *WOTIR* avoids low-bandwidth interconnects in fragmented workloads by routing the inter-accelerator communication through an intermediate accelerator. This method utilizes unused high-speed interconnects to improve the speedup of fragmented workloads.

In the second approach, *MAPA* uses a graph mining approach to drive generalized allocation support for allocating multi-accelerator workloads on multi-accelerator servers.

MAPA can reduce the execution time of multi-accelerator workloads and also provide generalized benefits across various accelerator topologies.

The third approach, *CC-Topo*, extends the generalized allocation policy to also select collective communication topology. This could further alleviate the fragmentation within the system.

Contents

List of Figures	xii
List of Tables	xv
1 Introduction	1
2 Speeding up Collective Communications Through Inter-GPU Re-routing	7
2.1 Introduction	7
2.1.1 Multi-GPU Server Architectures	8
2.1.2 Motivation - The Fragmentation Problem	9
2.2 WOTIR	13
2.2.1 NCCL primitives	13
2.2.2 Buffer Size Analysis	15
2.2.3 NVLink Routing	17
2.2.4 Collective Communication Operations	20
2.3 Evaluation	23
2.3.1 Memory Copy Performance	24
2.3.2 Collective Communication Operations	24
2.3.3 Machine Learning Workloads	25
2.3.4 Interference on route-GPU	29
2.4 Related Works	30
2.5 Conclusion	31
3 MAPA: Multi-Accelerator Pattern Allocation Policy for Multi-Tenant GPU Servers	33
3.1 Introduction	33
3.2 Motivation	37
3.2.1 Modern Multi-Accelerator Systems	37
3.2.2 Resource fragmentation in multi-tenant servers	41
3.2.3 Understanding Bandwidth sensitivity of ML workloads	44
3.3 MAPA: Multi-Accelerator Pattern Allocation	46
3.3.1 Application Topology	47

3.3.2	Hardware Topology	50
3.3.3	Pattern Matching	51
3.3.4	Pattern Scoring	53
3.3.5	Pattern Selection Allocation Policy	59
3.3.6	State Management	62
3.4	Evaluation	62
3.4.1	Evaluation on DGX-V System	64
3.5	Exploring Novel Hardware Topologies	69
3.5.1	Methodology	69
3.5.2	Simulation Framework	69
3.5.3	Exploration Results	72
3.5.4	Overhead of Scheduling	75
3.6	Related Works	75
3.7	Conclusion	77
4	Adaptive Collective Communication Topology	79
4.1	Introduction	79
4.2	Motivation	82
4.2.1	Limitations of MAPA	82
4.2.2	Tree and Ring Bandwidths	83
4.3	Collective Communication Topology Awareness	87
4.3.1	Combination Scoring	90
4.3.2	CC-Topo Allocation Policy	96
4.4	Evaluation	97
4.5	Conclusion	99
5	Conclusion & Future Work	100
	Bibliography	102

List of Figures

1.1	8-GPU Hybrid cube mesh topology	3
1.2	Multi-GPU System Fragmentation	4
1.3	WOTIR overview	4
1.4	MAPA overview	5
1.5	MAPA with application topology selection	6
2.1	NVIDIA DGX-1 topology	9
2.2	Placement scenario	11
2.3	Placement effects in GPU-to-GPU communication	12
2.4	NCCL initialized buffers	13
2.5	Copy Primitive	14
2.6	Double Copy Primitive	14
2.7	Reduce Primitive	14
2.8	Reduce Copy Primitive	15
2.9	Forward Primitive	15
2.10	Bandwidth analysis for different buffer sizes in NCCL Copy operation.	16
2.11	NCCL Broadcast	18
2.12	NCCL Reduce	18
2.13	NCCL All-Reduce	19
2.14	NCCL All-Gather	20
2.15	NCCL Reduce-Scatter	21
2.16	Bandwidth and Latency improvement with WOTIR.	23
2.17	Execution time improvement in Caffe of WOTIR compared to PCIe for 3 GPUs	24
2.18	Bandwidth (left) and Latency (right) results for NCCL microbenchmarks.	26
2.19	Execution time of PCIe and WOTIR normalized to NVLink.	28
2.20	All-Reduce Bandwidth scaling with the number of GPUs	29
3.1	Emerging multi-GPU accelerator topologies are increasingly heterogeneous.	34

3.2	Nvidia’s multi-GPU systems exhibit a variety of interconnects. This figure shows the various links available in DGX-1 Volta. These different links have significantly different bandwidths as well as an impact on applications such as CNN training. where NV2-Single and NV2-Double are Single and Double NVLink-v2 links respectively.	35
3.3	The number of Top500 supercomputers with accelerators are increasing, with GPUs being the most common. The ratio of these GPU HPC systems with heterogeneous interconnects has increased over time and is now dominant. .	38
3.4	Due to fragmentation of GPU allocations, a large portion of GPU jobs have sub-optimal allocated aggregate bandwidth ($BW_{Allocated}$) compared to the aggregate bandwidth of an ideal allocation ($BW_{IdealAllocation}$).	42
3.5	Communication Properties of ML workloads	43
3.6	Execution Time trends of Bandwidth Sensitive and Insensitive Networks. .	44
3.7	Overview of the Multi-Accelerator Pattern Allocation(MAPA) system . . .	47
3.8	Example application topology for 5-GPU workload utilizing NCCL collective communication for inter-GPU communication. Application topologies can be a ring (left), tree (middle), or a combination of both (right).	48
3.9	Examples of identifying application topology through source code analysis and runtime profiling.	49
3.10	Illustrative example showing bandwidths used for Aggregated Bandwidth score calculation (left) and Preserved Bandwidth calculation (right) given an allocation [1, 2, 4]	53
3.11	Evaluating pattern scoring metrics. (a) <i>AggBW</i> does not correlate well with execution time. (b) This is due to <i>AggBW</i> not correlating well with the effective achievable bandwidth of an allocation. (c) <i>EffBW</i> correlates well with execution time.	54
3.12	Predicted effective bandwidth correlates well with actual effective bandwidth and generalizes across jobs of different sizes.	59
3.13	Evaluation results (Execution Time) on DGX-V	65
3.14	Evaluation results (Predicted Effective Bandwidth) on DGX-V	66
3.15	Number of jobs scheduled on a given GPU ID.	68
3.16	MAPA simulation execution framework.	69
3.17	Effective bandwidth measured during DGX-V’s real and simulation runs correlated well.	71
3.18	Effective bandwidth vs execution time observed during real runs on DGX-V. .	71
3.19	16-GPU topologies	73
3.20	Simulation results for bandwidth-sensitive workloads on 16-GPU topologies. Improvements to the lower tail (min and 25th percentile) are better in both. .	74
3.21	Overhead analysis of MAPA with Preserve policy.	77
4.1	Nvidia DGX-V Hybrid Cube Mesh topology	80
4.2	Peak bandwidths were observed while running the microbenchmarks for tree and ring application topologies.	83
4.3	Example 3-GPU allocation 0,3,4	84
4.4	Characterizing fragmented allocation 0,3,4	85

4.5	Allocations where Peak bandwidths in Trees were greater than Rings. Where $x = Tree_{BW} - Ring_{BW}$	86
4.6	Training time for 1 epoch on all the possible 4-GPU allocations in DGX-V .	88
4.7	Training time for 1 epoch on all the possible 4-GPU allocations in DGX-V .	89
4.8	Overview of MAPA + CC-top framework	90
4.9	Modeling tree and ring bandwidths using linear regression	91
4.10	NAM model learning for Predicting Ring and Tree Bandwidths. L1Loss is MAE loss function.	94
4.11	NAM results for CC-Topo	95
4.12	CC-Topo evaluation on DGX-V	98

List of Tables

2.1	Distribution of BabelStream turnaround time observed for Interference Analysis	30
3.1	Peak Bandwidths per link	39
3.2	Allocations depicting different communication links in a DGX-V machine shown in Figure 3.1c.	40
3.3	Values of Coefficients.	58
3.4	Summary of results. Normalized execution time speed up and throughput (Tput) observed on DGX-1 V100.	67

Chapter 1

Introduction

Modern workloads are increasingly becoming data-intensive. The demand for speeding up the data-intensive workloads spun off an era of Microarchitectural advancement in Accelerator architectures. The widely adopted General Purpose Processing in Graphical Processing Units (GPGPUs) provide massive parallelism to compute data faster. Today, the Accelerator space is becoming increasingly crowded with multiple accelerator solutions such as Tensor Processing Units (TPUs) [48], Tensor Streaming Processor (TSPs) [6], Wafer Scale Engine (WSE), and Reconfigurable Dataflow Unit (RDUs). These accelerators are poised to keep up with the demand for processing exploding amount of data in the near future. These accelerators are also increasingly deployed in shared environments such as Cloud and High-Performance Computing (HPC). The servers in the shared environments are equipped with many accelerators within a server. The complexity is further compounded when several types of interconnects are at play.

A typical implementation of a distributed multi-accelerator data-intensive workload such as PyTorch involves two parts – Computation and Communication. Computation involves launching optimized kernels on the accelerator efficiently, and Communication is responsible for efficiently transferring data across many accelerators. Computation is facilitated by libraries such as CuDNN [19], CuBLAS, and ROCBLAS whereas Communication is handled by libraries such as NCCL [77] and RCCL [10]. In this dissertation, we focus on ensuring efficient communication during the execution of a workload.

Furthermore, most of the multi-accelerator workloads are run in a shared environment, a job scheduler is generally used to partition and manage the server among one or more jobs. Each of these jobs can have a specific communication pattern. The system is supposed to be partitioned and allocated within a multi-accelerator server, the obtained partition could have varying interconnect bandwidths potentially worsening the Communication cost of a multi-accelerator workload. On the other hand, incorporating a naive allocation strategy in a job scheduler exacerbates this problem leading to inefficient allocations consisting of lower interconnect bandwidth. In this dissertation, we focus on mitigating the effect of such inefficient allocations in a multi-accelerator server.

Hybrid Cube mesh topology shown in Figure 1.1 is one of the popular multi-GPU server architectures in use. This topology is used in Facebook’s Big-basin [26], Amazon’s P3 instances [126], Microsoft Azure’s NDv2 instances [68], and Nvidia’s DGX-V100. Any two GPUs in a hybrid cube mesh topology can have either a single NVlink (GPUs 0 & 1), double NVlinks (GPUs 0 & 3), or PCIe (GPUs 0 & 5). These interconnects can achieve different bandwidths. The heterogeneity in the interconnect bandwidth could potentially

cause pitfalls such as *fragmentation*. A system is said to be fragmented when one or more of the allocations on the system is hindered by low-bandwidth PCIe interconnects as illustrated in Figure 1.2. In this dissertation, we propose three solutions to mitigate the effects of fragmentation.

Chapter 2 motivates the problems associated with naive allocation policies. We use *WOTIR* to effectively circumvent the lower bandwidth interconnects by routing the traffic through a neighboring accelerator by utilizing the available high-speed interconnects. This is achieved by maintaining routing table information for all the low-bandwidth interconnects.

Since the GPU allocation 3, 4, and 5 in Figure 1.2 is fragmented, the inter-GPU communication between 3 to 4 and 3 to 5 are facilitated by PCIe interconnects. In WOTIR,

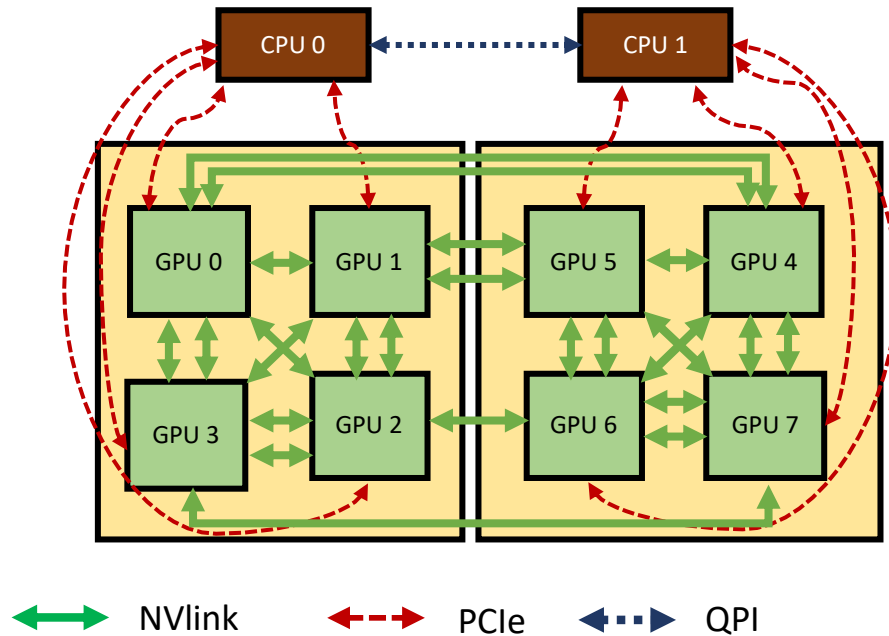


Figure 1.1 8-GPU Hybrid cube mesh topology

these PCIe links are avoided by using GPU 1 for routing the traffic from 3 to 4 and 3 to 7 through GPUs 1 and 7 respectively as shown in Figure 1.3.

Chapter 3 proposes an intelligent allocation policy called Multi-Accelerator Pattern Allocation (MAPA) to improve the quality of allocations. In this approach, the allocation policy tries to find the best available allocation for the job in the queue. The allocation

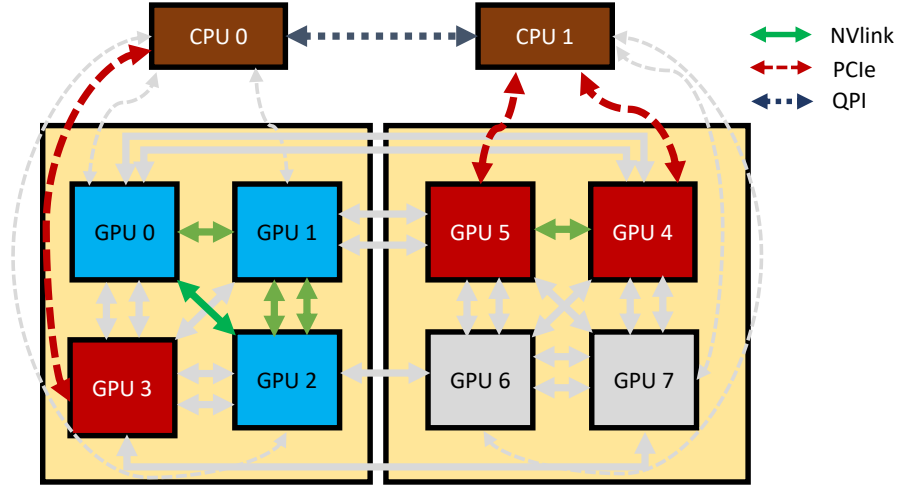


Figure 1.2 Multi-GPU System Fragmentation

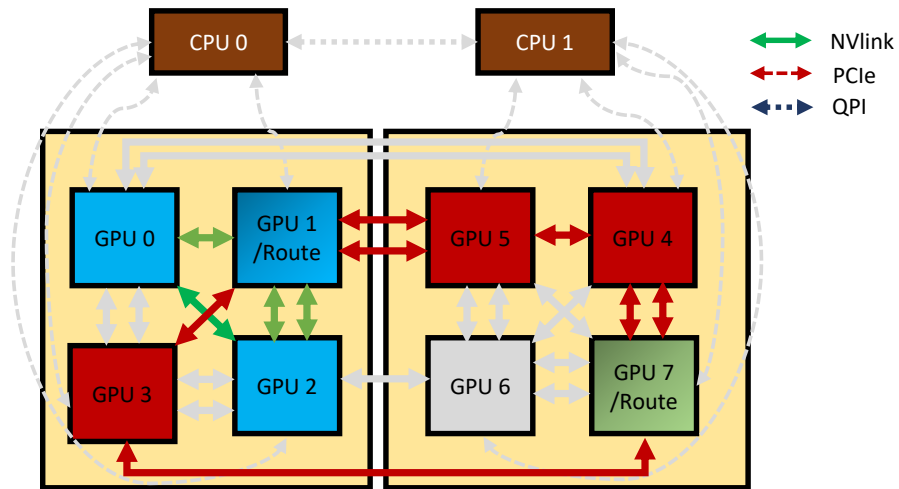


Figure 1.3 WOTIR overview

policy is driven by a regression model that can predict the quality of Communication of any given allocation pattern. The policy also takes into account the sensitivity of any workload to a given allocation.

MAPA relies on the user-provided information about the job to find the best suitable allocation to avoid fragmentation. In Figure 1.4, two jobs are concurrently sharing the multi-tenant GPU environment where both jobs have obtained identically better allocations with NVlinks.

Chapter 4 extends the MAPA to make it into a pattern-agnostic allocation policy. In this approach, the policy identifies the suitable allocation with a collective Communication topology. This chapter uses Neural Additive Models (NAM) to accurately predict bandwidth for different collective communication topologies. This policy can be used to drive the communication topology selection to further reduce fragmentation due to the ability to use rings or trees to avoid PCIe when there are limited allocations in the system.

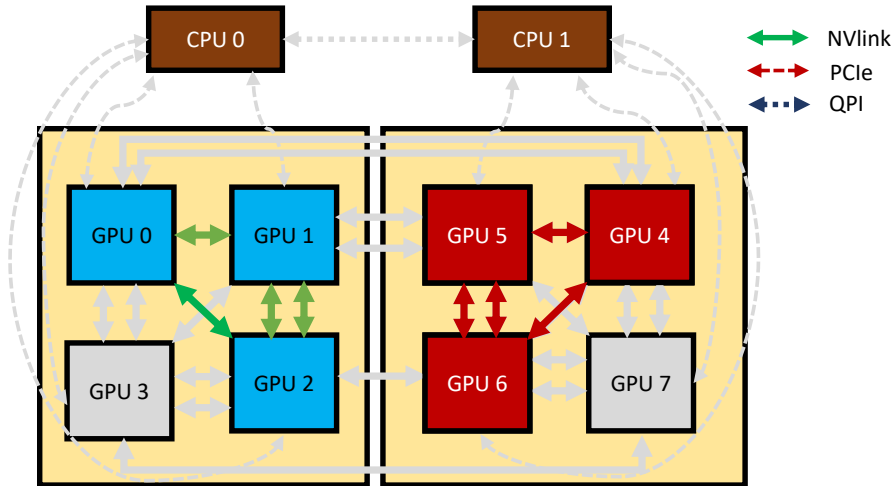


Figure 1.4 MAPA overview

The newest Collective Communication Library capabilities enabled the system to arbitrarily select trees or rings based on the obtained allocation. As shown in Figure 1.5, the GPU allocation 3, 4, and 7 in tree topology is not fragmented (with all NVlinks). On the contrary, forcing a ring on this allocation would have led to system fragmentation due to the unavailability of NVlink interconnection between GPUs 3 and 4.

The solutions (WOTIR, MAPA, and CC-Topo) discussed in this dissertation aim at alleviating the effect of fragmentation by minimizing the use of PCIe links in multi-accelerator jobs. Together, they achieve this by changing the inter-GPU communication paths or finding a better allocation, or selecting a suitable inter-GPU collective communication topology.

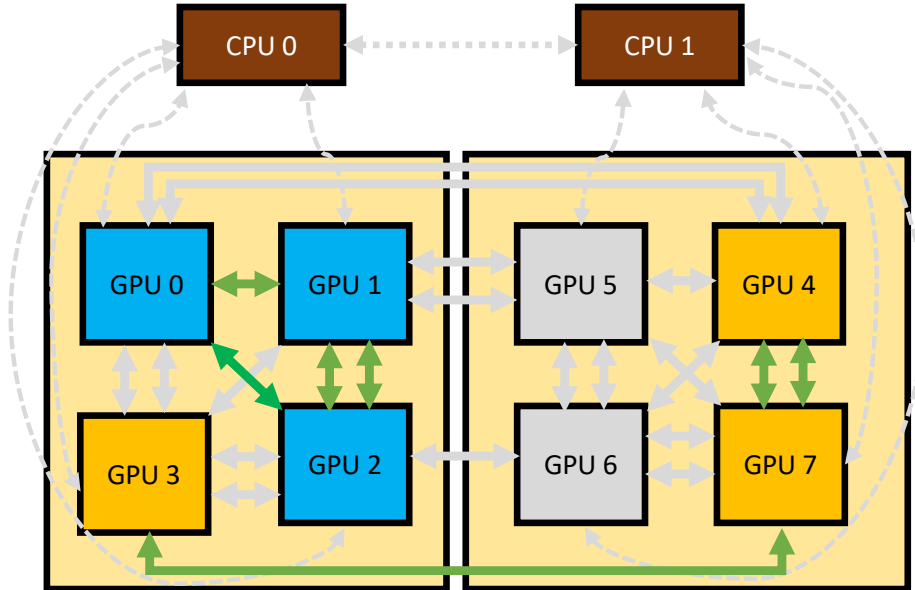


Figure 1.5 MAPA with application topology selection

Chapter 2

Speeding up Collective Communications Through Inter-GPU Re-routing

2.1 Introduction

Accelerator-based computing has become mainstream for many domains that have high computational needs. The most common accelerator architecture used in current systems is GPU-based. The domains that best exploits this increased computational power are usually HPC applications which are highly concurrent and floating point intensive, and Machine Learning (ML) workloads that require a large number of small computations and efficient communication. Moreover, these workloads continue to evolve to efficiently map to the ever-changing computational substrate.

Such computational pipelines have been augmented to take advantage of optimized collective operations provided for the hardware. Vendor-provided solutions such as the *NVIDIA Collective Communications Library (NCCL)* [77] exploit the system under “normal” conditions. However, due to complexities that arise from advanced workflows, frameworks, and multi-user environments, these solutions fail to fully leverage the underlying compute substrate. Domains like machine learning training, in which the process can take days, and graph-based analytics in cloud environments [9, 68] can suffer from these pathological scenarios, since several workloads might be required to co-exist inside a single compute node. Such partitioning of resources might lead to “bad” placements in which GPUs might become isolated¹ for a single workload, leading to inefficient communication.

2.1.1 Multi-GPU Server Architectures

The NVIDIA DGX-1 system was one of the first multi-GPU reference server designs released. This design has been adopted by various supercomputers and cloud data centers. Examples include the Summit Supercomputer [37], Microsoft Olympus [112], and Facebook Big Basin [36].

Figure 2.1 illustrates the GPU network topology in the DGX-1 system. In this server, two CPUs manage 4 GPUs through a PCIe link each (i.e. 8 GPUs in total). Every GPU has direct peer-to-peer access to 3 other GPUs through single and double NVLink connections (NVIDIA’s high-speed GPU-to-GPU interconnect [28]) as shown. In the DGX-1 systems (that use NVIDIA P100 GPUs), NVLink-v1 can reach up to 20 GB/s per link. Newer systems using V100s GPUs employ NVLink-v2 which can achieve up to 25 GB/s per

¹without a direct high-bandwidth communication link between them

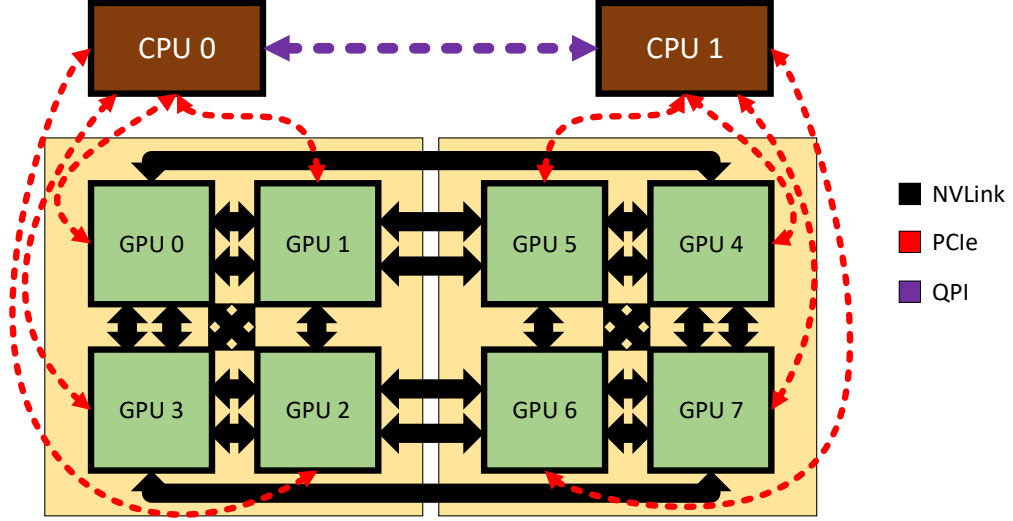


Figure 2.1 NVIDIA DGX-1 topology

link with the same network topology. In these configurations, the black arrows in Figure 2.1 signify peer-to-peer connectivity. Two black arrows mean double NVLink connections. The CUDA runtime API enables NVLink-based device-to-device communication only if the pair of GPUs are directly connected via NVLink [72].

2.1.2 Motivation - The Fragmentation Problem

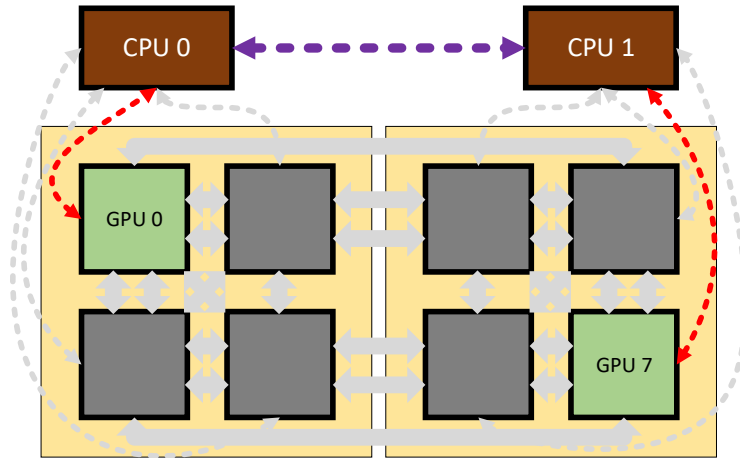
To highlight potential resource allocation and communication issues in cloud environments that can arise in multi-GPU servers, let us consider the following scenario. There is a job queue, where queued Jobs 1 through 4 utilize 1, 4, 2, and 2 GPUs respectively. These jobs can either be batch-type or latency-critical that are provisioned for request-response type workloads. Let us assume a naive scheduling scenario² where Job 1 takes

²which is the current standard policy in NVIDIA Docker to allocate GPUs in cloud environments

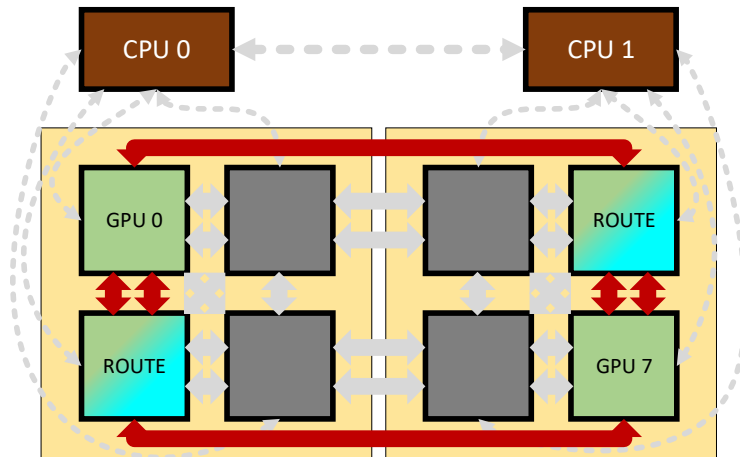
GPU 0; Job 2 takes GPUs 1, 2, 3 & 4; and Job 3 takes GPUs 5 & 6. Since there is only one GPU left, and Job 4 needs 2 GPUs, it waits until 2 GPUs are available. Now assume Job 1 frees GPU 0, and Job 4 is scheduled on GPUs 0 & 7. In the GPU topology shown in Figure 2.2a, GPUs 0 & 7 do not have NVLink connectivity. Although it is desirable to place workloads on adjacent GPUs for faster communication, some of them may already be occupied. Therefore, we would be forced to place the workloads on non-adjacent GPUs, which can add to the communication overhead. We call this problem *fragmentation*. Hence Job 4 running on GPUs 0 and 7 suffers from this communication overhead.

Characterizing the effect of poor placement

The effect on the execution time of a non-optimal placement can be seen in Figure 2.3(a). This figure shows execution times normalized to NVLink, i.e. an optimal placement using GPUs 0 and 1. We observe the execution time using PCIe (GPUs 0 and 7) to be as much as 2x compared to NVLink. To further demonstrate the performance gap between PCIe and NVLink, we present the bandwidth and latency of the memory copy operation in Figures 2.3(b) and 2.3(c). The memory copy is performed on contiguous data allocations with sizes ranging from 1 KB to 1 GB. The bandwidth of the memory copy operation saturates NVLink at a little over 22 GB/s and PCIe at 9 GB/s.

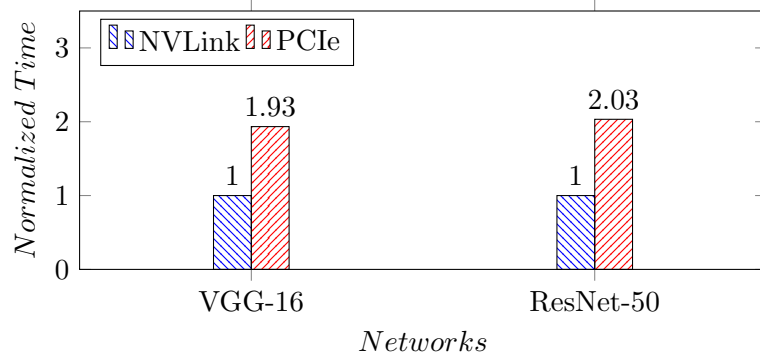


(a) GPUs 0 and 7 use PCIe since there is no direct NVLink

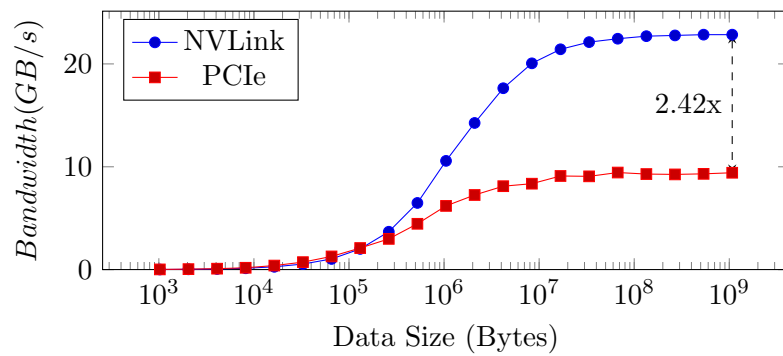


(b) WOTIR establishes NVLink route using *route-GPUs*.

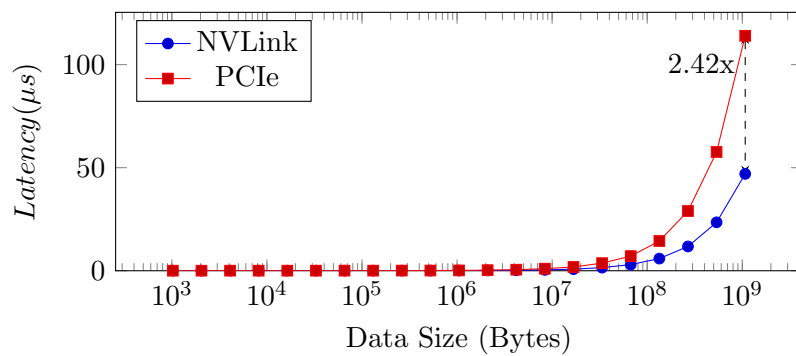
Figure 2.2 Placement scenario



(a) Caffe Training time normalized to NVLink



(b) Bandwidth characterization



(c) Latency characterization

Figure 2.3 Placement effects in GPU-to-GPU communication

2.2 WOTIR

In order to alleviate the degraded network performance due to poor workload placement within a single node, we propose Workload Optimization Through Inter-GPU Re-routing (WOTIR). In this technique, we augment NCCL to use two-hop NVLink-based routing to circumvent PCIe bottlenecks, thereby improving the overall performance. This work currently focuses on tolerating GPU fragmentation within a server node. We shall leave the study of cluster-level GPU fragmentation for future work.

2.2.1 NCCL primitives

NCCL implements various collective communication patterns using coordinated primitives. These primitives operate on input and output data from the application along with the send and receive buffers managed and accessible by the library as shown in Figure 2.4. There is one send buffer that resides on the GPU global memory. This buffer is enabled to be peer accessed by one or more of the participating GPUs as a read-only receive buffer.

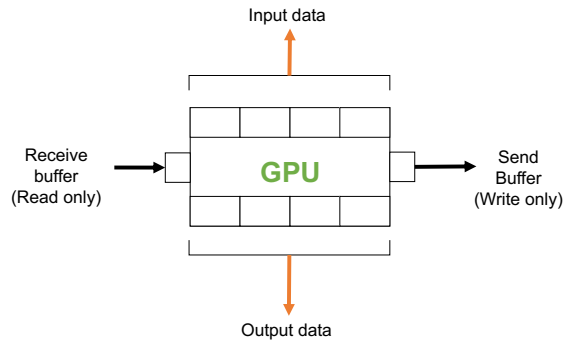


Figure 2.4 NCCL initialized buffers

NCCL uses the following primitives as building blocks to implement collective communication algorithms such as Broadcast, Reduce, All Reduce, All Gather, and Reduce Scatter.

Copy: Copy from receive buffer to GPU's output data region or from GPU's input data region to send buffer as shown in Figure 2.5.

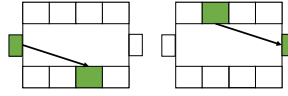


Figure 2.5 Copy Primitive

Double Copy: Copy the data in receive buffer to GPU's output data region along with duplicating the data onto the send buffer as shown in Figure 2.6.

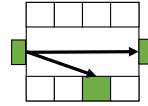


Figure 2.6 Double Copy Primitive

Reduce: Perform arithmetic or logical operation with data in the receive buffer and input data as operands, and store the reduced result in the output data region as shown in Figure 2.7.

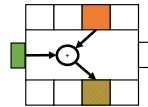


Figure 2.7 Reduce Primitive

Reduce-Copy: Performs reduce operation with input data and receives buffer as operands. The reduced result is stored in the output data region as well as duplicated onto the send buffer so the reduced result is available to the next GPU as shown in Figure 2.8.

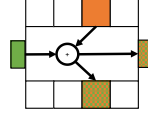


Figure 2.8 Reduce Copy Primitive

Forward: Copy the data directly from receive buffer to send buffer as shown in Figure 2.9. This primitive was introduced to facilitate the implementation of WOTIR in NCCL.

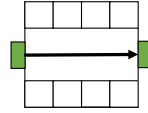


Figure 2.9 Forward Primitive

To facilitate communication, NCCL organizes participating intra-node GPUs in a *ring*. During the setup of these rings, communication channels (PCIe or NVLink) are enabled as shown in Figure 2.2b. These channels assist data movement among participating GPUs through primitive operations.

2.2.2 Buffer Size Analysis

Figure 2.10 shows the bandwidth and latencies observed with varying buffer sizes from 0.125 MB to 4 MB. It can be noticed that starting at 2 MB buffer sizes, peak achievable bandwidth caps at nearly 23 GBps. Hence, a 2 MB buffer size was found to be ideal.

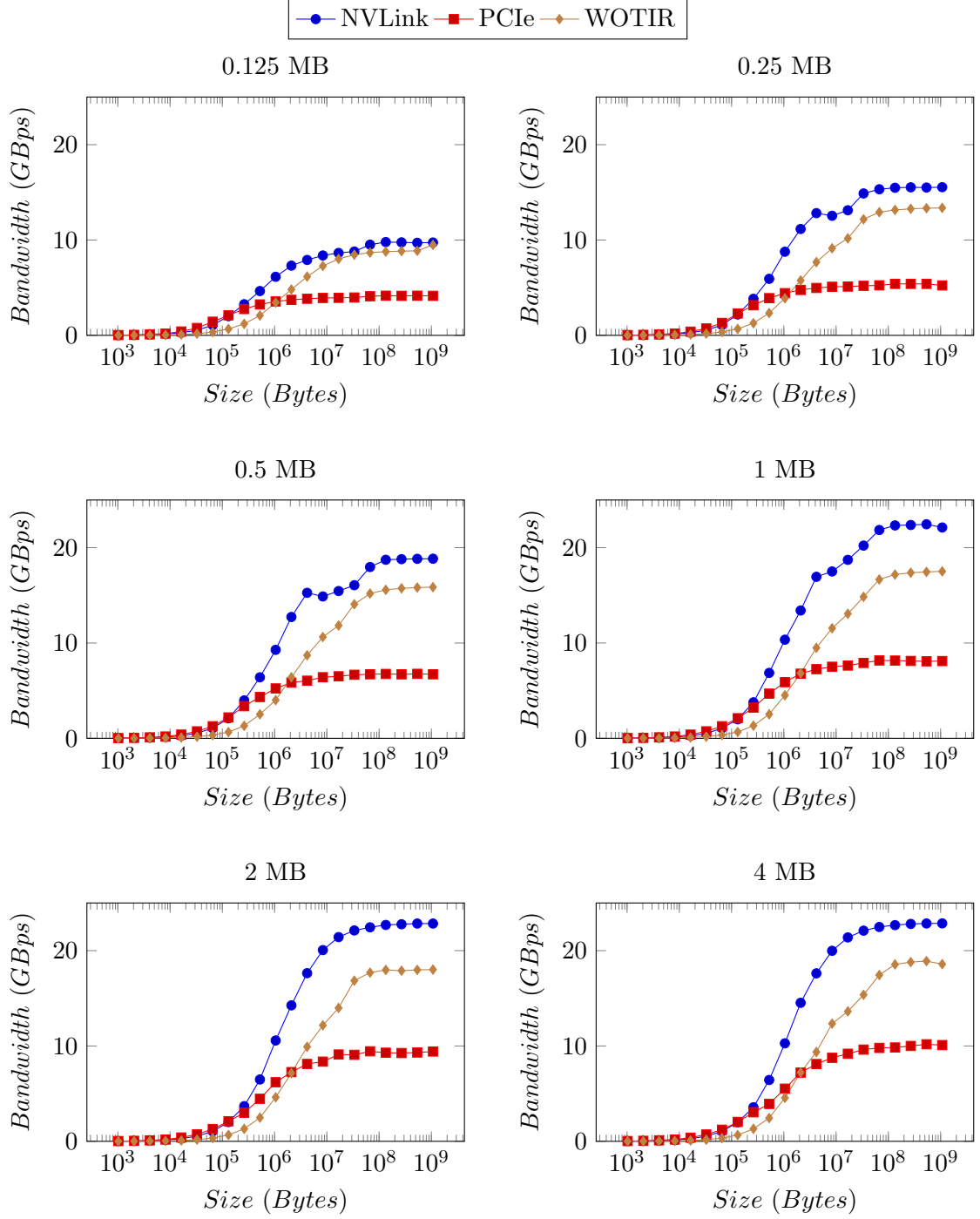


Figure 2.10 Bandwidth analysis for different buffer sizes in NCCL Copy operation.

2.2.3 NVLink Routing

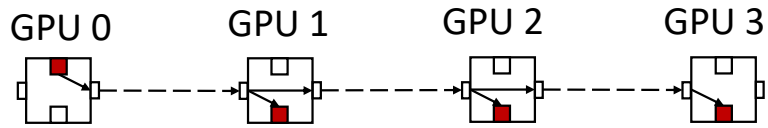
In WOTIR, we implemented a *route-GPU* to transport data without using PCIe. We perform a static lookup of the topology matrix during initialization to determine the ideal route for GPUs without a direct NVLink connection. Between any two non-NVLink connected GPUs, there exist two possible router GPUs. For example, between GPU 2 and 7, possible routing GPU candidates are 3 and 6. We naively select the GPU that would complete an NVLink-only ring in our current implementation.

After identifying a GPU as a *route-GPU*, we launch a routing kernel on a single warp to facilitate the data movement. This kernel performs a new primitive called *forward*. In this primitive, data is directly transferred from the receive buffer to the send buffer. Unlike existing NCCL primitives, the forward primitive does not require the route GPU to have data buffers in the GPU’s memory, only the send/receive buffer of the NCCL communication rings. In order to synchronize the coordination of data transfers in the collective operation, the routing GPU will also have to insert itself into the flagging mechanism that currently exists in NCCL rings.

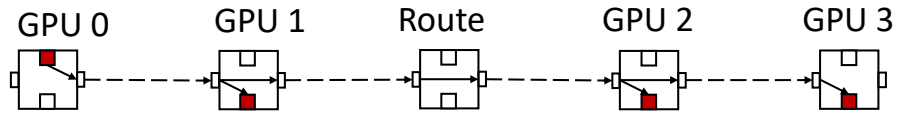
The route kernel persists until the job is complete. Since WOTIR maintains NCCL’s APIs, no change is required by the application to benefit from NVLink routing.

To illustrate the functionality of WOTIR we present the following illustration. For example, if GPUs 0 & 7 were assigned for the application as depicted in Figure 2.2a, the communication between 0 & 7 would take place via PCIe bus due to a lack of NVLink connection between them. However, with WOTIR, GPUs 3 & 4 are assigned as *route-GPUs*

as shown in Figure 2.2b, and data is forwarded through NVLink, eliminating the need for PCIe.

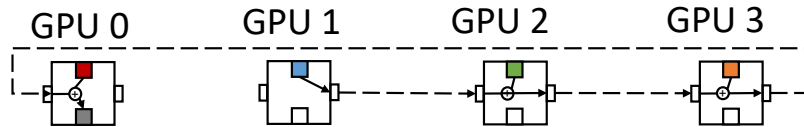


(a) Default implementation

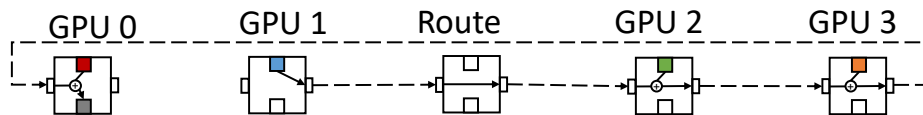


(b) With route

Figure 2.11 NCCL Broadcast

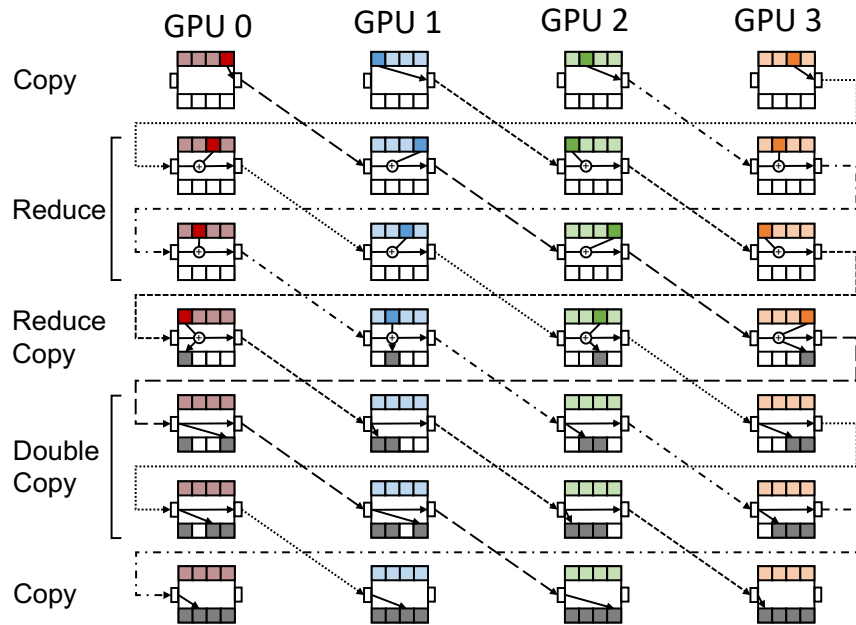


(a) Default implementation

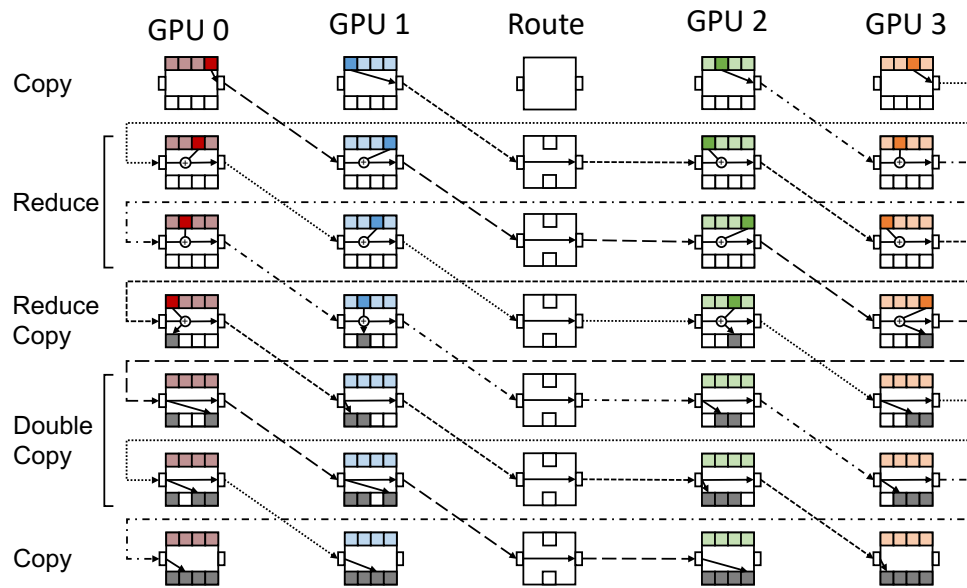


(b) With route

Figure 2.12 NCCL Reduce



(a) Default implementation



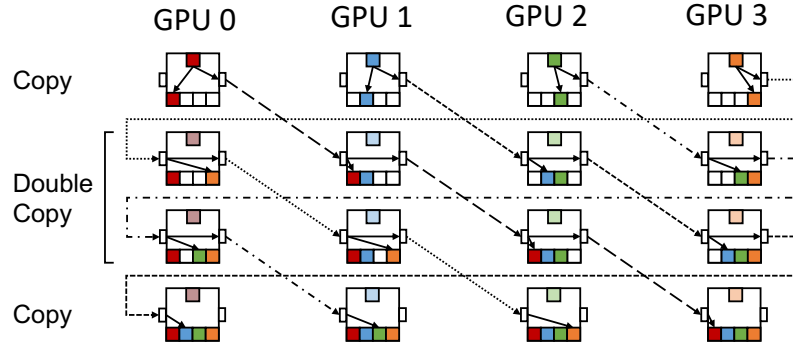
(b) With route

Figure 2.13 NCCL All-Reduce

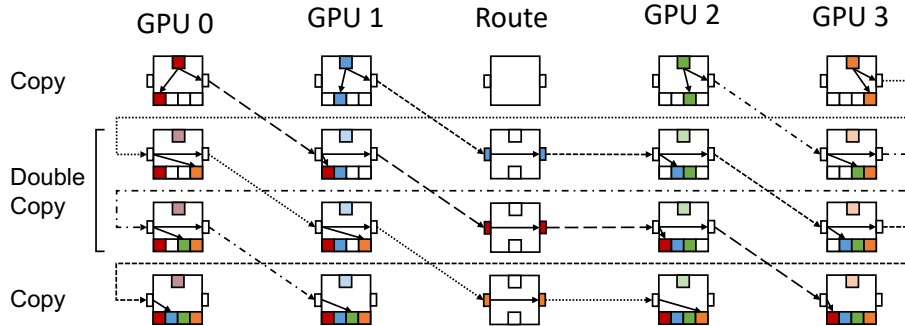
2.2.4 Collective Communication Operations

NCCL supports the following popular collective communication operations:

1. **Broadcast** – Send data from one GPU to many destination GPUs. Figure 2.11 describes the implementation of the broadcast algorithm in NCCL. In the default implementation shown in Figure 2.11a, GPU 0 consists of the source data that has to be broadcasted to GPUs 1, 2, and 3. In this algorithm, the participating GPUs are arranged in a chain topology starting with the source GPU in the front. The first step involves source GPU (0) performing a *Copy* primitive to copy the data from



(a) Default implementation



(b) With route

Figure 2.14 NCCL All-Gather

input data region to the send buffer. All the remaining GPUs except the last GPU in the chain performs a *Double Copy* primitive. The last GPU (3) performs a *Copy* primitive to copy the data in receive buffer to the output data region. These operations are looped when working on large data sizes. When a GPU-route is involved in the chain as shown in Figure 2.11b, the routing GPU in the chain performs the *Forward* primitive.

2. **Reduce** – Reduce the data across many GPUs and store the reduced result in one destination GPU. The operation is similar to Broadcast. However, in this case, the

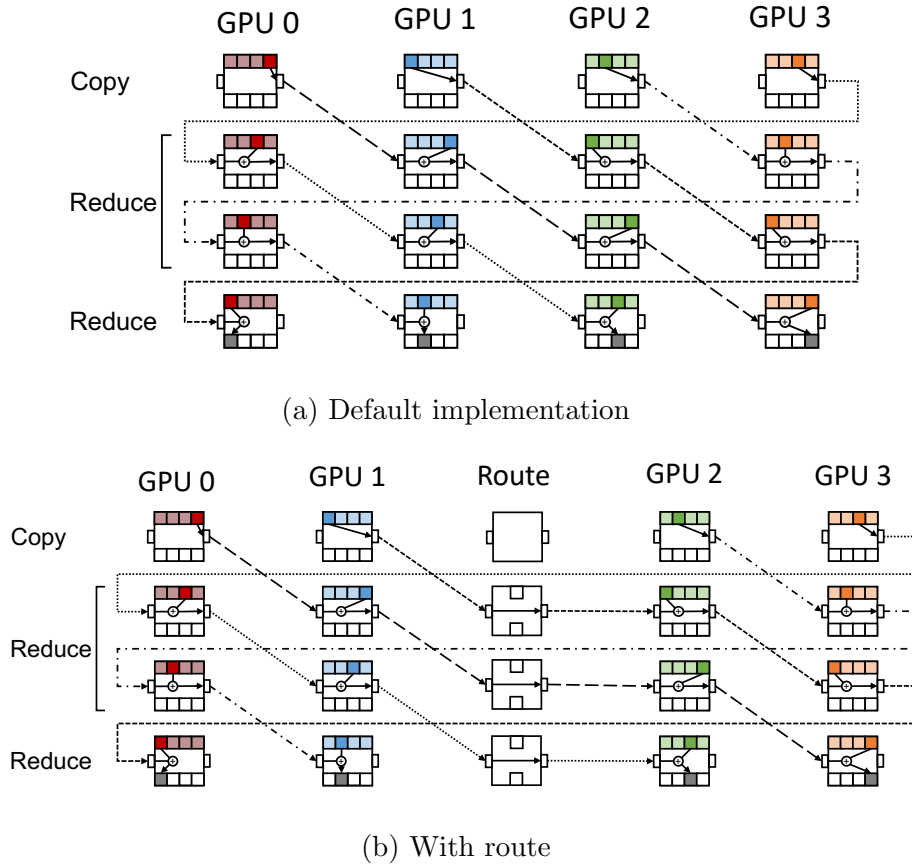


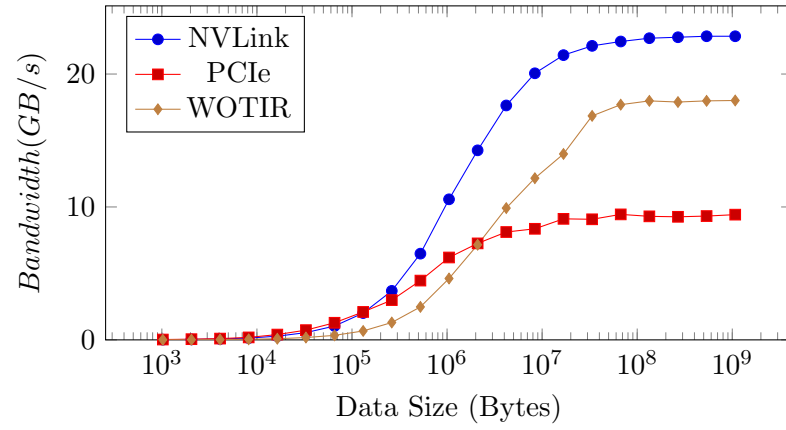
Figure 2.15 NCCL Reduce-Scatter

destination GPU where the result is expected to be stored is placed at the last of the chain as shown in Figures 2.12a and 2.12b.

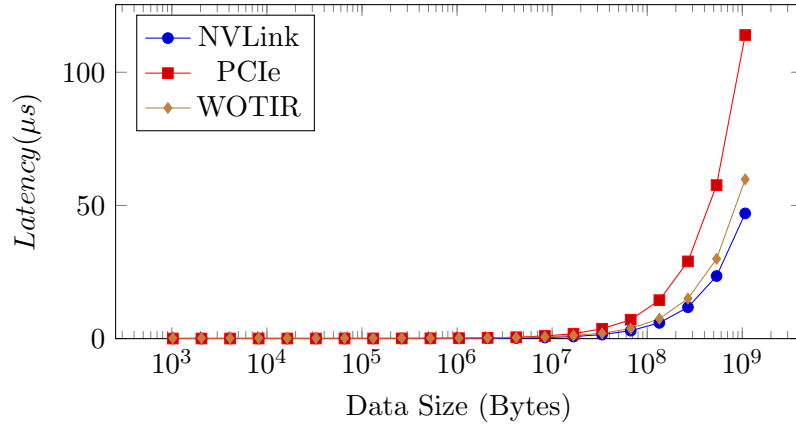
3. **All Reduce** – Reduce the data across many GPUs and place a copy of the reduced result in every GPU. This Collective Communication operation involves multiple steps involving all the NCCL primitives described in Section 2.2.1 as depicted in Figure 2.13. If there are N participating GPUs, the All Reduce operation starts with a copy primitive followed by $N - 2$ *Reduce* primitive operations, a *Reduce Copy*, $N - 2$ Double Copy operations, and finally a copy operation respectively. The total number of steps required to complete We should observe that when a routing GPU is added to the NCCL All Reduce, the number of steps remains the same.
4. **All Gather** – Gather data distributed across participating GPUs into a contiguous array copy on all GPUs. The operation of All Gather involves multiple steps using *Copy* and *Double Copy* primitives as shown Figure 2.14. Just like All Reduce, All Gather with routing can be performed without needing any additional steps.
5. **Reduce Scatter** – Reduce the data across participating GPUs and distribute the final result to all the participating GPUs. As shown in Figure 2.15, the collective communication operation uses *Copy* and *Reduce* primitives to implement the Reduce Scatter algorithm.

2.3 Evaluation

We perform our evaluation on NVIDIA DGX-1 V-100 system with 8 V-100 GPUs with global memory of 16 GB each. We evaluate our work at two levels: memory copy performance, and execution time improvement.



(a) Bandwidth



(b) Latency

Figure 2.16 Bandwidth and Latency improvement with WOTIR.

2.3.1 Memory Copy Performance

To demonstrate the benefit of WOTIR, we repeat the memory copy experiment leveraging our framework, and present the results in Figure 2.16. This figure shows the latency and bandwidth transferring data between 2 GPUs using NVLink, PCIe, and WOTIR. The peak bandwidth achieved in WOTIR is nearly 18 GB/s which is 2x peak bandwidth achievable in the same combination of GPUs (0 & 7) with PCIe. WOTIR does however exhibit poor bandwidth utilization for smaller data transfer sizes due to NVLink overheads, a phenomenon showcased in [56]. This degradation could be avoided by extending WOTIR to employ an adaptive communication strategy to ensure performance to be at least equal to PCIe but not worse.

2.3.2 Collective Communication Operations

Figure 2.18 shows the bandwidth and latency characterization of each of these aforementioned NCCL Collective Communication operations with NVLink, PCIe, and WOTIR.

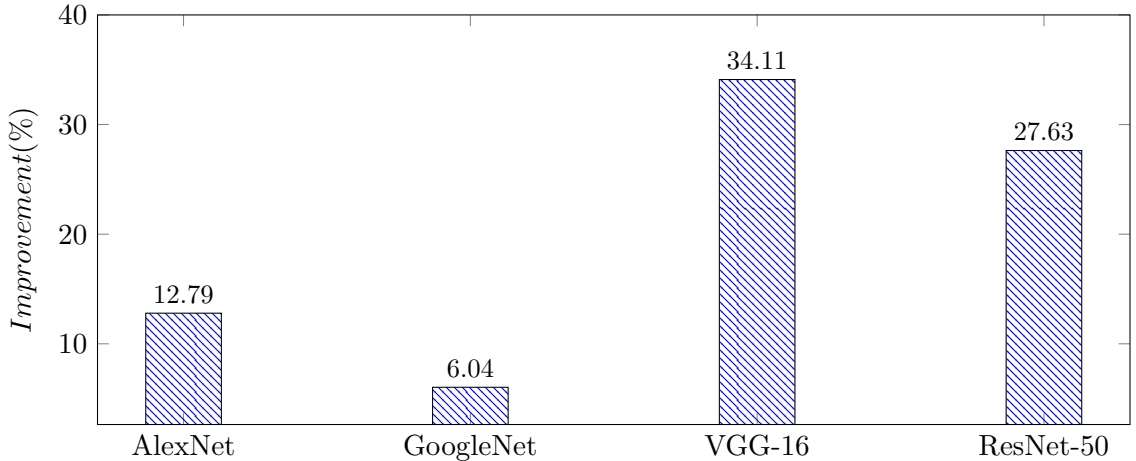


Figure 2.17 Execution time improvement in Caffe of WOTIR compared to PCIe for 3 GPUs

It can be inferred that all the NCCL APIs benefit from WOTIR over PCIe. The peak bandwidths in all the collective communication operations are at least 2x better than PCIe. In *Reduce* and *Reduce Scatter*, however, WOTIR was able to achieve a peak bandwidth close to NVlink. The latency characterization shows that the cost of *Forward* primitive overheads are amortized when the data size is greater than 10^6 Bytes for all operations.

2.3.3 Machine Learning Workloads

One prominent use of multi-GPU environments is Machine Learning (ML). Several ML frameworks have been designed to leverage multiple GPUs including Caffe, TensorFlow, and PyTorch. We demonstrate WOTIR using Caffe, an open-source deep learning framework that can be accelerated with GPUs. We train 4 ML models—AlexNet [55], VGG-16 [100], ResNet-50 [53], and GoogleNet [105] since these models are some of the most heavily used in deep learning. We measure the execution time of training for *max_iterations=10,000* on GPU combinations with NVLink, PCIe, and WOTIR.

Figures 2.17 and 2.19 showcase execution times compared to PCIe and NVLink respectively. In Figure 2.17, we see that WOTIR reduced the execution time by 34% in VGG-16 compared to PCIe. It is also noteworthy that in AlexNet, WOTIR performs better than NVLink. While the device-to-device communication is large enough to utilize the full bandwidth enable by WOTIR, the majority of the communication is host-to-device. This is an artifact of AlexNet’s inter-layer communication prevalent during training. By using two GPUs managed by two separate CPUs, we effectively parallelize the host-to-device communication.

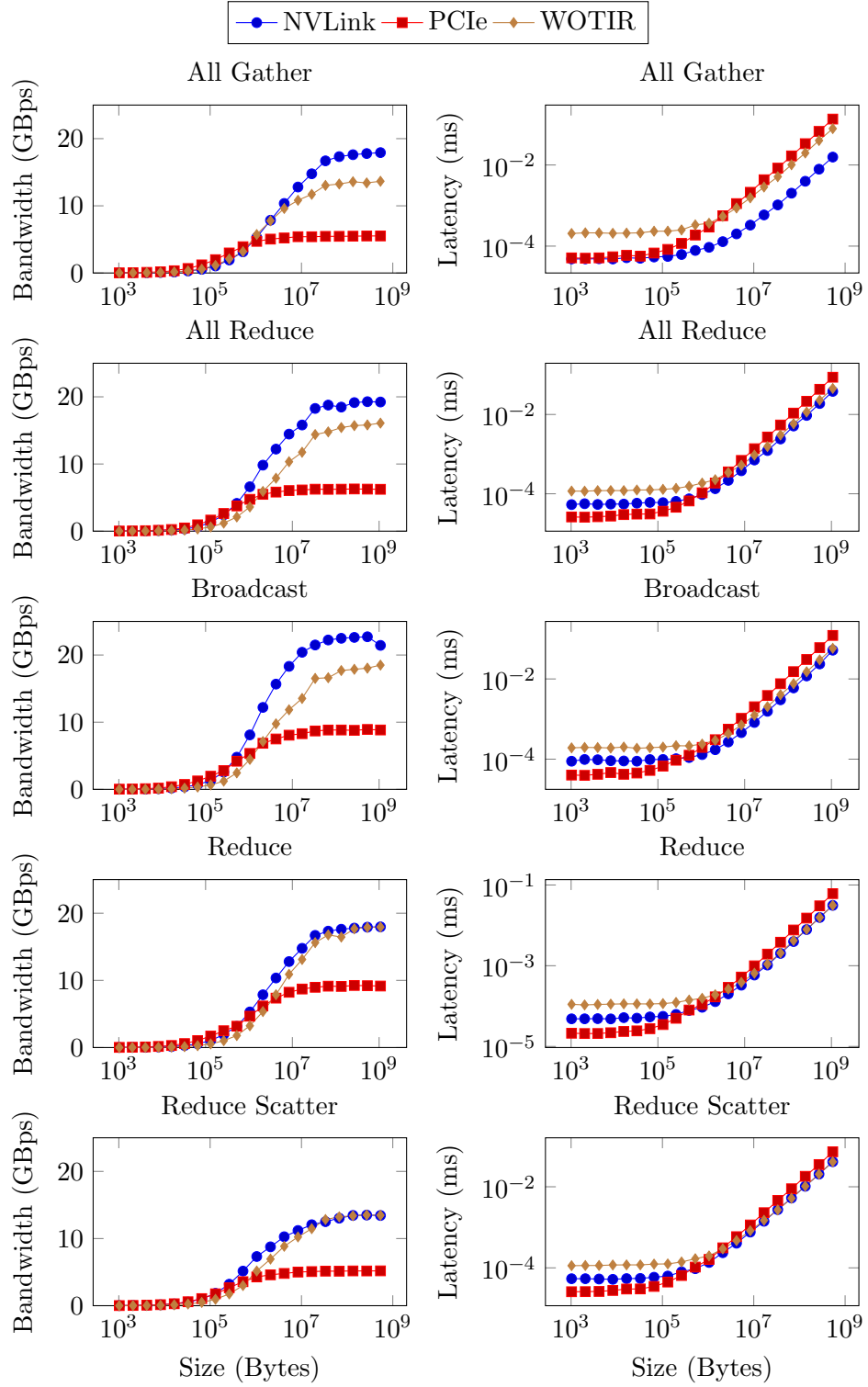
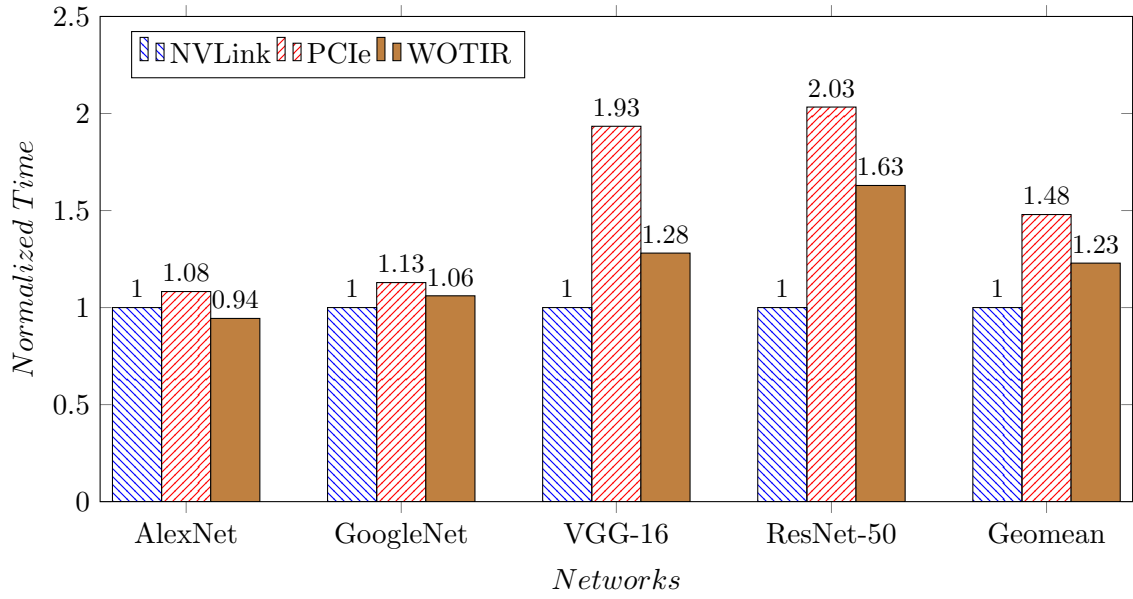
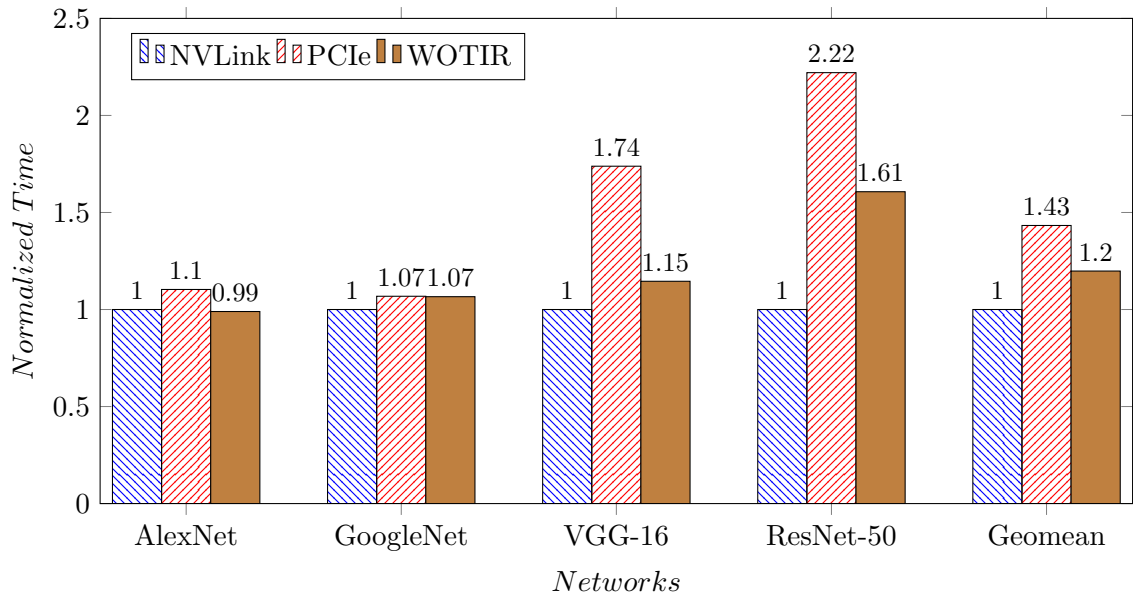


Figure 2.18 Bandwidth (left) and Latency (right) results for NCCL microbenchmarks.

In Figure 2.19, we observe an increase in execution time due to PCIe overheads. The increase is around 10% for AlexNet and GoogleNet, whereas in VGG-16 and ResNet-50, the PCIe overheads increase the execution times up to 120%. Hence we see that VGG-16 and Resnet-50 are the biggest beneficiaries of WOTIR in our evaluation.



(a) 2-GPU Normalized Execution time



(b) 3-GPU Normalized Execution time

Figure 2.19 Execution time of PCIe and WOTIR normalized to NVLink.

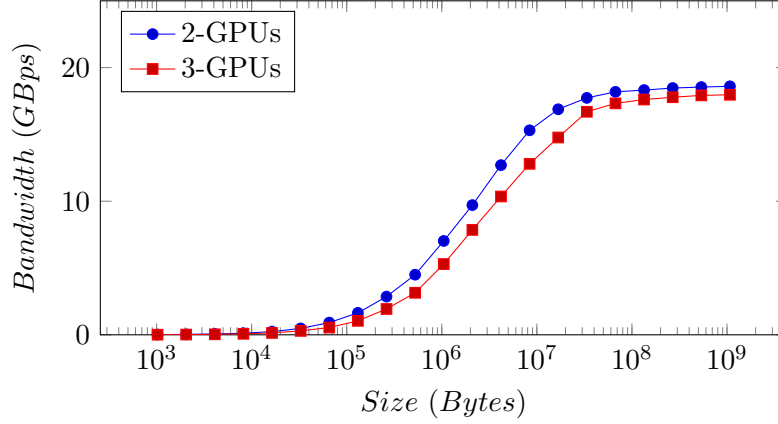


Figure 2.20 All-Reduce Bandwidth scaling with the number of GPUs

It also has to be noted that the peak bandwidth that is achievable with 2 and 3-GPUs with all NVlinks remains nearly the same around 20 GBps as shown in Figure 2.20. However, the bandwidth for the 3-GPU NVlinks lags behind 2-GPUs with NVlink between data sizes of 10^5 and 10^8 . This lag is due to the increase in synchronization primitives with many GPUs.

2.3.4 Interference on route-GPU

To analyze the interference of the routing operation on workloads running on the router GPU, we use BabelStream [21] to stress GPU memory bandwidth while performing multiple memory copies of 1 KB to 1 GB. We ran 1000 iterations of BabelStream to account for experimental variations and present the measured distribution in Table 2.1. Each timing represents one iteration of BabelStream which stresses the memory bandwidth by performing a copy operation of 16 GB to the global device memory. We observed that for

the majority of the case, memory bandwidth interference on the route GPU is minimal. Further work to understand the effect and reduce routing overhead at the tail is necessary.

Furthermore, we observed negligible computational and memory overhead since the routing operation only requires one warp (32 threads) and 2 MB of buffer size respectively. A persistent kernel is launched on the routing GPU to perform *Forward* primitive operations on the routing GPU.

2.4 Related Works

Multi-GPU clusters are increasingly being used for many large-scale applications at a rapid rate, including deep learning and high-performance computing [56, 11]. Recently there have been novel attempts to improve workload placement and inter-GPU communication and network topology.

[121] propose Blink, a set of tree-based broadcast collective communication primitives for multiple GPUs. [27] have addressed the effect of the topology of GPU networks within a node on the performance, and propose a topology-aware GPU assignment to MPI

State	Min (ms)	25th% (ms)	Median (ms)	75th% (ms)	95th% (ms)	Max (ms)
<i>w/o Routing</i>	2.033	2.034	2.035	2.036	2.040	3.140
<i>w/ Routing</i>	2.034	2.035	2.036	2.053	5.674	11.459

Table 2.1 Distribution of BabelStream turnaround time observed for Interference Analysis

tasks to improve communication and performance. [65] present ADAPT, an Open MPI-based collective communication framework that uses a topology-aware communication tree to improve collective operations and maximize the number of communications across the systems at any time. [130] introduce Gandiva, a cluster-scheduling framework for deep learning across multiple GPU clusters. However, the problem of slower PCIe communication remains in these works, which can exacerbate the performance. This chapter addresses and resolves this issue using router GPUs to forward data among the devices.

Gandiva also proposed migration of jobs to GPUs that are freed while the job is running. Since the scheduler constantly checks on GPUs to be freed, the communication problem persists until the job is potentially migrated. Moreover, as mentioned in [82, 110, 52], preemption and migration of GPU kernels have a significant cost. Furthermore, migration policies require GPUs to be more dependent on CPUs to constantly monitor the availability of a better allocation for the running jobs. In contrast, our proposed technique has the potential to provide near-optimal communication performance without the need for migration or preemption.

2.5 Conclusion

In this chapter, we proposed WOTIR, a set of collective communication primitives designed to transfer data among the GPUs in a multi-GPU system through the use of router-GPUs. WOTIR uses NVLinks to forward the data traffic between communicating GPUs that would otherwise only communicate through the PCIe bus due to inefficient

GPU mapping. We have tested this method on a variety of Caffe ML models, achieving an execution time reduction of up to 34% compared to PCIe.

Chapter 3

MAPA: Multi-Accelerator Pattern Allocation Policy for Multi-Tenant GPU Servers

3.1 Introduction

Chapter 2 proposed WOTIR to tolerate the effect of PCIe links in fragmented allocations by re-routing the inter-GPU communications. This chapter attempts to avoid fragmentation by using an intelligent allocation policy.

The never-ending demand for faster computation from data-intensive workloads has driven the growth of multi-accelerator servers. Systems equipped with accelerators, such as General Purpose Processing in Graphical Processing Units (GPGPUs) and Tensor Processing Units (TPU) [48] are increasingly being deployed in shared environments,

such as Cloud, Enterprise, and High-Performance Computing (HPC). These systems are increasingly modular with many accelerators within a single server.

As software and hardware becomes more complex and heterogeneous, new challenges have emerged in the software-hardware stack.

Two major challenges of modern large-scale systems are the need for faster collective communication operations [122, 89] and topology-aware scheduling [8, 131]. Recent works like topology-aware scheduling [8] and Gandiva [131] have motivated the importance of optimal placements to improve the performance of Machine Learning (ML) workloads within multi-GPU environments by efficiently utilizing inter-accelerator interconnection links. Systems such as Nvidia’s DGX-V100, Facebook’s Big-Basin [26], and Amazon’s P3DN [126] have accelerators connected with many different types of interconnection links.

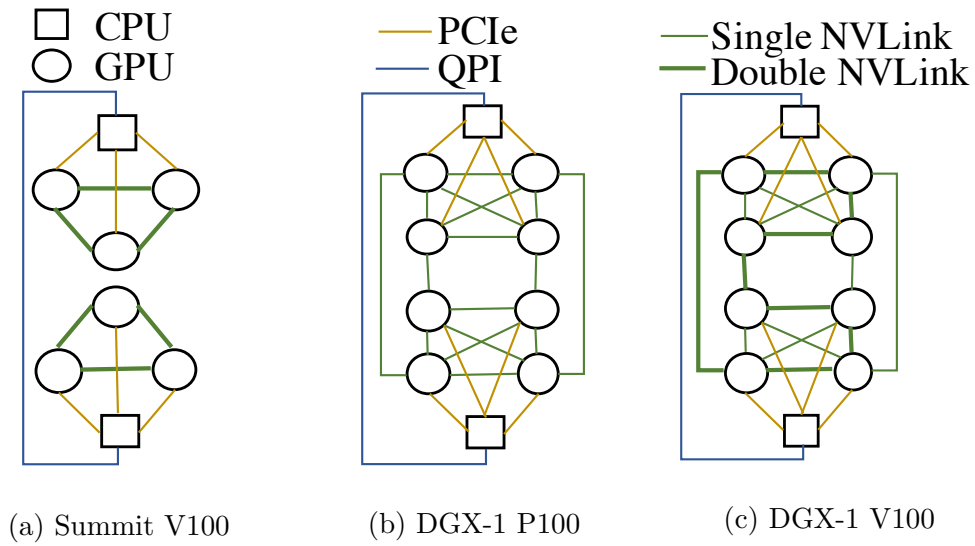
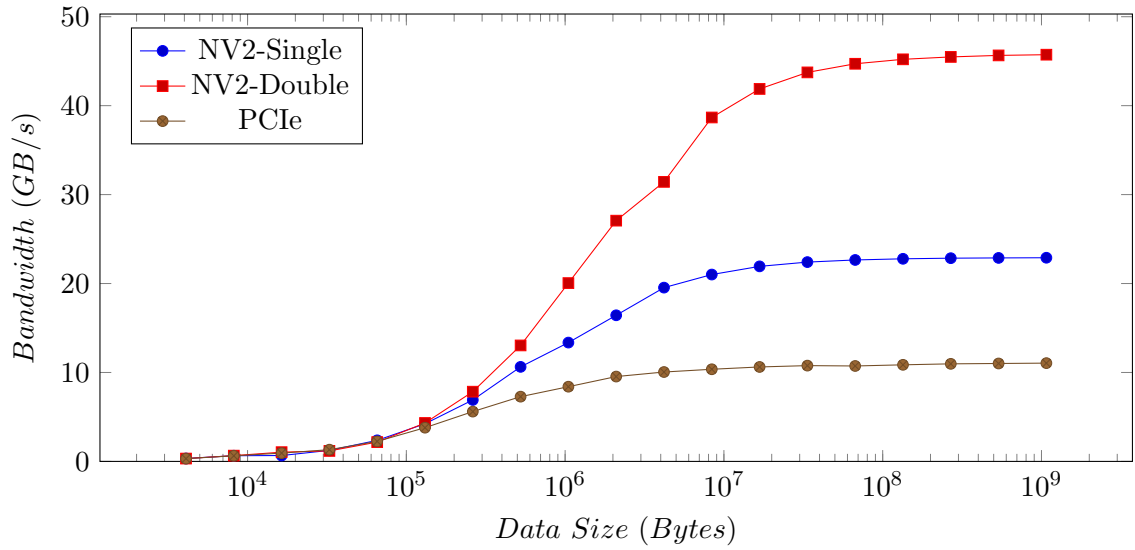
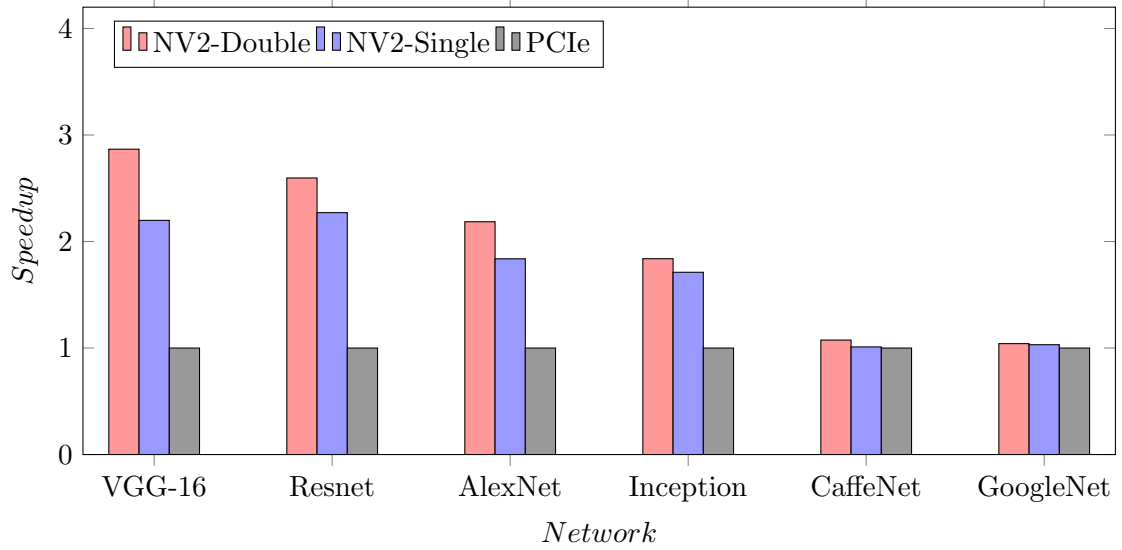


Figure 3.1 Emerging multi-GPU accelerator topologies are increasingly heterogeneous.



(a) Bandwidth characterization



(b) Speedup with different links

Figure 3.2 Nvidia's multi-GPU systems exhibit a variety of interconnects. This figure shows the various links available in DGX-1 Volta. These different links have significantly different bandwidths as well as an impact on applications such as CNN training, where NV2-Single and NV2-Double are Single and Double NVLink-v2 links respectively.

In this chapter, we focus on the challenge of inefficient job allocation in a multi-accelerator environment. These sub-par allocations can lead to a significant slowdown in execution time. These challenges are most prominent in architectures with high heterogeneity in their inter-accelerator inter-connect network (i.e., different number of links with different bandwidths, non-uniform network accesses, etc.) such as NVIDIA’s DGX-1 (Figure 3.1b), Facebook’s Big-Basin systems [26], Amazon’s P3DN [126] and DGX-1-V (Figure 3.1c). Even designs with constant access latency such as the DGX-2 exhibit NUMA effects [57] which can lead to allocation inefficiencies. Furthermore, new accelerator designs such as TPUs [48] and multi-chip accelerators [132] can further fuel the adoption of heterogeneous multi-accelerator designs. As the number of accelerators continues to grow, a smarter job scheduler and resource allocator is needed to fully utilize the underlying hardware and handle the increasing complexity of multi-accelerator workloads.

To this end, we propose a graph pattern matching-based allocation solution called Multi-Accelerator Pattern Allocation (MAPA) to address problems with the allocation of multi-accelerator workloads in multi-accelerator environments. MAPA aims to provide a generic framework applicable to any multi-accelerator environment.

The contributions of this chapter are the following:

- Performance analysis of increasingly heterogeneous accelerator communication links (i.e., PCIe, NVLink) to motivate the need for hardware topology-aware allocation policies.
- MAPA, a graph pattern matching approach for scheduling multi-accelerator workloads on multi-accelerator systems.

- Novel metrics to score matching patterns and predict the effective bandwidth of an allocation.
- Evaluation of MAPA with machine learning training workloads on real-world multi-GPU server.
- Exploration of MAPA on novel hardware topologies at a larger scale and complex non-uniform topologies.

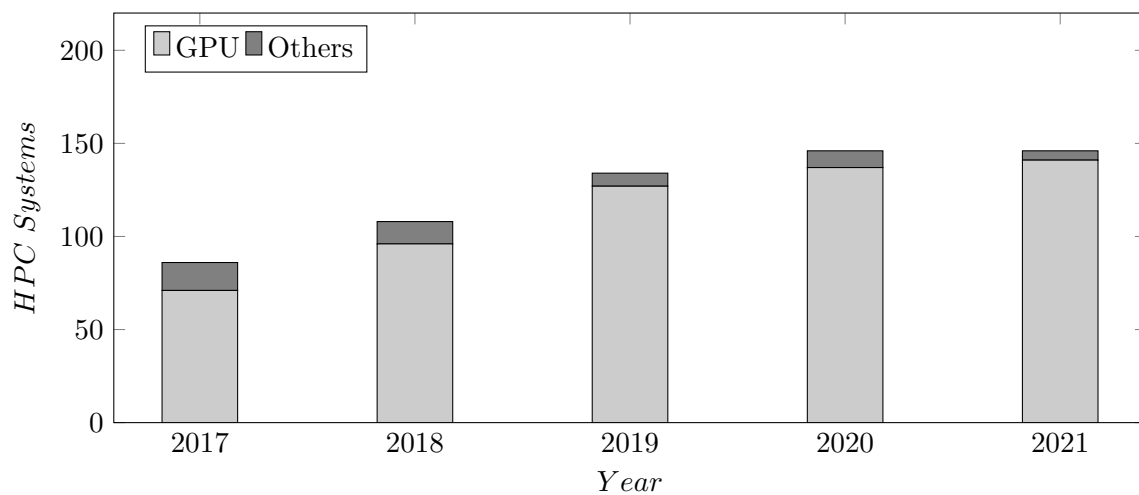
3.2 Motivation

Increasingly more popular cloud [9, 33, 26, 38, 39] and modern HPC systems [37] are accelerator-based and are used to train and deploy complex machine learning workloads across many different fields from proteomics to self-driving vehicles. While these systems primarily employ GPUs, in the future, systems are expected to take advantage of other types of accelerators such as FPGAs or TPUs [48]. The following describes some of the challenges posed by these multi-accelerator architectures.

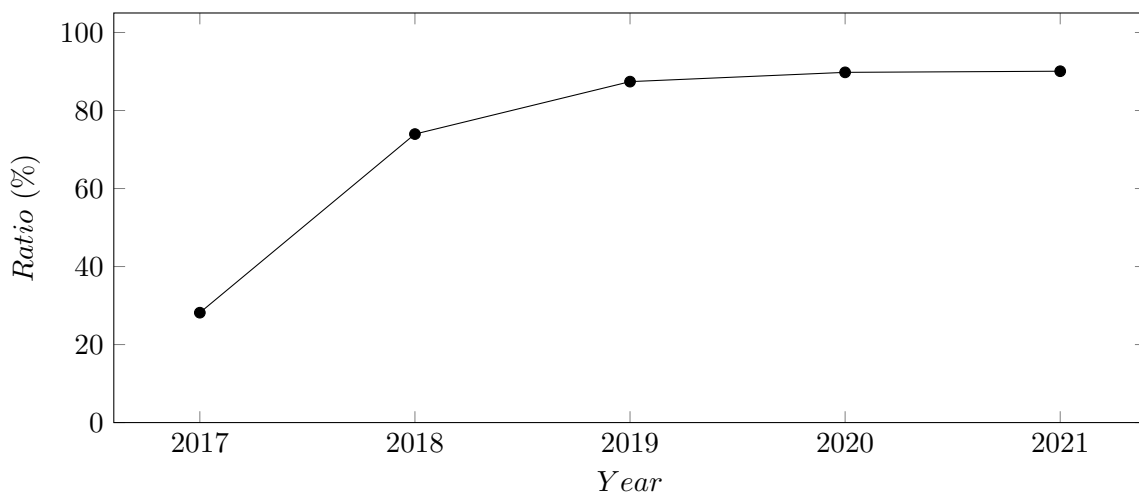
3.2.1 Modern Multi-Accelerator Systems

Characterizing Accelerator Interconnects. Modern multi-GPU servers exhibit a wide range of capabilities when dealing with inter-GPU communication. Table 3.1 lists the types of links used to connect accelerators in these systems and their respective bandwidths.

In systems like Big basin [26], P3DN [126], Summit [37], DGX-1 V100 [39], and DGX-1 P100 [38], accelerators are not uniformly connected. For example, in earlier gener-



(a) Accelerator-based



(b) Heterogeneous interconnects

Figure 3.3 The number of Top500 supercomputers with accelerators are increasing, with GPUs being the most common. The ratio of these GPU HPC systems with heterogeneous interconnects has increased over time and is now dominant.

Link	Bandwidth (GBps)
Single NVlink-v1	20
Single NVlink-v2	25
Double NVlink-v2	50
16-lanes PCIe Gen 3 [79]	12

Table 3.1 Peak Bandwidths per link

ations of the DGX systems, communication can be routed through PCIe links in the case that a direct NVLink cannot be found. Furthermore, in the case of DGX-1 with Volta GPUs (a.k.a. DGX1-V100) and Big-basin, there are some accelerators that are connected via double NVLink connections. Current support for communicating and synchronizing across accelerators include NVidia Collective Communication Libraries (NCCL) [77], AMD’s Radeon Collective Communication Library (RCCL) [10], and Baidu All-Reduce [94].

We observe from Figure 3.3 (a) that supercomputers are increasingly employing discrete GPUs. Figure 3.3 (b) shows the increased presence of heterogeneous interconnects in such systems. Hence, it is important to identify and explore allocation challenges in such computing environments. Additionally, machine learning-based workloads has recently gained attention in the HPC community, with efforts such as Mesh-TensorFlow [99] and Zero [88] which aim to improve the scalability and performance of machine learning on supercomputing systems. Furthermore, there exist numerous works that have attempted to utilize machine learning to accelerate various simulation workloads on HPC systems [23, 84, 86, 29, 16, 49, 50, 124].

GPU Allocation	Link type
(1,5)	Double NVLink
(1,2)	Single NVlink
(1,6)	PCIe

Table 3.2 Allocations depicting different communication links in a DGX-V machine shown in Figure 3.1c.

In Figure 3.2(a), we characterize the communication bandwidth achieved with different links by running the NCCL All-reduce microbenchmark on a DGX-V100 system. This figure demonstrates the peak achievable communication bandwidth of various links across different data transfer sizes. While smaller data transfer sizes achieve lower bandwidth, the relative performance of each link type to each other remains, with double NVLink being the fastest.

In Figure 3.2(b), we show the impact of allocation on popular ML training jobs to GPUs connected by these links. We obtained this by running Caffe workloads across 2 GPUs to utilize the various interconnects. To utilize double NVLink, single NVLink, and PCIe, we allocate to GPUs 1 and 5, 1 and 2, and 1 and 6, respectively. We see that certain networks, such as VGG-16, experience up to 3x execution time speedup using double NVLink compared to PCIe, while other workloads, such as GoogleNet are less impacted. In general, we observe that the allocation of high-bandwidth links is critical for workloads with larger data transfers.

Multi-tenant Multi-Accelerator Servers. It was shown in Philly [45] and Gandiva [131] that jobs running in cloud environments often do not use all of the available

accelerator resources. Thus to ensure the best return on investment in terms of costs and energy, co-location of jobs might be desirable in order to boost utilization. In fact, co-location has already appeared in modern Nvidia GPUs with the Multi-Instance GPU (MIG) [76] feature which enables the GPU accelerator to be shared by up to 7 instances. However, co-location introduces challenges for hard-limit real-time applications, secure applications, or high-performance workloads in general. The effects on performance/security for co-locating jobs require further in-depth exploration to ensure that the loss in these metrics is acceptable for these applications.

3.2.2 Resource fragmentation in multi-tenant servers

One critical challenge caused by multi-tenant servers is that allocated hardware resources can become *fragmented*, that is, the allocated GPUs can be scattered across the entire topology resulting in the loss of high-bandwidth interconnect available to the workload. For example, a 3-GPU allocation will experience fragmentation when allocating GPUs 1, 2, and 5 on the DGX-V system shown in Figure 3.1c. This allocation would require the use of low-bandwidth PCIe that traverses the CPU’s QPI interconnect in order to communicate directly between GPU 2 and GPU 5.

To quantify and highlight this problem, we present Figure 3.4. The x-axis shows the *quality of bandwidth allocation* which we quantify as the *aggregate bandwidth of an allocation* ($BW_{Allocated}$) with respect to the *ideal aggregate bandwidth of an ideal allocation* ($BW_{IdealAllocation}$). For example, for a 3-GPU allocation of GPUs 1, 2, and 5, $BW_{Allocated}$ is 87 GBps (1 PCIe, 1 Single NVLink, 1 Double NVLink). The ideal 3-GPU allocation would be GPUs 1, 3, and 4, where $BW_{IdealAllocation}$ is 125 GBps (1 Single NVlink and 2 Double

NVLinks). Similarly, an ideal 4-GPU allocation would be GPUs 1, 2, 3, and 4 with 175 GBps (1 Single NVlink and 3 Double NVlinks).

We ran 100 machine learning training jobs, each utilizing a different number of GPUs (y-axis), on a DGX-V system using the default baseline scheduling in Nvidia Docker where GPUs are assigned to jobs based on the lowest available GPU IDs (see Section 3.4 for experimental methodology details). The box plot shows the distribution of bandwidth allocation quality.

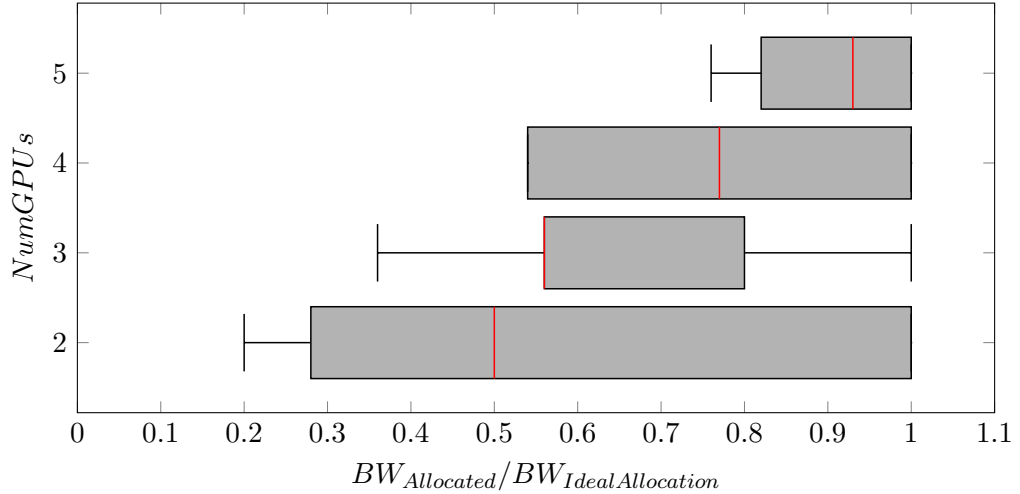
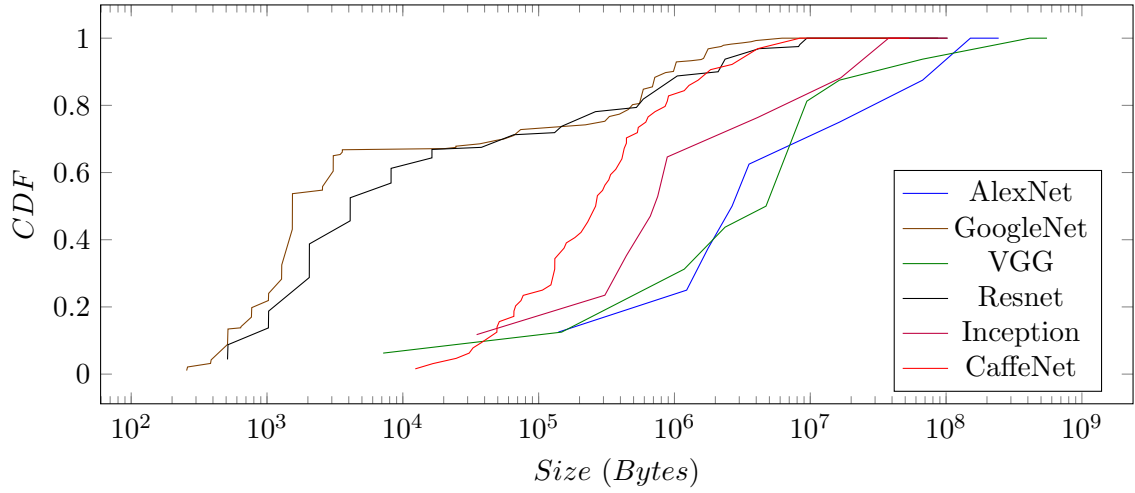


Figure 3.4 Due to fragmentation of GPU allocations, a large portion of GPU jobs have sub-optimal allocated aggregate bandwidth ($BW_{Allocated}$) compared to the aggregate bandwidth of an ideal allocation ($BW_{IdealAllocation}$).



(a) Cumulative distribution of collective communication calls

Network	Communication calls per iter.	Bandwidth Sensitive
AlexNet	80,001	Yes
Inception-v3	2,830,001	Yes
VGG-16	160,001	Yes
Resnet-50	1,600,001	Yes
CaffeNet	84,936	No
GoogleNet	640,001	No

(b) NCCL calls and bandwidth sensitivity

Figure 3.5 Communication Properties of ML workloads

We observe that a large majority of jobs receive suboptimal allocations. It should be noted that smaller jobs with fewer GPUs suffer more due to the potential for being spread out more across the interconnect topology. For example, with 3 GPU jobs, 75% of jobs experience allocations with 20% less bandwidth availability or worse, and 25% of jobs experience allocations with 45% less bandwidth availability or worse.

3.2.3 Understanding Bandwidth sensitivity of ML workloads

As machine learning continues to spread across all aspects of modern life, it is no surprise that ML workloads are the most popular workloads for multi-accelerator systems [45, 131]. While these workloads are characterized to very compute intensive, they have different degrees of sensitivity to the bandwidth provided by the system.

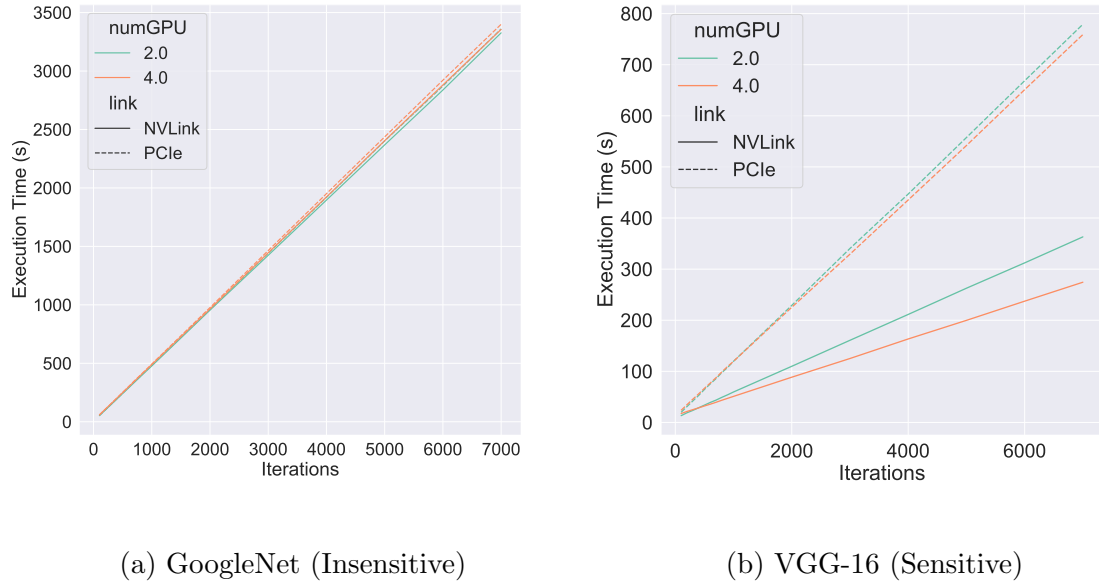


Figure 3.6 Execution Time trends of Bandwidth Sensitive and Insensitive Networks.

Figure 3.5(a) shows the distribution of data sizes that are communicated during the synchronization phase of ML training. Figure 3.5(b) shows the number of collective communication calls per GPU that is employed in training these networks. We can infer from Figure 3.5(a) that Alexnet, VGG, Inception, and CaffeNet involve an average communication data size of at least 10^5 bytes during the synchronization. Similarly, in Figure 3.5(b), we can observe that Inception, Resnet, and GoogleNet involve a large number of communication calls.

It is also to be noted from Figure 3.2(a) that data size has to be larger than 10^5 bytes to make use of the available high-speed links. In GoogleNet, the number of communication calls is higher, however, the average communication size is smaller than 10^5 bytes. In CaffeNet, even though the average size is higher, there are not enough communication calls made to extract the benefit of high-speed links.

Hence, networks such as CaffeNet and GoogleNet are not bandwidth sensitive whereas VGG-16, Inception, Alexnet, and Resnet are. Furthermore, this assertion holds true when increasing the number of GPUs and iterations as shown in Figure 3.6a and Figure 3.6b. Other bandwidth-sensitive networks such as Alexnet, Inception, and Resnet, and bandwidth insensitive networks, such as CaffeNet, follow similar trends to that of VGG and GoogleNet, respectively.

If a bandwidth-sensitive network gets placed on a fragmented allocation, it may slow down ML training jobs by more than 50% as shown in Figure 3.2(b). A solution that could potentially avoid the scenarios like this could improve the overall throughput of the multi-accelerator systems.

In summary, the trends of heterogeneous link topologies and job co-location for multi-GPU servers can leave hardware resources fragmented. Existing job allocation policies are unaware of the hardware diversity leading to misappropriation of bandwidth to jobs. Popular workloads such as ML training can be particularly susceptible to poor allocations. There is a need for a generalized allocation policy that can take into account the growing diversity of inter-accelerator interconnects and multi-accelerator workloads. For the remainder of this work, we will focus our attention on GPUs and ML workloads, however, our approach can be easily generalized to various accelerators and workloads.

3.3 MAPA: Multi-Accelerator Pattern Allocation

The Multi-Accelerator Pattern Allocation (MAPA) framework introduces a generalized solution for the allocation of multi-accelerator workloads on multi-accelerator servers in multi-tenant (shared) environments such as cloud/enterprise data centers, virtualized environments, and shared high-performance computing facilities. Figure 3.7 shows an overview of MAPA. Multi-accelerator applications and multi-accelerator servers are abstracted as smaller application graphs and larger hardware graphs, respectively. The application graphs capture the compute accelerator requirement and inter-accelerator communication topology of the workload, while the hardware graph captures the multi-accelerator system topology. In order to account for fragmentation and application bandwidth sensitivity, allocation decisions must consider the inter-accelerator communication properties of both the application and hardware. To solve this, we take a graph pattern matching approach where we mine the larger hardware graph (i.e., the data graph) for the smaller application graph (i.e., the

pattern graph). Given a set of possible matches, we then assign a score to each pattern match to quantify the quality of each allocation and then select an allocation pattern using our proposed policy. In the remainder of this section, we will describe in detail each component of MAPA.

3.3.1 Application Topology

To make allocation decisions, MAPA abstracts applications into *application graphs* depicting the communication pattern across GPUs. In an application graph, vertices represent an accelerator compute resource (i.e. GPU) and the edges indicate communication between accelerators, as illustrated in Figure 3.8. This application topology graph represents a summary of the application’s communication pattern. While an application’s communication pattern may vary over time, we cannot dynamically reallocate the hardware resource at runtime due to limited support for hardware preemption and the overhead of migration. Thus, we utilize a fixed application topology graph for allocation decisions.

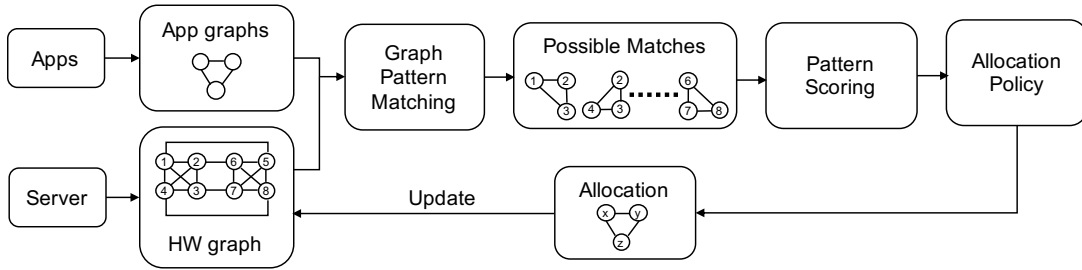


Figure 3.7 Overview of the Multi-Accelerator Pattern Allocation(MAPA) system

Application communication patterns can be manually specified by the programmer or can be automatically extracted through program analysis or profiling [108, 127, 27, 25]. We will outline how each can be performed in the remainder of this subsection.

Source code analysis: Multi-GPU communication is typically coordinated through well-defined APIs. Examples include the NCCL library for collective communications and `cudaMemcpyPeer()` (which explicitly passes the source and destination device) for peer-to-peer communication. By identifying these API calls, communication patterns can be identified through a source code analysis. Figure 3.9a illustrates this through a code sample from Caffe which performs the training operation of a layer. In this example, a collective all reduce is performed with `ncclAllReduce()` before the performing the layer’s training computation in `caffe_gpu_scal()`.

NCCL handles collective communications by building rings or trees and utilizes them depending on the data transfer size that is required by the application. Figure 3.8 shows potential application graphs for a 5-GPU allocation utilizing the NCCL library. Therefore, a 5-GPU application can have varying application topologies depending on the API that is used. Since the communication pattern can be identified based on the NCCL

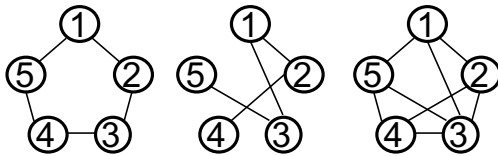


Figure 3.8 Example application topology for 5-GPU workload utilizing NCCL collective communication for inter-GPU communication. Application topologies can be a ring (left), tree (middle), or a combination of both (right).

API, we can build an application topology graph by combining the graph of all NCCL API calls used in the program.

Besides NCCL and CUDA APIs, multi-GPU communication can also occur through MPI. For example, many HPC applications pair a single MPI rank to a single GPU and use MPI calls to communicate across ranks. With CUDA-aware MPI [12], these GPU-GPU communications can be handled directly through NVLink without going through the host. While source code analysis of MPI calls can explicitly identify the communication pattern,

```
void run(int layer) {
    ...
    cudaMemcpy(Y, X, sizeof(Dtype) * N, cudaMemcpyDefault);
    ...
    ncclAllReduce(blobs[0]->mutable_gpu_diff(),
                  blobs[0]->mutable_gpu_diff(),
                  size,
                  nccl::dataType<Dtype>::type,
                  ncclSum, comm_, stream_);
    caffe_gpu_scal(size, (Dtype) 1.0 / Caffe::solver_count(),
                  blobs[0]->mutable_gpu_diff(), stream_);
    ...
}
```

(a) Sample multi-GPU CUDA program using NCCL.

```
$ nvidia-smi nvlink -g 0 -i 5
GPU 5: Tesla P100-SXM2-16GB (UUID: GPU-e00421d4-7649-f32e-c405-335c546b3e2c)
Link 0: Rx0: 5242973340 KBytes, Tx0: 6417636280 KBytes
Link 1: Rx0: 0 KBytes, Tx0: 0 KBytes
Link 2: Rx0: 0 KBytes, Tx0: 0 KBytes
Link 3: Rx0: 0 KBytes, Tx0: 0 KBytes
$ nvidia-smi nvlink -g 0 -i 6
GPU 6: Tesla P100-SXM2-16GB (UUID: GPU-8d3669aa-dbf3-9df8-dafb-edc9861dc344)
Link 0: Rx0: 0 KBytes, Tx0: 0 KBytes
Link 1: Rx0: 0 KBytes, Tx0: 0 KBytes
Link 2: Rx0: 6417636280 KBytes, Tx0: 5242973340 KBytes
Link 3: Rx0: 0 KBytes, Tx0: 0 KBytes
```

(b) Sample NVlink traffic profiling.

Figure 3.9 Examples of identifying application topology through source code analysis and runtime profiling.

many recent works have aimed to automatically identify MPI application topologies [27, 25], or automatically identify communication through compiler-assisted skeletons [108, 127].

Runtime profiling: Runtime profiling of multi-GPU workloads can identify an application’s communication pattern through the monitoring of interconnect traffic over PCIe and NVLink. For example, tools such as `nvidia-smi` track the amount of traffic sent over each NVlink. Figure 3.9b shows an example output for GPU 5 and 6. We can identify that these GPUs are directly connected by Link 0 of GPU 5 and Link 2 of GPU 6. Therefore, at runtime we can monitor the various interconnects to identify any inter-GPU communication between any given pair of GPUs to construct the application topology.

Runtime profiling is especially beneficial when a multi-GPU program has a complex and dynamic communication pattern that is implicit (i.e., Unified Memory) and cannot be easily identified through source code analysis. In these scenarios, instead of conservatively assuming a fully connected application topology, runtime profiling allows us to identify a more representative communication pattern enabling higher-quality allocations.

3.3.2 Hardware Topology

In order to find an allocation, MAPA aims to find a pattern (the application graph) in the larger graph representing the server hardware resource.

In the hardware graph, the vertices represent the compute accelerators, and edges are used to indicate the hardware links available on the server. While the underlying system can have multiple paths (e.g. both an NVLink and PCIe) between two accelerators, edges are labeled with the highest available link bandwidth. For simplicity, we assume the hardware graph to be a fully connected graph as there always exists a path to each accelerator through

the host. For example, if two GPUs are directly connected with double NVLink-V2, then the edge will be labeled with 50. If two GPUs have no NVLink connectivity, then it will be labeled with the PCIe bandwidth of 12. The hardware graphs can be automatically extracted from existing tools, such as *nvidia-smi*, which describes how the accelerators and Compute Units are connected to each other.

Note that our current approach only includes accelerators as vertices and not CPUs. We can potentially extend our approach to also include CPUs in both the application and hardware graph to account for CPU-GPU effects, such as potential NUMA effects. However, the goal of this work is to demonstrate the benefit of improving inter-accelerator communication and thus leave CPU-centric research for future explorations. Another challenge for the hardware topology representation is virtualized accelerators (e.g., Nvidia Multi-Instance GPU or AMD MxGPU) where jobs can be allocated to a virtual device and where inter-accelerator interconnects can be shared between multiple jobs. A potential solution to address this is to label the vertices (which represents a physical device) with the number of physical resources available and then account for resource usage as resources are allocated to jobs and for the potential interference of the inter-accelerator interconnects.

3.3.3 Pattern Matching

To do the application to hardware graph pattern mining, we define a graph g which contains a set of vertices $V(g)$ and labeled edges $E(g)$, a subgraph s of g which containing a subset of edges in g and their endpoints. Given a hardware graph G and the application pattern graph P , we aim to find a match M which is a subgraph of G that is isomorphic to P .

Isomorphic is defined when there is a one-to-one mapping between the set of vertices in the application pattern graph $V(P)$ and the matching pattern graph $V(M)$ such that adjacent vertices in P are also adjacent in M with their corresponding vertices. This can be formulated as a *subgraph isomorphism* (or subgraph matching) problem [80]. Several well-known algorithms exist for solving this problem, such as Ullmann’s algorithm [118, 117], VF2 [80], and VF3 [17]. Since the goal of this chapter is not in proposing a novel subgraph matching algorithm, we choose to utilize existing graph mining systems to implement MAPA’s pattern matching stage. Many general-purpose graph mining systems have been proposed, such as Arabesque [113], AutoMine [66], and Peregrine [42]. Specifically, Peregrine is a state-of-the-art fully pattern-aware graph mining system and pattern-aware programming model to create pattern-aware mining programs. Thus, we implement our pattern matching stage with Peregrine which takes our application pattern graph and hardware graph as input, and all matching subgraph patterns as outputs.

This pattern matching scheme assumes one-to-one mapping between GPU applications and GPU hardware. Many-to-one mapping, where multiple applications can map to the same GPU hardware, is currently emerging. For example, GPUs can be virtualized for multi-tenancy [97] or GPUs can be hardware-partitioned into multiple GPUs (Nvidia multi-instance GPU). Identifying many-to-one mapping is non-trivial and is outside the scope of this work. However, MAPA can potentially support many-to-one mapping by representing virtual GPUs as separate nodes in the hardware graph, or by labeling the nodes of the application/hardware graph with resource requirements/availability (threads, register, NVLink, etc.). This would require label-aware pattern matching and potentially

partitioning of the application graph to fit into the available hardware resources or utilize more complex many-to-one scheduling policies, such as in [97].

3.3.4 Pattern Scoring

Given the set of matching patterns from the previous stage, MAPA then must select the best pattern for allocation. MAPA aims to assign a *score* to each matching pattern which predicts which allocation will result in the most performance. To this end, we need to first answer *How do we score each pattern match?*

Pattern scoring metrics

To find a suitable pattern scoring metric, we explore two proposed metrics called *Aggregated Bandwidth (AggBW)* and *Effective Bandwidth (EffBW)*.

Aggregated Bandwidth: We define *Aggregated Bandwidth (AggBW)* as the total allocated bandwidth in the matching pattern M that is used by the application pattern

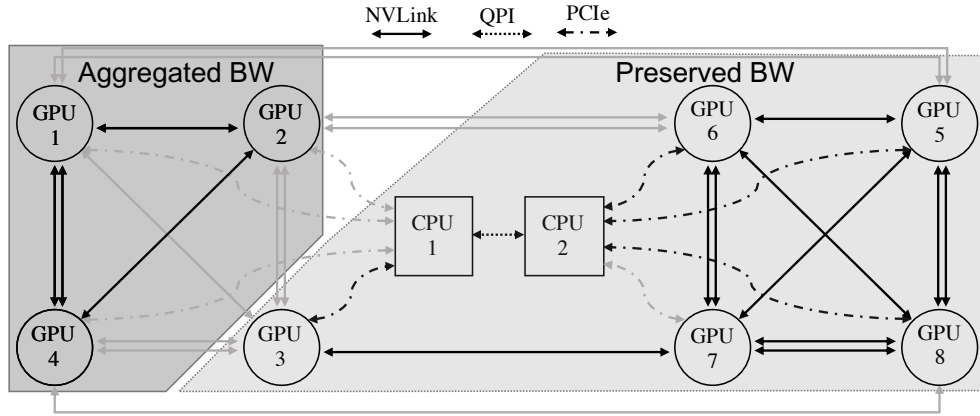
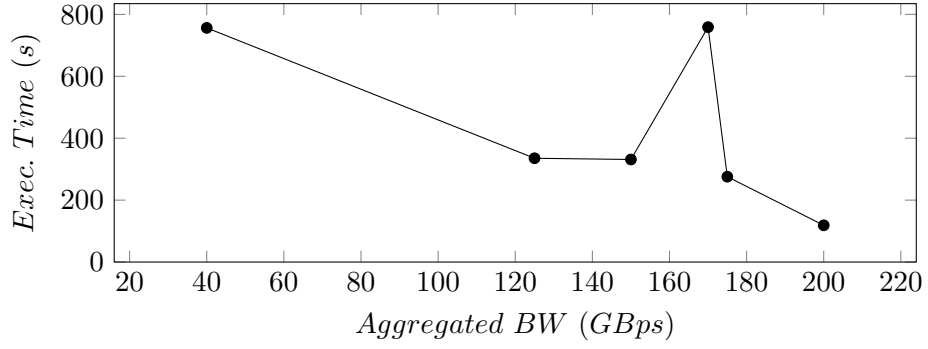
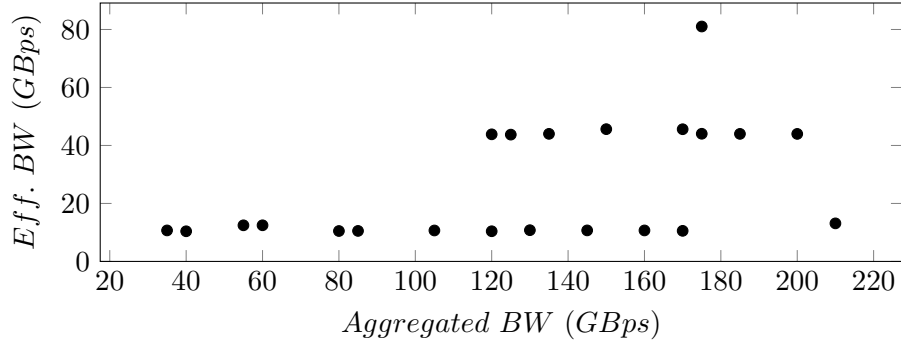


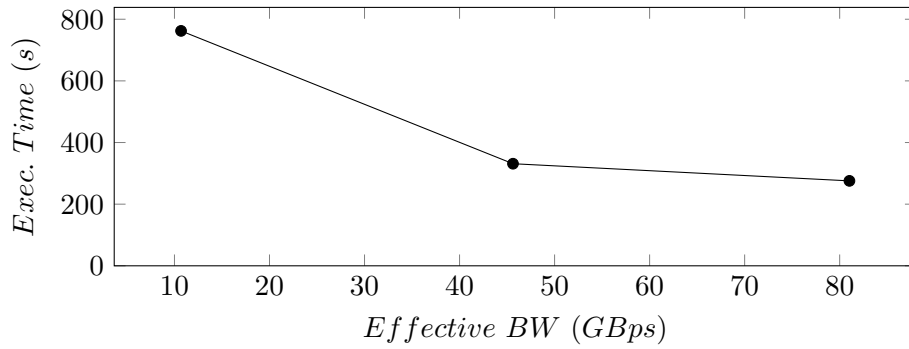
Figure 3.10 Illustrative example showing bandwidths used for Aggregated Bandwidth score calculation (left) and Preserved Bandwidth calculation (right) given an allocation [1, 2, 4]



(a) VGG-16 training Execution Time



(b) Aggregated vs Effective Bandwidth



(c) VGG-16 training Execution Time

Figure 3.11 Evaluating pattern scoring metrics. (a) *AggBW* does not correlate well with execution time. (b) This is due to *AggBW* not correlating well with the effective achievable bandwidth of an allocation. (c) *EffBW* correlates well with execution time.

graph P . Since the application pattern graph P is isomorphic to the matching pattern M , we know that $V(P) = V(M)$. However, the application's communication pattern may not use all of the available hardware interconnects that is allocated to it. That is, the set of edges in the application pattern may be a subset of the edges in the matching pattern, $E(P) \subseteq E(M)$. Therefore, the set of edges that are actually used by the application pattern in the matching pattern is denoted as $E(P) \cap E(M)$. Recall that the edges e of the hardware graph $E(G)$, and therefore the edges of the matching pattern $E(P)$, are weighted $w(e)$ with the highest available bandwidth between the two accelerator devices corresponding to the vertices of the graph. Therefore, we formally define *Aggregated Bandwidth* as shown in Equation 3.1.

$$AggBW = \sum_{e \in (E(P) \cap E(M))} w(e), \quad (3.1)$$

where $E(P) \cap E(M)$ represents the allocated interconnect in the matching pattern M that is used by the application P , e represents a used interconnect, and $w(e)$ represents the bandwidth of the interconnect. Specifically, $AggBW$ takes into account the application's communication pattern in order to quantify the amount of *usable* communication bandwidth that was allocated to it.

To illustrate $AggBW$, Figure 3.10 shows a possible allocation of a 3-GPU tasks that is mapped to GPU 1, 2, 4. Therefore, the $AggBW$ is the sum of the bandwidth of the interconnects between GPUs 1 and 2, 2 and 4, and, 1 and 4.

Effective Bandwidth: We define *Effective Bandwidth* ($EffBW$) as the peak achievable bandwidth for a given allocation. This metric is measured by running mi-

crobenchmarks to measure the peak effective real-world bandwidth across multiple links that is achievable for a given allocation. In our experiments, we use the NCCL All-reduce microbenchmark to determine the peak effective bandwidth. We selected this benchmark because the All-reduce collective communication pattern is the most used and has the greatest impact to overall execution time. The effective bandwidth that we observe with different allocations is dependent on the number of links and the type of links (i.e. double NVLink, single NVLink, and/or PCIe).

Evaluating Metrics

Now let us evaluate the two metrics, Aggregated and Effective Bandwidths. We ran a multi-GPU training workload, VGG-16, with various 4-GPU and 5-GPU jobs and potential matching allocations. We measured the execution time of the workload, and the *AggBW* and *EffBW* of the allocation. Figure 3.11(a) shows that *AggBW* does not correlate well with the workload’s execution time. For example, an allocation with *AggBW* of 170 is much slower than an allocation with *AggBW* of 150. An ideal metric for scoring pattern matches would be correlated and be able to predict a workload’s execution time.

We find that this discrepancy is due to the fact that naively using the aggregated bandwidth *AggBW* does not correlate with the effective bandwidth *EffBW* that is achievable for a given allocation. This is demonstrated in Figure 3.11(b) which is collected using microbenchmarks to measure the effective bandwidth of various allocations ranging from 2-5 GPUs. *Therefore, we find that the execution time of workloads cannot be predicted by naively aggregating the allocated bandwidth. Instead, the execution time of workloads must*

be predicted by the effective bandwidth. Figure 3.11(c) demonstrates this fact by showing that effective bandwidth correlates well with workload execution time.

However, a major challenge of using effective bandwidth as a metric to score matching patterns is that effective bandwidth cannot be trivially obtained given an allocation without microbenchmarking. Therefore, we need to create a model for predicting effective bandwidth.

Predicted Effective Bandwidth

In the previous section, we demonstrated that the execution time is a function of effective bandwidth. Hence, we need to figure a way to predict *EffBW* without having to run the microbenchmarks for a matching pattern. This could be achieved by solving a non-linear polynomial regression model. Here the Effective Bandwidth is related to the number of Double NVLinks (x), Single NVLinks (y), and PCIe links (z) in a given matching pattern M .

$$\begin{aligned}
& \textit{Predicted Effective Bandwidth} = \\
& \theta_1 x + \theta_2 y + \theta_3 z + \theta_4 \frac{1}{x+1} + \theta_5 \frac{1}{y+1} + \theta_6 \frac{1}{z+1} \\
& + \theta_7 xy + \theta_8 yz + \theta_9 zx + \theta_{10} \frac{1}{xy+1} + \theta_{11} \frac{1}{yz+1} + \theta_{12} \frac{1}{zx+1} \\
& + \theta_{13} xyz + \theta_{14} \frac{1}{xyz+1}
\end{aligned} \tag{3.2}$$

To obtain data to train the model, we generate a set of 2, 3, 4, and 5-GPU allocations in a DGX-V machine described in Figure 3.1c. To limit the size of the generated set, we use an exhaustive set of allocations with unique (x, y, z) resulting in a total of 31 samples. Next, we recorded the *EffBW* by running the NCCL microbenchmark as described

previously. Next, we solve equation 3.2 using non-linear polynomial regression and the collected data (corresponding (x, y, z) and the recorded $EffBW$), to learn the relationships between the types of allocated links (x, y, z) and $EffBW$. Through the regression model in equation 3.2, we learn the coefficient θ of the following linear and non-linear features to capture their impact on effective bandwidth – linear (x, y, z) , inverse-linear $(\frac{1}{x+1}, \frac{1}{y+1}, \frac{1}{z+1})$, pairwise (xy, yz, zx) , inverse-pairwise $(\frac{1}{xy+1}, \frac{1}{yz+1}, \frac{1}{zx+1})$, triplet (xyz) , and inverse-triplet $(\frac{1}{xyz+1})$. The values of each of the coefficients are tabulated in table 3.3.

Figure 3.12 shows the predicted versus actual Effective Bandwidths given a (x, y, z) . For this model, the Relative Error, Root Mean Square Error (RMSE), and Mean Absolute Error (MAE) were found to be 0.0709, 1.5153, and 7.0539 respectively. The model shows a strong correlation between the predicted $EffBW$ and the measured $EffBW$ and generalizes well even when the number of GPUs in a job varies. This demonstrates that the Effective Bandwidth is strongly related to the mix of links allocated and not necessarily the amount of aggregate bandwidth allocated. Using equation 3.2 we can now directly utilize $EffBW$ as our pattern scoring metric without the need for microbenchmarking.

Coeff.	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
Value	16.396	4.536	1.556	-20.694	-9.467	7.615	-7.973
Coeff.	θ_8	θ_9	θ_{10}	θ_{11}	θ_{12}	θ_{13}	θ_{14}
Value	12.733	-4.195	-8.413	62.851	27.418	-5.114	-46.973

Table 3.3 Values of Coefficients.

3.3.5 Pattern Selection Allocation Policy

Once the matching patterns are scored, MAPA will then select a matching pattern for allocation. Recall from Section 3.2.3 and Figure 3.5b that certain workloads are bandwidth sensitive while others are bandwidth insensitive. Thus, in order to maximize the overall performance of scheduled jobs, the pattern selection policy must account for (1) the effective bandwidth of an allocation, (2) the bandwidth sensitivity of the job, and (3) avoiding starving future bandwidth sensitive jobs of effective bandwidth. To account for bandwidth sensitivity, MAPA assume that an application's bandwidth sensitivity is known and already annotated. The bandwidth sensitivity of an application can be determined through various means, for example, by profiling execution time vs allocated links as shown in Figure 3.6.

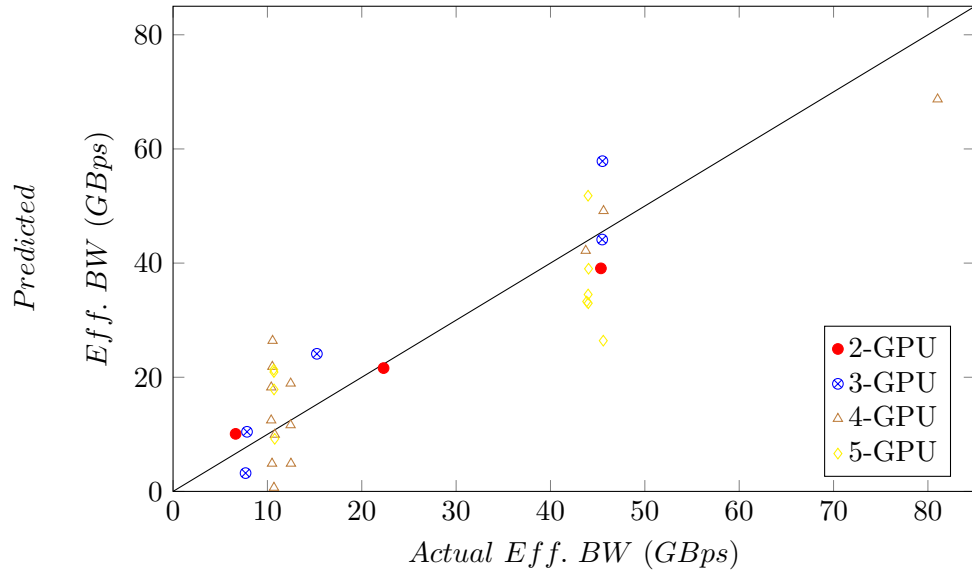


Figure 3.12 Predicted effective bandwidth correlates well with actual effective bandwidth and generalizes across jobs of different sizes.

A novel aspect of MAPA is that when we select an allocation for bandwidth insensitive jobs, we try to *preserve* as much remaining effective bandwidth as possible for future sensitive jobs. This bandwidth preservation scheme will then be able to optimize the allocation of bandwidth-sensitive jobs. In order to quantify the amount of remaining bandwidth that is preserved, we introduce a new metric as follows.

Preserved Bandwidth

We define *Preserved Bandwidth* as the aggregate bandwidth of the *usable links that remain* (preserved) if a pattern match M is allocated on the hardware graph G . The remaining hardware graph is denoted as $G \setminus M$ which is the subgraph of G *induced* by the remaining available accelerator devices $V(G) \setminus V(M)$. In other words, the remaining hardware graph $G \setminus M$ is an *induced subgraph* which is constructed by deleting the pattern match vertices $V(G) \setminus V(M)$ (which allocates the corresponding accelerator devices) and with them all the incident edges (the hardware links that are no longer usable for future allocations). Figure 3.10 illustrates the calculation of preserved bandwidth if GPUs 1, 2, and 4 are allocated. Hence, we formally define *Preserved Bandwidth* in Equation 3.3.

$$\text{Preserved Bandwidth} = \sum_{e \in E(G \setminus M)} w(e) \quad (3.3)$$

Preserve Allocation Policy

We present the Preserve Allocation policy in Algorithm 1. In this policy, we rely on the programmer annotated bandwidth sensitivity (*bwSensitive*), the Preserved Bandwidth (*PreservedBW*), and the Predicted Effective Bandwidth (*EffBW*). If the job to be

allocated is bandwidth insensitive, we allocate the matching pattern that obtains the largest *Preserved Bandwidth*. This means we are preserving the amount of remaining available high-bandwidth links in the hardware graph for bandwidth-sensitive allocations to avoid

Algorithm 1: Preserve Allocation Policy

Result: Allocation

HWgraph hGraph;

AppGraph aGraph;

Allocation alloc = { };

Patterns possiblePatterns = graphPatternMatching (aGraph, hGraph);

if *aGraph* **is** *bwSensitive* **then**

foreach *pattern* **in** *possiblePatterns* **do**

if *EffectiveBW* (*pattern*) > *EffectiveBW* (*alloc*) **then**

alloc = *pattern*;

end

end

end

else

foreach *pattern* **in** *possiblePatterns* **do**

if *PreservedBW* (*pattern*) > *PreservedBW* (*alloc*) **then**

alloc = *pattern*;

end

end

end

potentially starving these jobs. If the job to be allocated is bandwidth sensitive, we allocate the matching pattern with the highest *Predicted Effective Bandwidth*.

3.3.6 State Management

Once a matching pattern is selected for allocation, we then must update the hardware graph G . The hardware graph G is updated whenever there is an allocation (a job is scheduled) and deallocation (a job is finished). Once an allocation is obtained, we update the hardware graph to remove the unavailable vertices and incidental edges. When a job is complete and the hardware resource is deallocated, we update the hardware graph by adding back the vertices and incidental edges that were previously removed.

3.4 Evaluation

To evaluate MAPA, we use a combination of real-world runs and simulation. Specifically, we first evaluate the effectiveness of MAPA and the impact on performance on an NVIDIA DGX-1 V100 machine running on Ubuntu-16.04 with CUDA 11.3 and NCCL-2.10.3. The DGX-1 V100 hardware topology is shown in Figure 3.1c. The MAPA framework is built on top of Peregrine [42], a graph mining engine, which performs subgraph pattern matching. Although MAPA is agnostic to scheduling policies and can be extended to any scheduling policy and can employ reordering. However, in this chapter, we use First-in First-out (FIFO) for scheduling jobs from the queue.

Later in Section 3.5, we will evaluate MAPA on different multi-accelerator topology configurations by simulating the schedulers’ benefit on various representative hardware graphs.

Workloads: In our evaluation, we use a set of Caffe [46] training jobs which makes use of multiple GPUs – AlexNet [55], VGG-16 [100], Resnet-50 [53], Inception [104], GoogleNet [105], and CaffeNet [47]. These neural networks are trained using the Image-Net dataset [22]. Each of the evaluated networks have different compute and communication patterns as discussed in Section 3.2.3. In addition, we use three other non-neural network multi-GPU workloads. They are a parallel simulated annealing algorithm for global optimization (Cusimann) [61], Gaussian Mixture Model (GMM) [61], and a Jacobi solver [74]. Furthermore, previous works [61, 60] have demonstrated that CUSIMANN and GMM have negligible inter-GPU communication during the course of execution. Furthermore, we observed less than 3% execution time improvement with the Jacobi solver. Hence, we classify CUSIMANN, GMM, and Jacobi to be bandwidth insensitive. In this chapter, we focus on the inter-GPU communication aspect when multiple GPUs are employed in a single job.

Jobs configuration: We randomly generated a job file of 300 jobs consisting of a uniform mix of training jobs for machine learning networks as shown in Figure 3.5.

In addition, these jobs are generated with a random number of requested GPUs, from 1 to 5, which follows a uniform distribution. Prior work [45] has shown that the number of request GPUs for multi-GPU jobs in multi-tenant environments tends to be uniformly distributed.

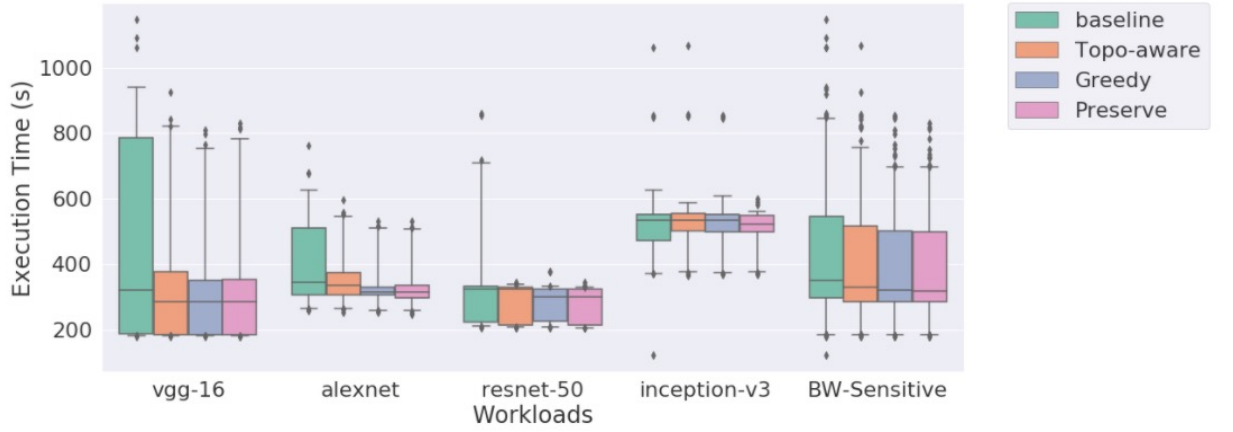
Baseline Scheduling Policies: To evaluate MAPA, we compare the preserve policy against three multi-GPU allocation policies—*Baseline*, the current state-of-the-art scheduling technique *Topo-aware* [8], and a simple greedy policy *Greedy*. The *Baseline* policy simply allocates GPU by ID by selecting the lowest IDs. This is how current GPU allocation is done in existing frameworks such as Nvidia Docker [75]. The *Topo-aware* allocation policy [8] utilizes recursive bi-partitioning to select GPUs for allocation. This scheduler in effect selects GPU allocations under the same PCIe tree (CPU socket). The Greedy allocation policy simply selects a matching pattern with the highest *Aggregated Bandwidth* for allocation.

3.4.1 Evaluation on DGX-V System

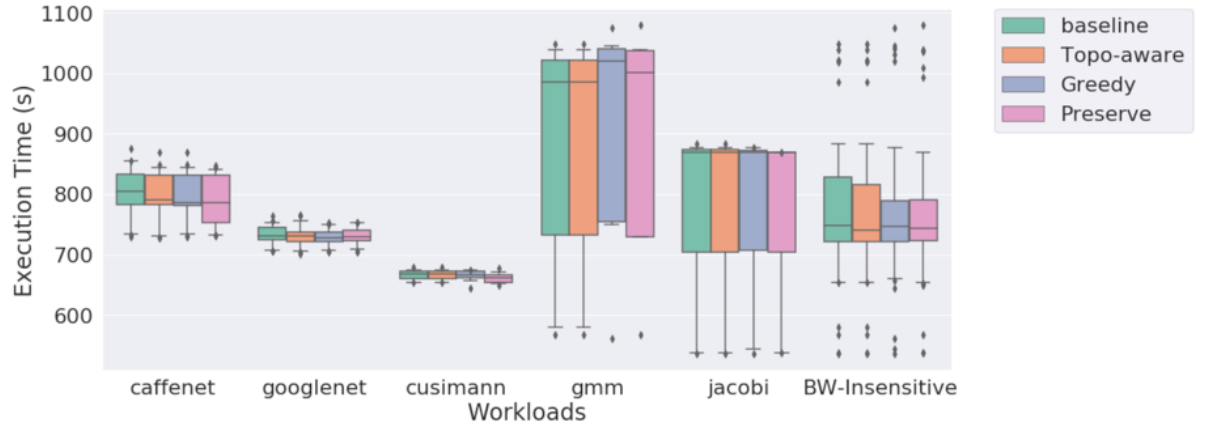
We ran a mix of 300 jobs on the target DGX-1 V100 machine with Baseline, Topo-Aware, Greedy, and Preserve. These jobs are provisioned concurrently if sufficient hardware resources are available to allow multiple jobs to run concurrently. For each job, we record the quality of the allocation using the predicted Effective Bandwidth score and the execution time. Figures 3.13 and 3.14 shows our results, separated by sensitive and insensitive workloads.

Figure 3.13(a) and 3.13(b) shows the execution time of the experiments. Note that when running jobs on a multi-tenant server, not all jobs will experience poor allocation due to fragmentation. Instead, the main point of focus should be the long tail of execution time where workloads that exhibit poor allocation will similarly exhibit poor execution time.

The baseline policy allocates based on the smallest available GPU ID and thus suffers significantly when allocations are fragmented, as demonstrated by the long tails of most bandwidth-sensitive workloads, except Inception. The Topo-aware policy aims to schedule jobs under the same CPU socket which consists of fully inter-connected GPUs. This results in significantly improved tail execution times, most notably in VGG and Alexnet



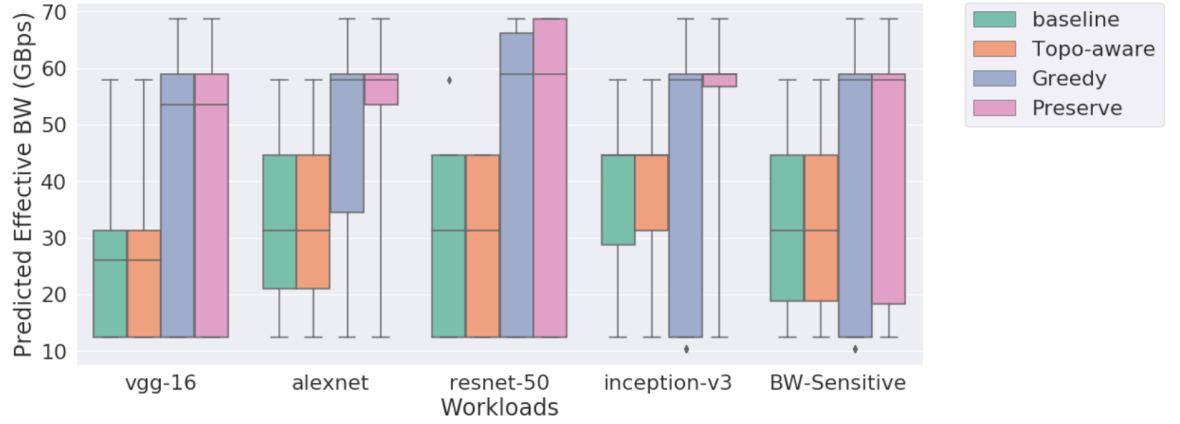
(a) Execution time of bandwidth sensitive jobs



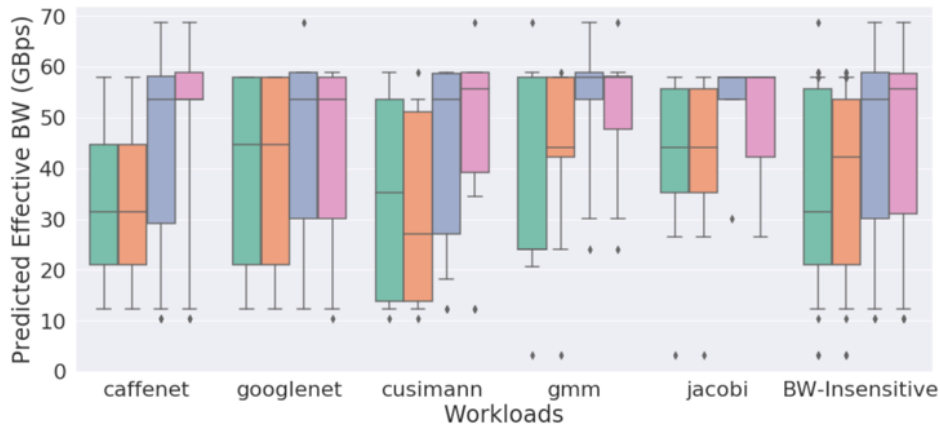
(b) Execution time of bandwidth insensitive jobs

Figure 3.13 Evaluation results (Execution Time) on DGX-V

at the 75th percentile execution time, which improved from 785s to 378s and 511s to 374s, respectively. Overall, Topo-aware reduced the 75th percentile execution time from 540s to 505s for bandwidth-sensitive jobs. However, this Topo-aware policy is not generalized to support arbitrary application and hardware topologies. As shown in Figure 3.14(a) and (b),



(a) Effective bandwidth of bandwidth sensitive jobs



(b) Effective bandwidth of bandwidth insensitive jobs

Figure 3.14 Evaluation results (Predicted Effective Bandwidth) on DGX-V

the chosen allocations’ effective bandwidth does not significantly improve upon the baseline policy with the barplot of baseline and Topo-aware being nearly identical.

We evaluate MAPA with two pattern selection allocation policies—Greedy and Preserve. The MAPA Greedy policy greedily selects the allocation with the most aggregated bandwidth. Although aggregated bandwidth does not correlate with effective bandwidth, the Greedy policy nevertheless significantly improves the quality of allocation. As shown in Figure 3.14(a) and (b), the median effective bandwidth across all workloads (57.85GBps for Greedy and Preserve) is nearly the maximum effective bandwidth of baseline and Topo-aware which does not take into account application and hardware topologies. This demonstrates the benefits of MAPA and the benefits of being application and hardware topology aware.

However, the Greedy policy does not consider application bandwidth sensitivity nor aims to preserve bandwidth for future bandwidth-sensitive workloads. In Figure 3.14(a) we see that the Greedy policy has allocations with a lower 25th percentile of effective bandwidth (12.33 GBps), indicating that more sensitive jobs are starved.

Policy	MIN	25th %	50th %	75th %	MAX	Tput
Baseline	1.000	1.000	1.000	1.000	1.000	1.00
Topo-aware	1.002	1.029	1.385	1.014	1.075	1.07
Greedy	0.997	1.059	1.519	1.048	1.319	1.08
Preservation	1.006	1.057	1.119	1.124	1.352	1.12

Table 3.4 Summary of results. Normalized execution time speed up and throughput (Tput) observed on DGX-1 V100.

The Preserve policy can successfully preserve bandwidth for bandwidth-sensitive workloads. This policy can achieve similar median effective bandwidth as the Greedy policy (57.85 GBps) without suffering at the 25th percentile. In many cases, the 25th percentile effective bandwidth also significantly improved as in the case of AlexNet and Inception-v3. In terms of execution time, the Preserve policy achieves the lowest maximum tail execution time and 75th percentile execution time (498s) across the majority of the networks.

Table 3.4 summarizes the speedup across all allocation policies and the quartiles, normalized to the baseline policy. By greedily selecting the most aggregated bandwidth, the Greedy policy performs the best in the median case at the cost of less improvement for the longer-running jobs at the tail. The Preserve policy can achieve the best speedup at the tail by improving the 75th percentile and Max by 12.4% and 35.2% over baseline. By improving the longer-running jobs, the Preserve policy can improve throughput by 11.7%. This throughput improvement is due to better utilization of available high-speed communication links, which results in higher GPU utilization and reduced execution times. Figure 3.15 shows the nearly uniform distribution of jobs across all the GPUs.

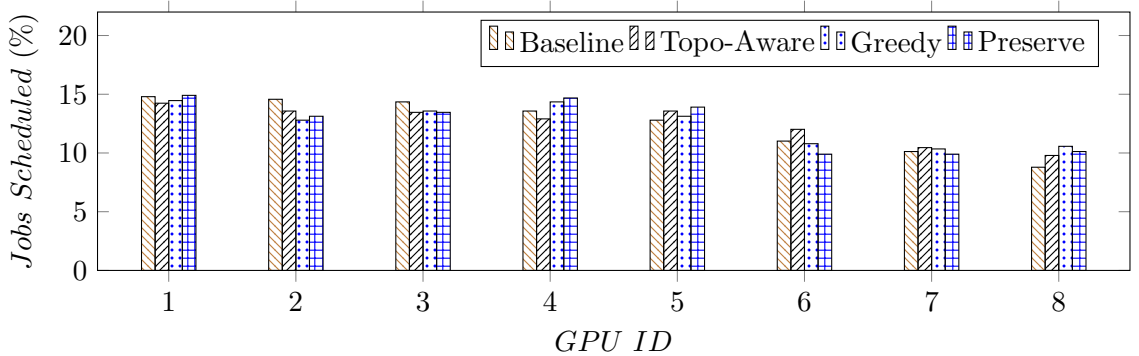


Figure 3.15 Number of jobs scheduled on a given GPU ID.

3.5 Exploring Novel Hardware Topologies

3.5.1 Methodology

To explore the effects of scheduling and fragmentation for novel accelerator topologies, we built the MAPA simulator framework to evaluate the quality of allocation for arbitrary hardware topologies.

The simulation takes as input the hardware topology graph and a job file consisting of jobs represented by the application pattern graph and its execution times. For the job file input to the simulator, we obtained the extracted application pattern graph and measured baseline execution time from our real-world runs on the DGX-V. The output of the simulator is the effective bandwidths of each job. In lieu of building a full-featured performance model to predict the execution time of the workload, our simulator uses effective bandwidth as a proxy for execution time.

3.5.2 Simulation Framework

Details of the simulation framework are shown in Figure 3.16. The simulation starts with a job file. Each row in a job file corresponds to a job and is annotated with a job ID, number of GPUs, application topology, and bandwidth sensitivity. The *Dispatcher*

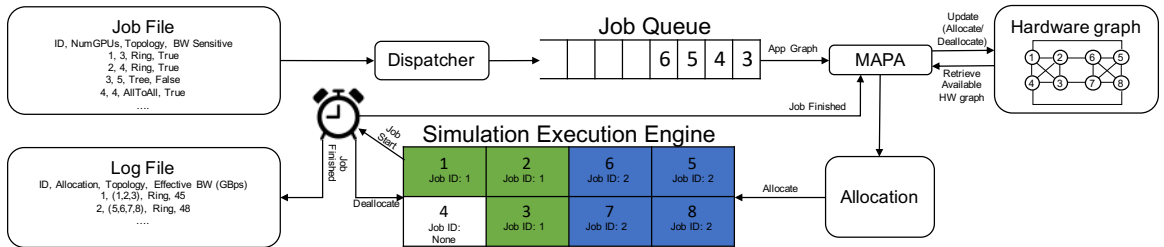


Figure 3.16 MAPA simulation execution framework.

reads the job file and puts the job in the *Job Queue*. The Job Queue employs a First-in First-out policy to mimic the FIFO scheduling in real-world experiments. If there exist available GPU resources, the simulator invokes MAPA to obtain an allocation for the next job.

The execution engine of the simulator is cycle-based and models the availability of a hardware resource. When a job is allocated, we flag the hardware as busy, record the cycle time, and begin the execution of the job. Once the specified execution time has elapsed, we flag the hardware resources as free, log the job’s information into a log file, and send a *Job Finished Signal* to MAPA to update its hardware state. The logger records the Predicted Effective Bandwidth information along with other job properties.

Simulator validation and soundness of effective bandwidth proxy. In order to validate the simulator with real runs, we correlate the predicted Effective Bandwidth obtained in the real run results with the simulator configured for DGX-V. As shown in Figure 3.17, the simulated and real effective bandwidth correlates well indicating that the simulation adequately captures the scheduling behaviors of the real DGX-V system. We believe this simulation methodology can scale to evaluate future topologies since our evaluation metric (effective bandwidth) is based on the resource provisioned for a job, and not based on global topology properties. Therefore, we’re confident our simulator result is accurate for future topologies utilizing the same link types.

To demonstrate the soundness of using effective bandwidth as a proxy for execution time, we collected the effective bandwidth and measured the execution time of the real run for each workload. As shown in Figure 3.18, we can see for bandwidth insensitive workloads

that execution time is not impacted by effective bandwidth as expected. For bandwidth-sensitive workloads, as effective bandwidth increases the execution time of the workload also improves (decreases). Although the amount of execution time improvement is limited

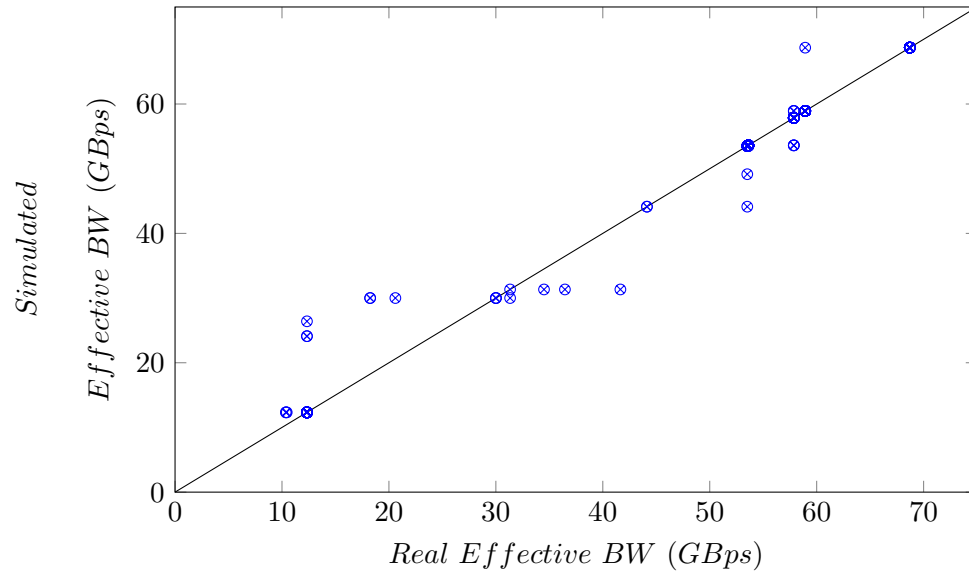


Figure 3.17 Effective bandwidth measured during DGX-V's real and simulation runs correlated well.

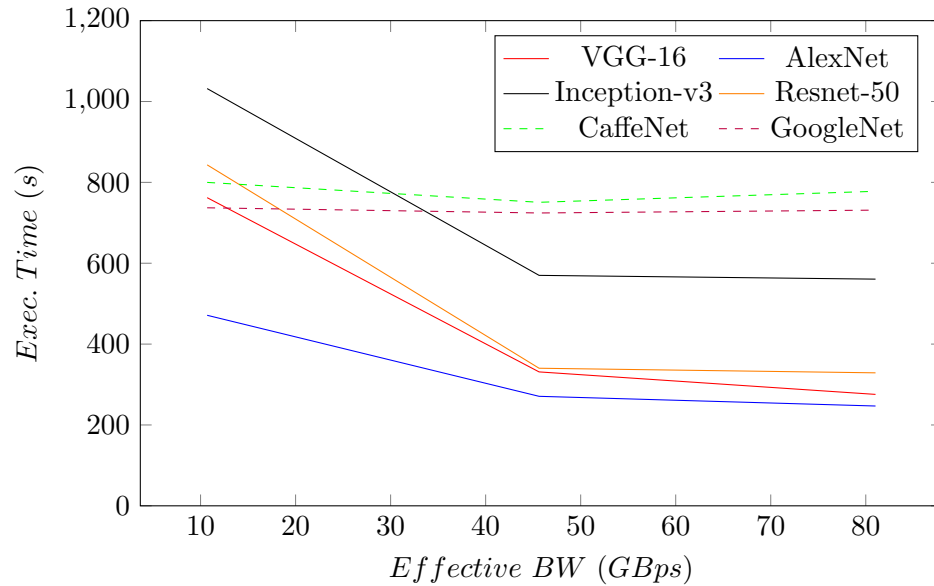


Figure 3.18 Effective bandwidth vs execution time observed during real runs on DGX-V.

once the effective bandwidth is past 50 GBps, the general trend still holds. Thus, effective bandwidth can be used as a good proxy for evaluating execution time improvements.

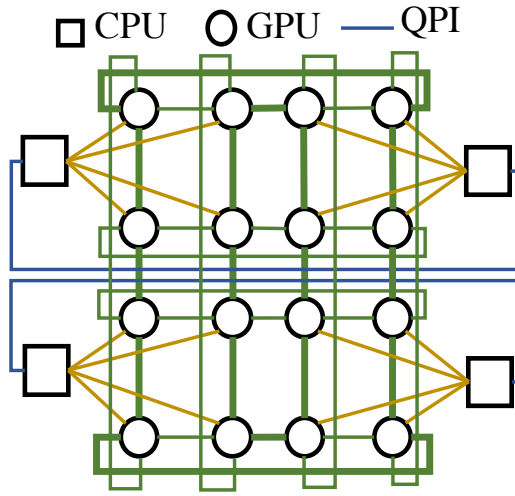
Novel 16-GPU topologies. We explore the impact of scheduling policies on two-novel 16-GPU hardware topologies – Torus-2d, and Cube-mesh topologies. The accelerators in Torus-2d and Cube-mesh topology are configured to have double NVLinks, single NVLinks, and PCIe as shown in Figures 3.19a and 3.19b, respectively. Although 16-GPU topologies exist with crossbar switches (NVSwitch), we aim to explore alternative topologies consisting of cost-effective point-to-point links.

3.5.3 Exploration Results

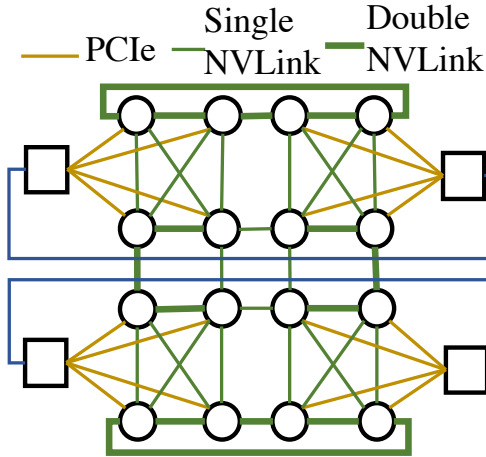
Recall that the aim is to improve the upper tail of execution time, and by proxy, to improve the lower tail (min and 25th percentile) of effective bandwidth. For brevity, we omit the results for bandwidth insensitive workloads since the execution times of these workloads are not impacted by effective bandwidth as shown in Figure 3.18.

For the 16-GPU Torus-2d (Figures 3.20a), we observe that Preserve significantly improves the 25th percentile and is better than the median of baseline and Topo-aware. In addition, the minimum of Preserve is equivalent to the 25th percentile of all other policies, demonstrating Preserve’s ability to rein in the tail execution time. Due to the uniformness of the Torus-2d interconnect network, the Greedy policy can easily select high bandwidth allocations improving the 75th percentile (making fast jobs even faster).

For the Cube-mesh topology (Figure 3.20b), it is a more irregular network and thus more difficult to greedily select optimal allocations. Here, Preserve performs even better for



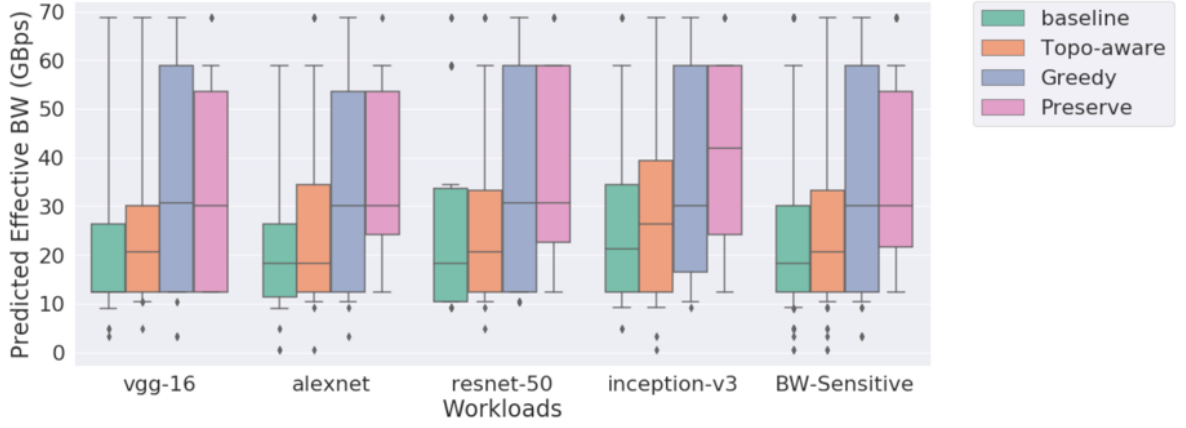
(a) Torus-2d



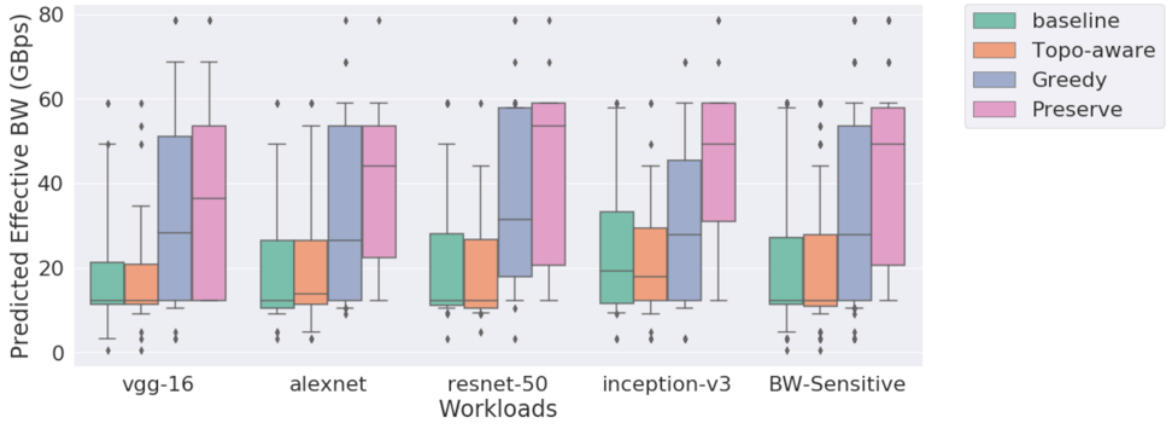
(b) Cube-mesh

Figure 3.19 16-GPU topologies

sensitive workloads. While the minimum effective bandwidth of Preserve is equivalent to the 25th percentile of all other workloads, the median is near the 75th percentile of Greedy and the maximum of baseline and Topo-aware. **Therefore, half of the jobs allocated with Preserve will effectively run faster than all of the jobs with baseline and Topo-aware and the majority of Greedy.**



(a) Torus-2d



(b) Cube Mesh

Figure 3.20 Simulation results for bandwidth-sensitive workloads on 16-GPU topologies. Improvements to the lower tail (min and 25th percentile) are better in both.

These results demonstrate that as hardware topologies scale and become more complex and non-uniform, the greater the need for scheduling and allocation policies that are application communication pattern-aware and hardware topology-aware.

3.5.4 Overhead of Scheduling

Figure 3.21 presents the scheduling overhead analysis of the MAPA framework. We evaluate this overhead across different sizes of application pattern graphs (x-axis) and different sizes of hardware topology graphs. We evaluate hardware topology graphs of sizes 6, 8, and 16 for Summit, DGX-V, and Torus-2d/CubeMesh-16, respectively. Typically, we observe scheduling overheads in the negligible order of milliseconds. However, the scheduling overhead does increase modestly for larger job sizes (9 GPUs and above) on larger hardware graphs (16 GPUs with 120+ edges). This is due to more combinations of matching patterns which require more scoring of patterns.

Note that this experiment is done on an idle hardware graph and *represents an upper-bound of scheduling cost*. In reality, the allocation search will be performed on a smaller graph of available hardware which leads to significantly smaller pattern matches. Also, our evaluation utilizes a single-thread implementation to perform scoring. This overhead can be reduced by parallelizing the scoring process since it is a data parallel problem. Therefore, we expect our overhead to be manageable in real-world conditions and can scale to larger servers with parallel optimizations of our implementation.

3.6 Related Works

Scheduling for multi-node GPU clusters: Many works [114, 5, 115, 116, 134, 135, 4, 109, 58, 103, 63, 59, 24, 41, 128, 64, 3, 2] have proposed optimizations to improve GPU performance and Energy efficiency. Recent works, such as Gandiva [131] and Philly [45], propose scheduling policies for multi-GPU jobs on multi-node multi-GPU

clusters. Specifically, Gandiva proposes support for transparently migrating and time-slicing jobs for better job-to-GPU fit. Philly, on the other hand, aims to maximize the locality of multi-GPU allocations for non-preemptive multi-node multi-GPU clusters in multi-tenant environments. Both prior works aim to minimize fragmentation by either adding preemption support for migration, or by allocating across nodes to minimize fragmentation. Our work is complementary and aims to alleviate fragmentation that occurs within a node itself in a multi-tenant environment due to the heterogeneity of links.

Collective communication: In [122, 54], the authors have proposed techniques for achieving efficient collective communication. Blink [122] offers a new approach to collective communication by creating sets of spanning trees instead of rings. The spanning trees are dynamically generated based on the topology detected to utilize the links best. Specifically, given allocations from Philly, which are unaware of GPU-GPU interconnection topology, the goal of Blink is to identify optimal communication paths using spanning trees. Gossip [54] proposes flow-oriented collectives and generates transfer plans to best schedule packets. Works like WOTIR [89] present software optimization techniques to improve the execution times of bad allocations using NVLink. These works seek to optimize bad allocations, while our work seeks to reduce the number of bad allocations for bandwidth-sensitive jobs.

Multi-GPUs for Machine Learning: From recent works [131, 45, 69], Machine Learning (ML) is one of the primary workloads on multi-GPU systems. Hence, we use ML training as a target workload in this chapter, as well. We used Caffe [46] framework for Machine Learning in this chapter. These machine learning workloads use Nvidia Collec-

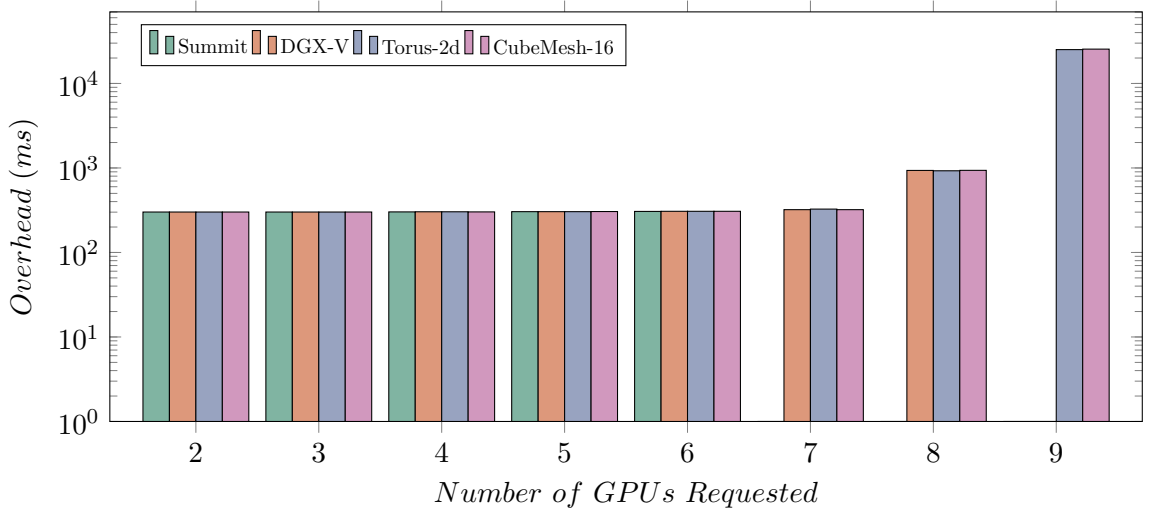


Figure 3.21 Overhead analysis of MAPA with Preserve policy.

tive Communications Library (NCCL) [77] to perform operations like Reduce, AllReduce, Broadcast, Gather, Scatter, and Scatter-Gather. While we only demonstrated software NVLink routing in NCCL integrated into Caffe, our observed results and trends should generalize to other machine learning frameworks that use NCCL as the collective communication backend. Besides, as ML models grow in size and complexity, the communication intensity will only increase, leading to a greater reliance on maximum achievable communication bandwidth.

3.7 Conclusion

In this chapter, we proposed Multi-Accelerator Pattern Allocation (MAPA), a novel approach to perform efficient scheduling and allocation of multi-accelerator workloads on multi-accelerator systems using a generalized graph pattern matching approach. Through real-world evaluations, MAPA improves overall system throughput by up to 12%

and reduced the worst-case execution time by 35% over baseline. Through simulation, we explore larger novel hardware topologies and find that MAPA’s benefit grows as hardware topologies scale and becomes more non-uniform. We demonstrate that more than half of the jobs allocated with MAPA can effectively run faster than all jobs allocated with existing state-of-the-art scheduling policies.

Chapter 4

Adaptive Collective Communication Topology

4.1 Introduction

Modern data-intensive applications such as Machine Learning, Artificial Intelligence, and High-Performance Computing demand ever-increasing performance demands. Modern accelerators such as General Purpose Processing in Graphical Processing Units (GPGPUs), Tensor Processing Units (TPUs) [48], and Tensor Streaming Processors (TSP) [6] are increasingly adopted in modern Cloud and High-Performance Computing Systems. Additionally, these systems are equipped with many accelerator modules within a said system. For instance, Amazon P3DN instance [126] and Facebook’s Big Basin [26] consist of eight GPUs connected in a hybrid cube mesh topology as shown in Figure 4.1. We can

also notice that these systems can have different interconnect bandwidths across different combinations of GPUs.

Furthermore, modern multi-accelerator systems are increasingly shared not only across many clusters of servers but also within the servers. Job schedulers such as Slurm [133] are used to facilitate the partitioning and allocation of requested resources within and across the system. Many recent papers such as Gandiva [131], MAPA [91], and WOTIR [89] have demonstrated the need for novel allocation strategies to improve the throughput of the system. MAPA even proposed intelligent graph-based allocation policies to ensure efficient utilization of all the available NVlinks. We also noticed in chapter 3 that sharing the systems can lead to fragmented allocations where the bandwidth may suffer. Works like Blink [121] have shown promising results when it comes to improving peak bandwidths among inter-server GPUs using trees.

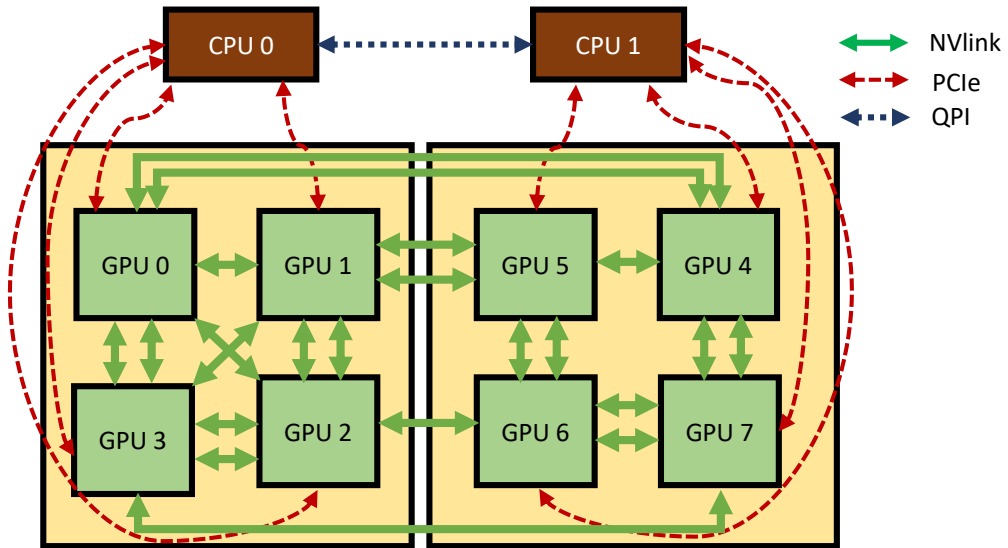


Figure 4.1 Nvidia DGX-V Hybrid Cube Mesh topology

Nvidia Collective Communication Library (NCCL) [77] is one of the popular libraries used to perform intra-server and inter-server collective communication operations efficiently. In this chapter, we explore a novel allocation system using modern NCCL capabilities to further reduce fragmentation in a multi-accelerator system. We believe that using these strategies could reduce execution times of jobs and thereby improve the overall speedup of the system.

We extend the graph matching allocation policy from Chapter 3 to an additional account for adapting the collective communication topology that is utilized. By additionally allowing collective communication topologies to utilize ring or tree topologies, we can further reduce the fragmentation. We use the modern NCCL capabilities to drive these allocation strategies and just like MAPA, provide a generic framework that can be used in any multi-accelerator environment.

The outline of this chapter is as follows:

1. Performance analysis of different collective communication topologies using modern NCCL capabilities [43] to motivate the need for better topology-aware allocation policies.
2. CC-Topo, a Collective Communication Topology selection approach for scheduling multi-accelerator workloads in multi-accelerator systems.
3. Propose Linear regression and Machine Learning-based modeling techniques to predict the topology-specific effective bandwidth.

4.2 Motivation

Modern Cloud and High-Performance Computing systems are increasingly becoming heterogeneous as shown in Chapter 3.2. These heterogeneous systems are optimized to accelerate modern ML and AI workloads that demand massive parallelism and data compute capabilities. The demand for massive performance along with inefficient allocations puts the system in a precarious situation. We can also see how commonly fragmentation occur in a DGX-styled multi-GPU system as described in Chapter 2.1.2. In the next subsection, we shall discuss the limitations of the existing state-of-the-art solutions like MAPA.

4.2.1 Limitations of MAPA

MAPA uses a graph-pattern-based allocation system as described in 3.3. In this approach, the framework takes in the hardware topology and an application topology to sort all the matching subgraphs within the hardware topology. The framework uses a regression model to predict the effective bandwidth that could be achieved in a given matching allocation.

Nvidia’s Collective Communication Libraries has introduced support for their collectives to run operations such as *AllReduce* in a tree topology. This means that the application can be configured to be run as a tree or ring. Thus, applications do not have a specific communication topology that needs to be honored at the time of allocation.

4.2.2 Tree and Ring Bandwidths

In order to demonstrate the difference in peak Bandwidths between ring and tree topologies, we run a set of NCCL AllReduce microbenchmarks on all 2, 3, 4, 5, and 6 GPU allocations in a DGX-V system 4.1. The results are shown in scatterplot Figure 4.2. We can notice that the allocations above the regression line can achieve higher peak bandwidths in a tree topology whereas the allocations below reach higher peak bandwidths as a ring. Hence, having a topology-aware allocation could enable us to dynamically choose between rings and trees based on achievable peak bandwidth.

Now let us assume that we have a 3-GPU job in the queue and the only GPUs that are free are 0, 3, and 4. Since MAPA uses a First Come First Server (FCFS) policy, the job is allocated with the available GPUs. Figure 4.3a shows the peer configurations by NCCL

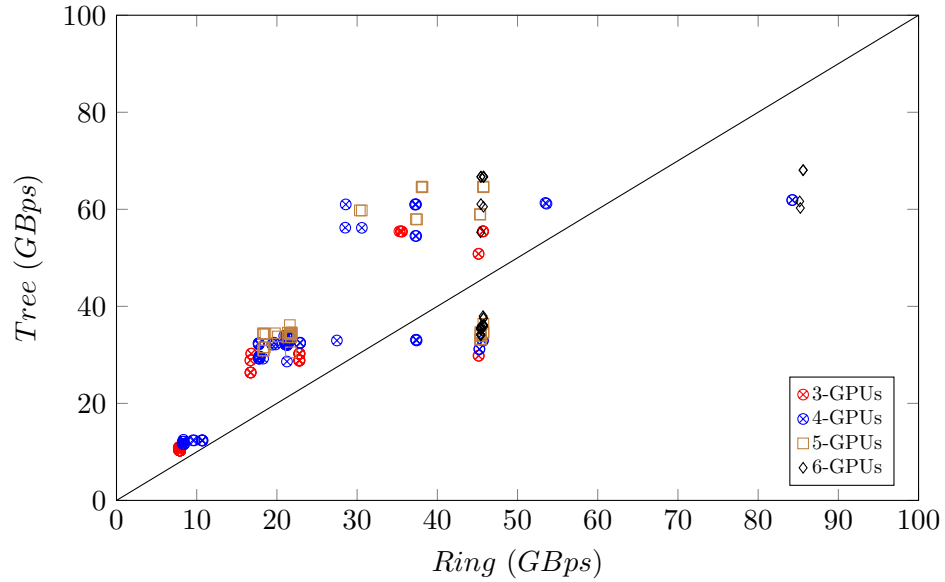


Figure 4.2 Peak bandwidths were observed while running the microbenchmarks for tree and ring application topologies.

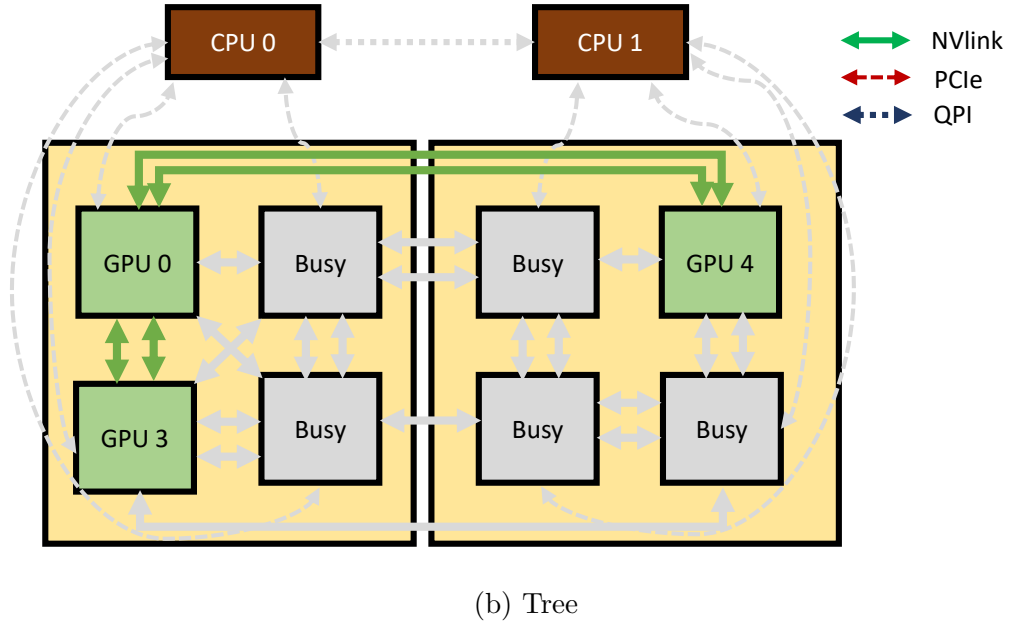
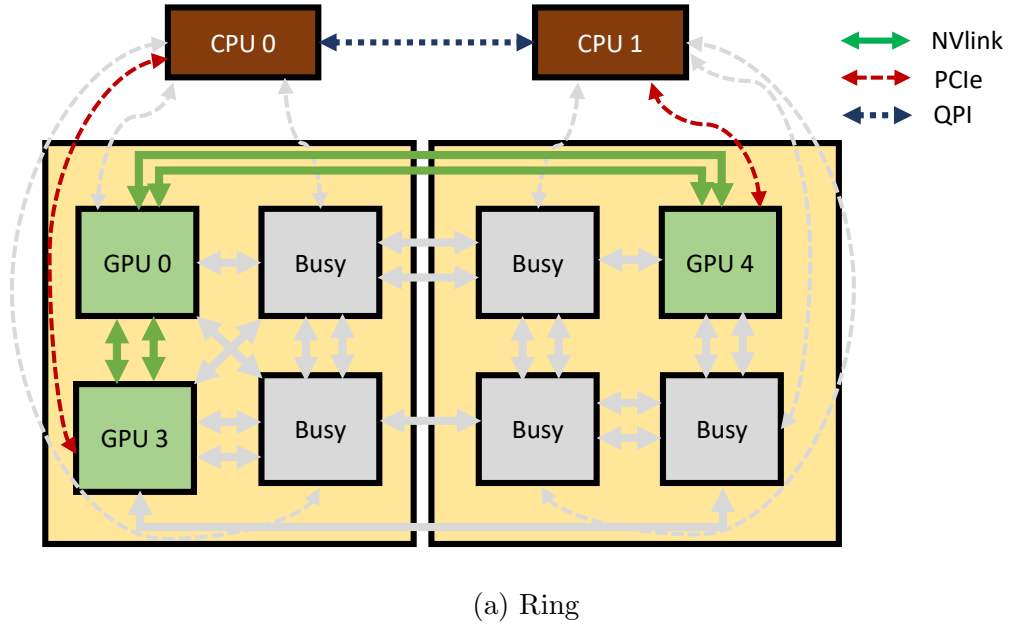
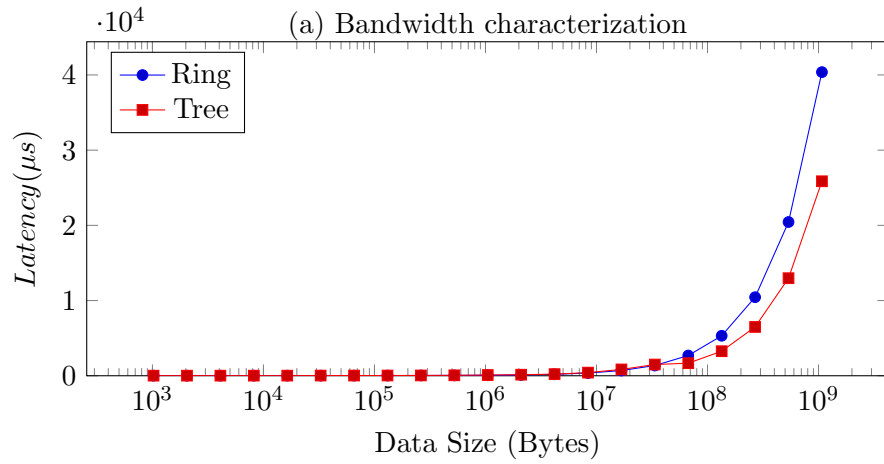
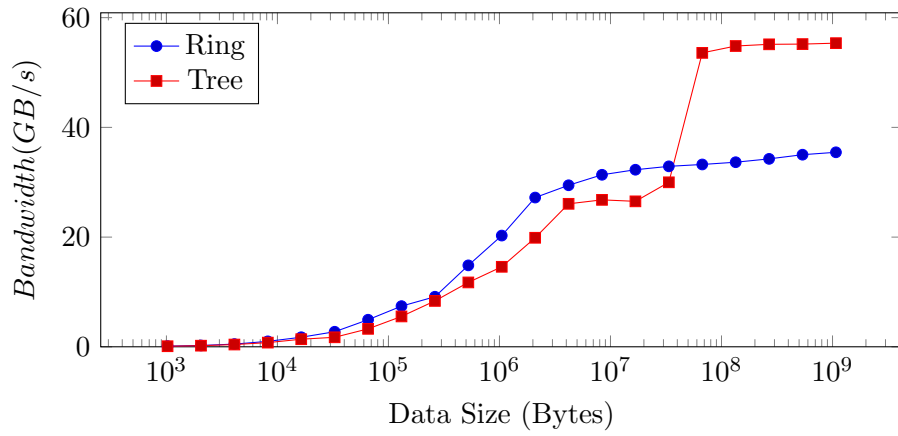


Figure 4.3 Example 3-GPU allocation 0,3,4

when the workload is operated as a ring. This allocation would be treated as fragmented if operated as a ring due to the PCIe link between GPUs 3 and 4 that is unavoidable. This takes a toll on the peak achievable bandwidth as shown in Figure 4.4(a). The ring topology

can at most reach 35 GBps. However, if we configured the available allocation as a tree with GPU 0 as a root, we can avoid the PCIe links as in Figure 4.3b and improve the peak bandwidth up to 55 GBps.

Figure 4.5 shows the % of in 3, 4, 5, and 6 GPU allocations where Tree topology outperforms rings. There are 8 3-GPU allocations where Trees outperform rings by at least



(b) Latency characterization

Figure 4.4 Characterizing fragmented allocation 0,3,4

10 GBps. Next, there are 40 4-GPU allocations that are at least 10 GBps better than rings and there are 6 allocations that are at least 20 GBps better. Similarly, 38 5-GPU allocations beat rings. Moreover, the majority of 4 and 5-GPU allocations perform well in a tree topology. Hence, it is important to use allocation policies that can also predict the right communication topology.

The effect of variations in the peak achievable tree and ring bandwidths on Resnet-101 and VGG-16 training jobs for all the 3 and 4 GPU allocations as shown in Figures 4.6a, 4.6b, 4.7a, and 4.7b.

Many allocations such as (0,3,5) for Resnet and VGG in Figures 4.6a perform better as trees. Similar illustrations can be found in 4-GPU allocations (0,1,3,7) and (2,3,4,5) for Resnet and VGG in Figures 4.6b and 4.7b respectively. Furthermore from Figures 4.7 and 4.6, it is evident that are many allocations where the performance of trees was better

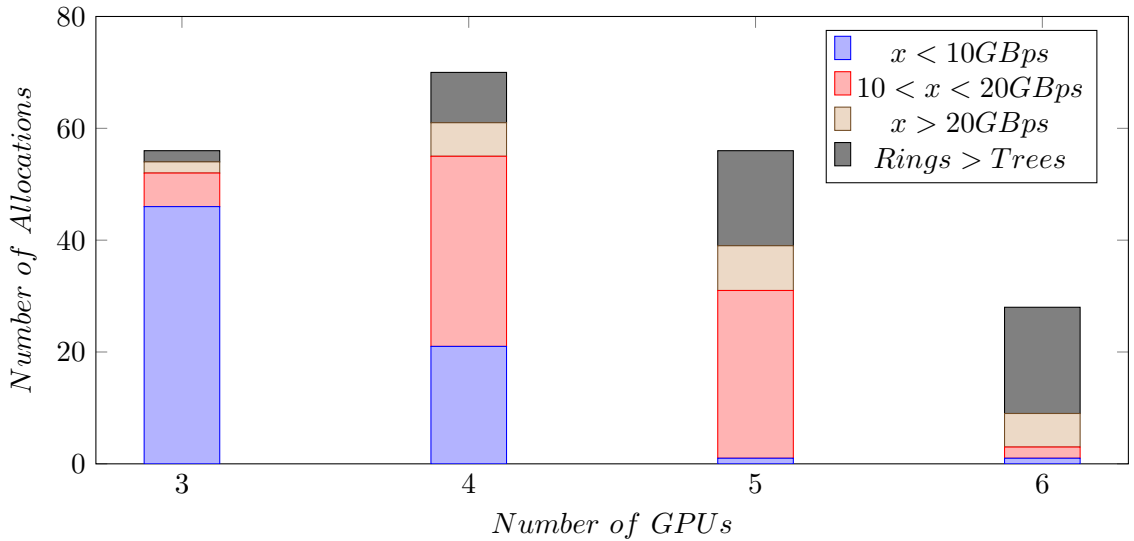


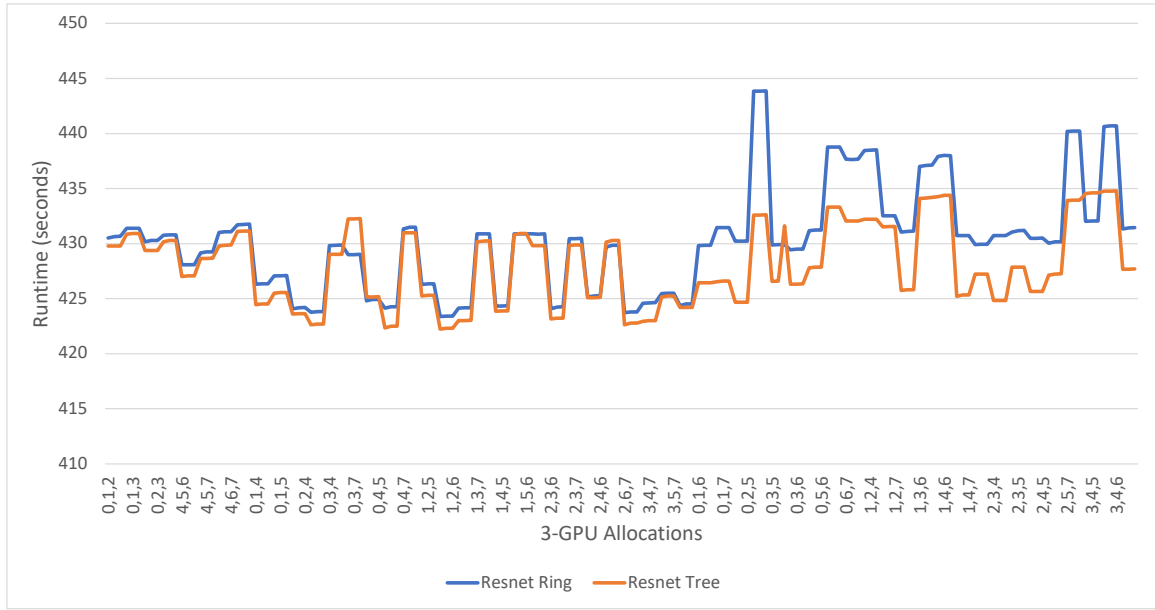
Figure 4.5 Allocations where Peak bandwidths in Trees were greater than Rings. Where $x = Tree_{BW} - Ring_{BW}$

than rings. Hence, characterizing a given allocation to determine peak achievable bandwidth can drive allocation policies to score allocations according to the intended application topology.

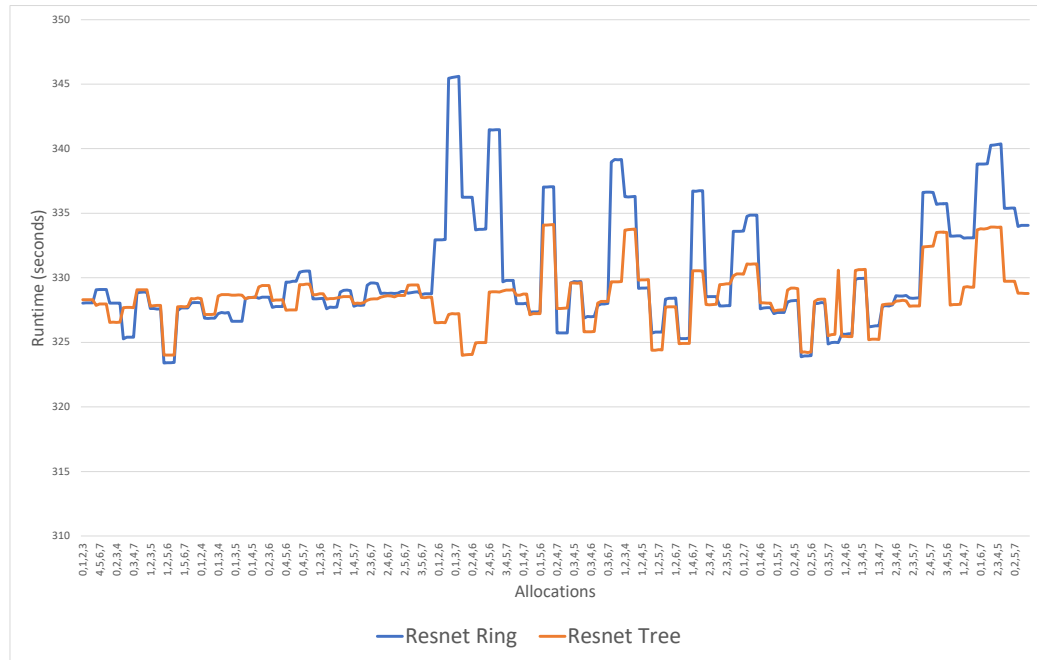
Topology detection: Chapter 3.3.1 described approaches to determine the application topology of any arbitrary workload using source code analysis and runtime profiling. Although it is indeed a generic solution to ideally map any application to any hardware topology, the methods incur user-overhead and certain profiling methodologies demand superuser privileges on the system. Hence, having an allocation policy that will find an allocation along with a collective communication topology could minimize fragmentation, and potentially achieve near to no fragmentation.

4.3 Collective Communication Topology Awareness

CC-Topo is a Multi-accelerator pattern allocation system that finds a suitable allocation to achieve the highest possible bandwidth along with communication topology. The overview of CC-Topo framework (built on top of MAPA) is shown in Figure 4.8. The system consists of a hardware topology of the accelerator system that can be extracted with libraries such as the Nvidia Management library (NVML). The user requests the number of accelerators that are needed to successfully run the application. Based on the number of accelerator nodes (n) the user has requested, the vertex combination generator identifies k n -GPU allocations. All the k combinations are scored for tree and ring topologies. The allocation policy then uses the score to decide the ideal allocation for the application.

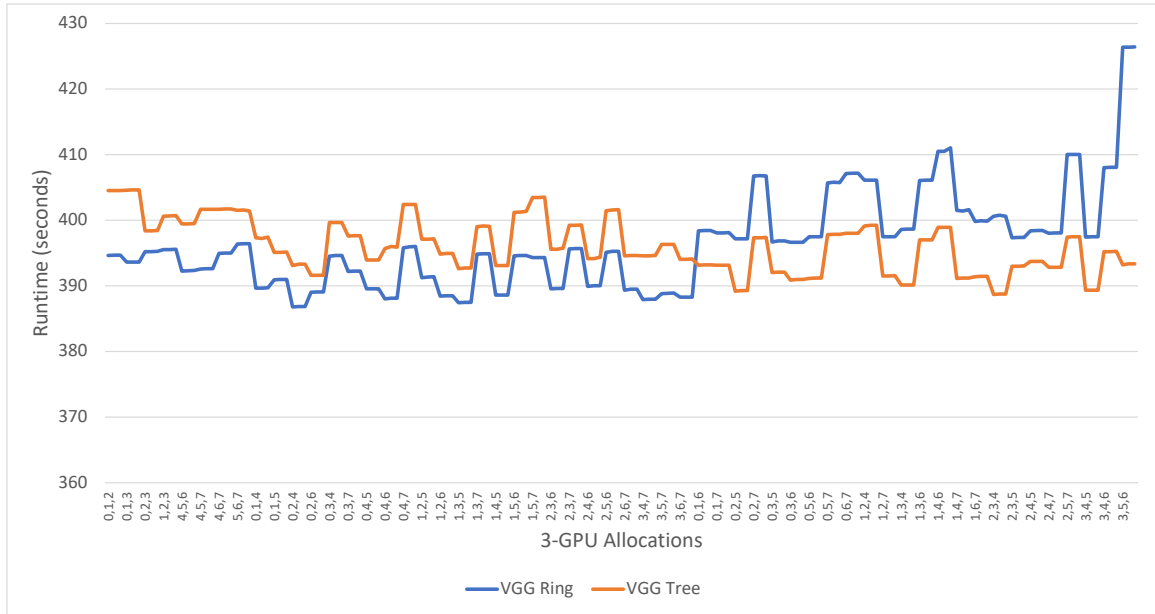


(a) Resnet-101 3-GPU training time for 1 epoch in rings and trees

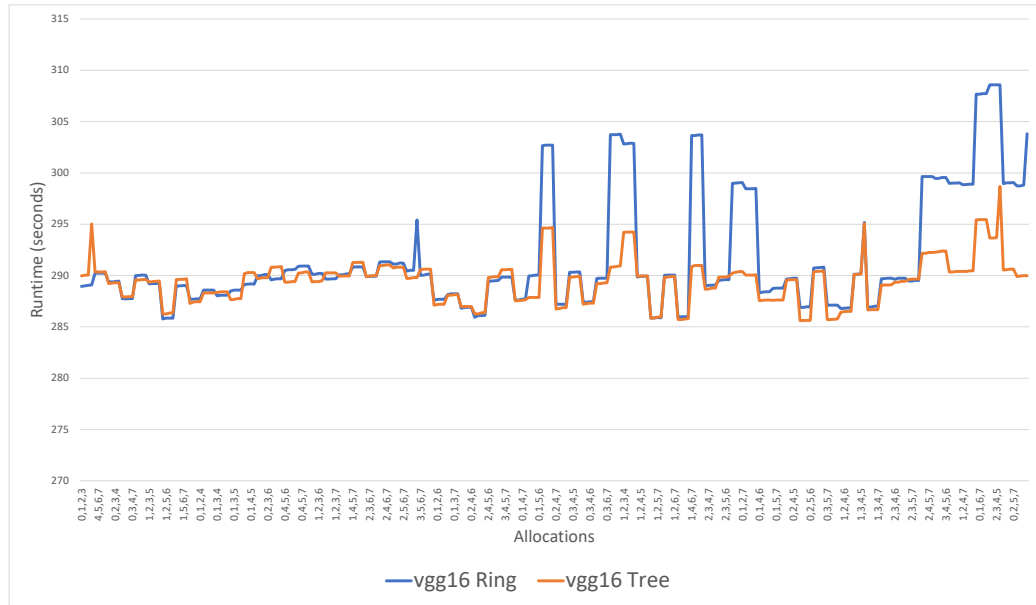


(b) Resnet-101

Figure 4.6 Training time for 1 epoch on all the possible 4-GPU allocations in DGX-V



(a) VGG-16 3-GPU training time for 1 epoch in rings and trees



(b) VGG-16

Figure 4.7 Training time for 1 epoch on all the possible 4-GPU allocations in DGX-V

Finally, the hardware topology is updated to reflect the successful mapping of the application onto the allocation.

4.3.1 Combination Scoring

In the previous chapter 3, the MAPA framework assigned a suitable score to determine the impact of the scored allocation on application speedup. From Section 3.3.4, we know that the Peak Effective Bandwidth can have a direct impact on the performance of a job. In this section, we model the peak effective bandwidths of all the possible 3, 4, 5, and 6-GPU allocations using Linear regression and the Neural Additive Model (NAM). In addition, CC-Topo is also tasked to determine the suitable communication topology for the allocation. We first modeled the Peak Effective Bandwidth of trees and rings.

Linear regression

We model the Peak Effective bandwidth for Trees and rings using the following main features obtained from hardware topology for a given allocation of A .

- **GPU IDs:** A vector of GPU IDs in the allocation.

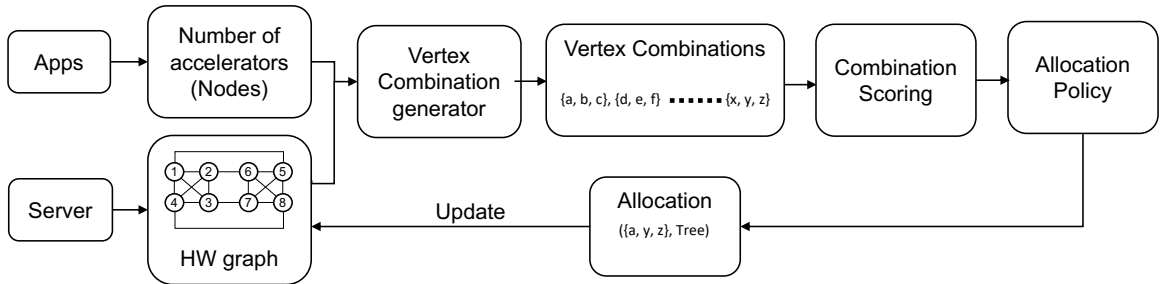
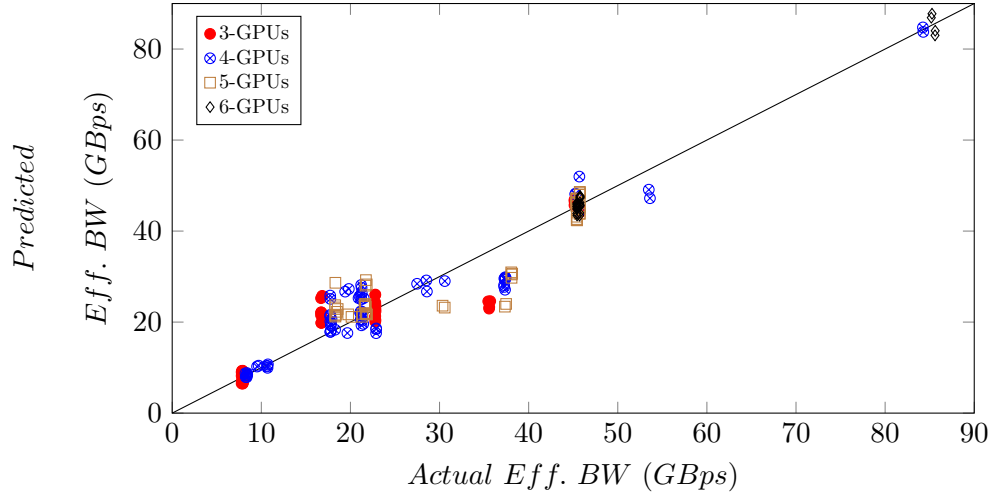
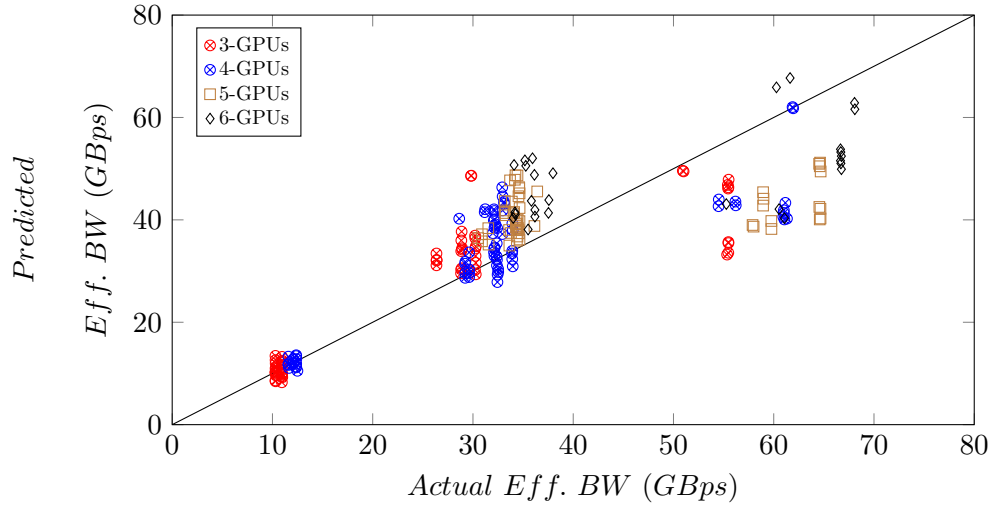


Figure 4.8 Overview of MAPA + CC-top framework

- **Number of Double NVlinks (NvD):** number of double NVlinks when GPU IDs are connected in all-to-all topology.
- **PCIe links (Pci):** number of PCIe links when GPU IDs are connected in all-to-all topology.



(a) Ring



(b) Tree

Figure 4.9 Modeling tree and ring bandwidths using linear regression

- **Maximum PCIe to Nvlink Ratio (MxP2N):** Maximum number of PCIe links in all combinations of GPU IDs in a collective communication topology.
- **Minimum PCIe to NVlink Ratio (MnP2N):** Minimum number of PCIe links in all combinations of GPU IDs in a collective communication topology.
- **PCIe Avoidable (PciAvoid):** Number of combinations of GPU IDs in a given collective communication topology that has no PCIe links (all NVlinks).

We regress the features to ascertain a linear relationship to accurately predict the peak effective bandwidth for a given tree and ring allocation. The overall R^2 , Root Mean Square Error (RMSE), and Mean Absolute Error (MAE) are 0.809, 7.395, and 4.956 respectively.

The scatter plot in Figure 4.9 describes the accuracy of ring and tree models. Figure 4.9 (a) shows the distribution of Actual Peak Effective bandwidth and Predicted Effective Bandwidths in rings. The ring model prediction for 6-GPU allocations are nearly accurate. However, the model is mispredicting some of the 3, 4, and 5-GPU allocations whose actual effective bandwidth happens to be around 40 GBps. The R^2 , RMSE, and MAE for the ring model are 0.943, 4.03, and 2.715 respectively.

Figure 4.9(b) shows the detailed information of the prediction accuracy of tree bandwidths for all allocations. The model is clearly mispredicting for 5 and 6-GPU allocations in the 35, 60, and 70 GBps marks. The R^2 , RMSE, and MAE are 0.653, 9.651, and 7.196 respectively.

Since the Tree Regression model has a high RMSE and MAE, we need another metric to infer the usefulness of this model. We propose a metric called **Decision Accuracy** in Equation 4.1 to determine the number of misclassifications between trees and

rings. In this equation, R and T are R and T refers to actual Ring and Tree bandwidths whereas $\hat{R}$ and $\hat{T}$ refer to predicted Ring and Tree bandwidths. Analyzing the predicted Effective bandwidth using this metric showed an accuracy of 22.8%. More than 77% of the time the regression failed to accurately infer if $(T > R)$ or $(R > T)$. We shall discuss a Machine Learning based approach to model the non-linear nature of the Ring and Tree Peak Bandwidths.

$$Decision\ Accuracy = ((\hat{R} > \hat{T}) \ \& \ (R > T)) \mid ((\hat{R} < \hat{T}) \ \& \ (R < T)) \quad (4.1)$$

Neural Additive Model (NAM)

Traditional Machine Learning techniques such as Deep Neural Networks (DNN) have proven to be a promising approach to providing intermediate representation for various types of data. However traditional solutions suffer when it comes to learning *Jagged* shape functions. Neural Additive Models (NAM) [7] is a promising solution to more accurately model Jagged shape functions. Each feature takes a different branch in this model and the final prediction is obtained by adding all the non-linear feature functions.

We use the same feature set as used in the Regression model in Section 4.3.1. The model learns from all the features $NvD, Pci, \dots MnP2N$. Finally, the predicted bandwidth is obtained using equation 4.2.

$$Predicted\ Eff.\ Bandwidth = f([GPU\ IDs]) + f(NvD) + \dots f(MnP2N) \quad (4.2)$$

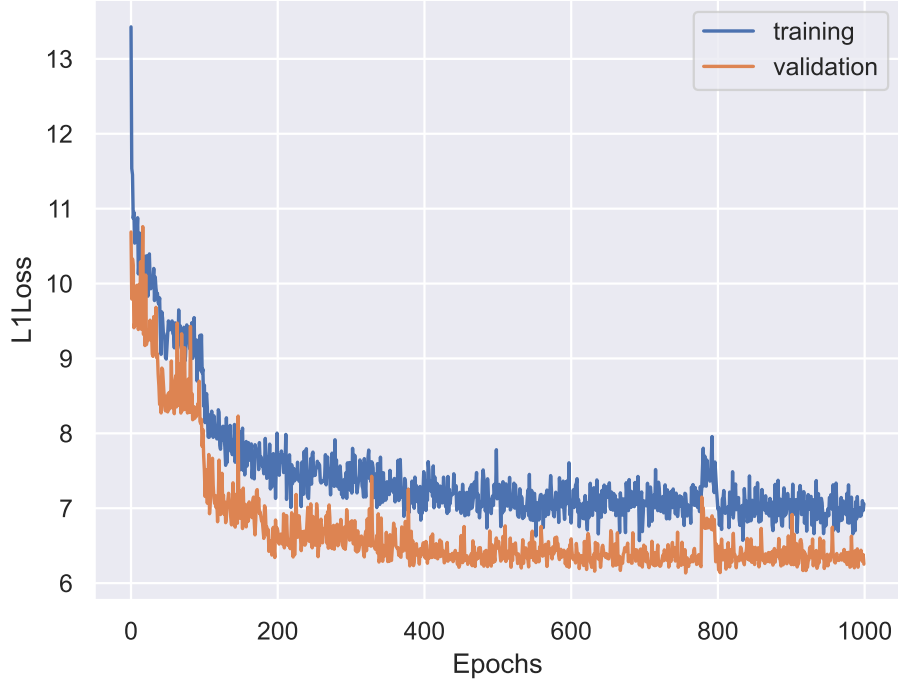
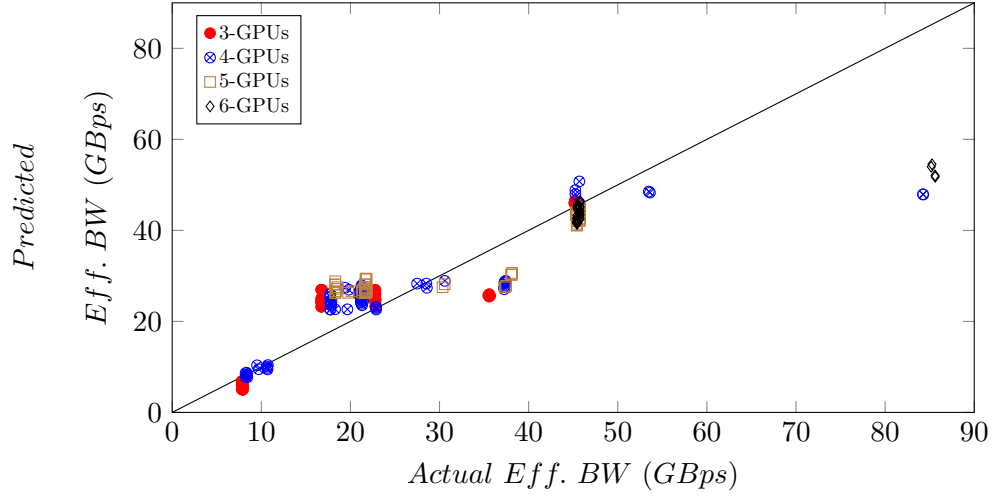


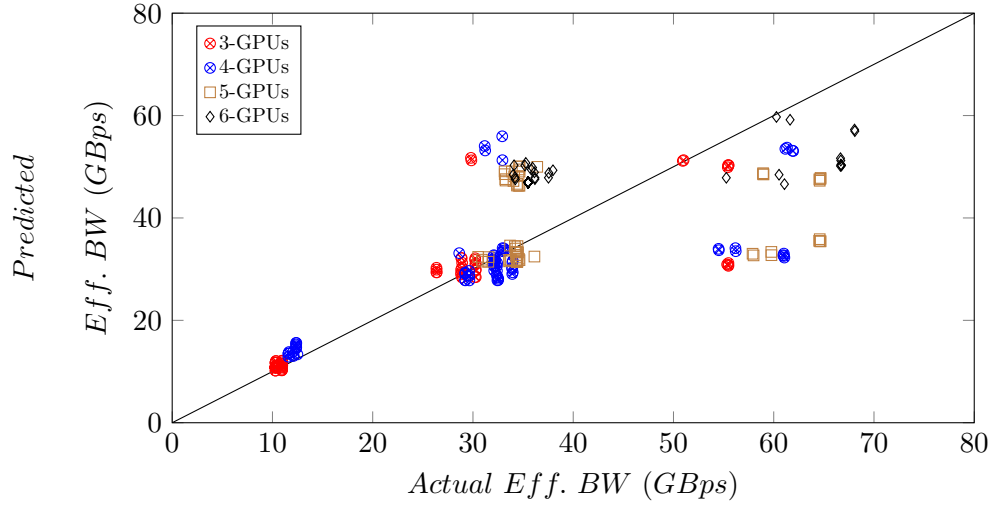
Figure 4.10 NAM model learning for Predicting Ring and Tree Bandwidths. L1Loss is MAE loss function.

The NAM model for 1000 epochs using the aforementioned features. The model started to converge at around 200 epochs as seen in Figure 4.10. Although the overall RMSE and MAE for NAM training were 9.581 and 6.135 respectively, the decision accuracy from Equation 4.1 showed that 77.6% of the time, NAM models were able to appropriately predict the bandwidth more accurately.

Figure 4.11 shows the breakdown of Actual Effective bandwidth and Predicted Effective Bandwidth for ring and tree topologies. The scatter plot in Figure 4.11 (a) shows the Predicted effective bandwidth for rings. The model greatly suffers when predicting the few 4 and 6-GPU outliers which lie around 80 GBps of Actual Effective Bandwidth. The R^2 , RMSE, and MAE for rings using the NAM model are 0.611, 7.464, and 4.727 respectively.



(a) Ring



(b) Tree

Figure 4.11 NAM results for CC-Topo

The prediction of tree bandwidth as shown in Figure 4.11 (b). There are many 4, 5, and 6-GPU mispredictions in the tree model. It has an R_2 , RMSE, MAE of 0.598, 11.309, and 7.543 respectively. It has to be noted that the same NAM model is used to predict the tree and ring bandwidth by flipping the topology variable to derive the respective Peak

effective bandwidth. Hence, using the NAM-based model to obtain the predicted effective bandwidth is a viable solution.

4.3.2 CC-Topo Allocation Policy

We present the CC-Topo allocation policy in Algorithm 2. In this policy, the algorithm expects the user to only provide the number of GPUs required for the job. CC-Topo picks the predicted peak bandwidth for all the possible combinations of GPUs in the hardware graph for the job. The policy returns IDs of the chosen GPUs along with collective communication topology and predicted peak bandwidth.

Algorithm 2: CC-Topo Allocation Policy

Result: Allocation

HWgraph hGraph;

int numGpus;

Allocation alloc = { gpuIDs = { }, topo = { }, bw = 0 };

Patterns possibleCombinations = CombinationGenerator (numGPUs, hGraph);

foreach *combination in possibleCombinations* **do**

 Allocation currAlloc = EffBandwidthUsingNAM (combination);

if *currAlloc.bw > alloc.bw* **then**

 alloc = currAlloc;

end

end

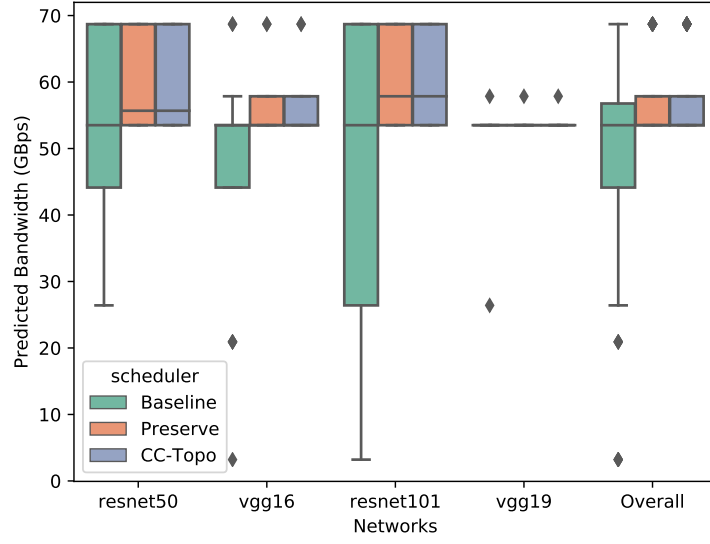
4.4 Evaluation

CC-Topo was evaluated on an Nvidia DGX-1 V100 System running Ubuntu 16.04 with CUDA-11.3 and NCCL-2.10.3. The 8-GPUs are connected in a Hybrid Cube mesh interconnect topology as shown in Figure 4.1. We extended MAPA framework [91] by adding allocation policies driven by the NAM models discussed in Section 4.3. We retained the First-in First-out (FIFO) scheduling strategy for dispatching jobs in the queue.

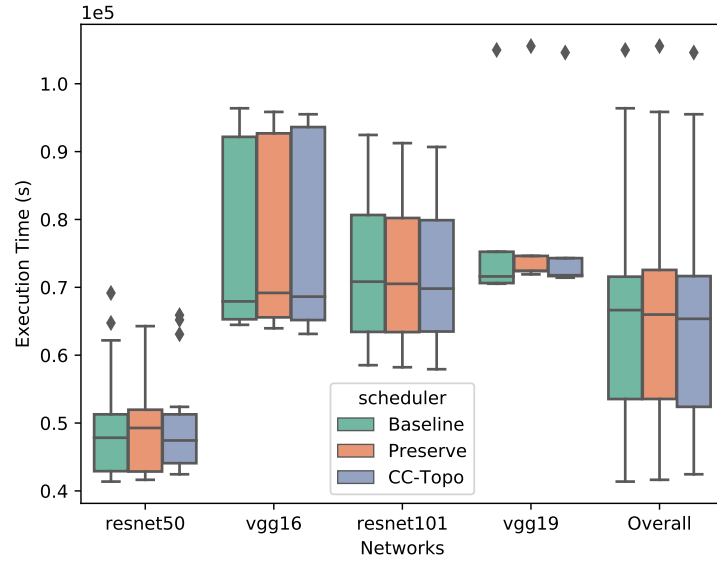
Workloads: We used PyTorch multi-GPU training jobs – VGG-16, VGG-19, ResNet-50, and ResNet-101. We used VGG and ResNet mainly due to their sensitivity to interconnect bandwidths as described in chapters 2 and 3. We randomly generated 50 jobs consisting of a uniform mix of training jobs with a random number of requested GPUs ranging from 3 to 6.

Allocation Policies: To evaluate CC-Topo, we use the Baseline allocation policy from Chapter 3 which is a naive strategy based on the lowest available GPU ID. Next, we evaluate CC-Topo described in Section 4.3.2 against MAPA’s Preserve Allocation Policy described in Section 3.3.5. The preserve policy allocates GPUs based on user-provided topology information whereas CC-Topo chooses a suitable topology along with GPU selection.

Evaluation on DGX-V: Figures 4.12a and 4.12b shows the predicted Bandwidth and Execution times observed for each of the jobs that ran in a given allocation policy. We noticed that the Overall predicted Bandwidth in Figure 4.12a for 25th percentile, 50th percentile (median), and 75th percentile of the jobs were better than the Baseline for both MAPA and CC-Topo. Although the Predicted Bandwidth for both *Preserve* and *CC-*



(a) Predicted Bandwidth distribution



(b) Execution time distribution

Figure 4.12 CC-Topo evaluation on DGX-V

Topo remained similar, we notice that the 25th percentile and 50th percentile (median) of the jobs were better in *CC-Topo* compared to both Baseline and *Preserve* Allocation Policy. Overall, we observe that by adapting collective communication topology when making allocation decisions can lead to better execution time in real-world workloads.

4.5 Conclusion

In this chapter, we motivated the difference in Tree and Ring Bandwidths in different Allocations on a DGX-V system. Additionally, enabling the Allocation policy to determine application topology at runtime using *CC-Topo*. We extended MAPA allocation policies to choose allocations with a collective communication topology. Finally, an evaluation of the real-world DGX-V system showed reduced execution time in more than 50 % of the jobs over baseline and MAPA’s *Preserve* allocation policy.

CC-Topo relies on independently scoring each of the possible combinations of nodes on the hardware graph. Using Graph Attention Networks to accurately predict the allocation could be an interesting future direction.

Chapter 5

Conclusion & Future Work

In this dissertation, we motivated the problem with fragmentation due to the most commonly used allocation strategies. We proposed three solutions to alleviate the problems due to low-bandwidth interconnects. First, Workload Optimization Through Inter-GPU Re-routing (WOTIR), a low-overhead solution that re-routed the inter-GPU communication traffic through an intermediate GPU. Second, a Multi-accelerator pattern allocation (MAPA) system based on generalized Graph Pattern Matching techniques. Third, Adaptive Collective Communication Topology (CC-Topo) finds an allocation along with a suitable collective communication topology. The solutions were evaluated on a real-world system with real-world scenarios.

WOTIR was able to reduce the execution time of Machine Learning training jobs by up to 34% with negligible overhead on the routing GPU. MAPA was able to reduce the worst-case execution time by 35% over baseline. MAPA was also evaluated against novel hardware topologies to analyze the benefits and limitations of allocation policies. CC-Topo

eliminated the user’s burden to provide the collective communication topology and reduced the execution time of the 50th percentile of the jobs over MAPA.

This dissertation motivated the effects of fragmentation in a single node multi-GPU system. It can be interesting to explore the effect of fragmentation in a multi-node multi-GPU environment. Extending the allocation policy for multi-node environments may require collective communication topology to be a tree, ring, or combination of both.

Furthermore, the effect of fragmentation may not be restricted to collective communication operations in GPUs. Using CUDA graphs, HIP graphs, DAGEE, and SambaFlow have enabled the implementation of application workflows as graphs in GPUs and other accelerators. It can be worthwhile to study the effect of fragmentation in such workflows.

Bibliography

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
- [2] Mohammad Abdel-Majeed, Daniel Wong, and Murali Annavaram. Warped gates: Gating aware scheduling and power gating for gpgpus. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 111–122, 2013.
- [3] Mohammad Abdel-Majeed, Daniel Wong, Justin Kuang, and Murali Annavaram. Origami: Folding warps for energy efficient gpus. In *Proceedings of the 2016 International Conference on Supercomputing, ICS '16*, 2016.
- [4] AmirAli Abdolrashidi, Hodjat Asghari Esfeden, Ali Jahanshahi, Kaustubh Singh, Nael Abu-Ghazaleh, and Daniel Wong. Blockmaestro: Enabling programmer-transparent task-based execution in gpu systems. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 333–346. IEEE, 2021.
- [5] AmirAli Abdolrashidi, Devashree Tripathy, Mehmet Esat Belviranli, Laxmi Narayan Bhuyan, and Daniel Wong. Wireframe: Supporting data-dependent parallelism through dependency graph execution in gpus. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 600–611, 2017.
- [6] Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, Temesghen Kahsai, Garrin Kimmell, et al. Think fast: a tensor streaming processor (tsp) for accelerating deep learning workloads. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 145–158. IEEE, 2020.
- [7] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in Neural Information Processing Systems*, 34:4699–4711, 2021.

- [8] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. Topology-aware gpu scheduling for learning workloads in cloud environments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2017.
- [9] Amazon. Amazon ec2 elastic gpus, 2019. Accessed 04-20-2019.
- [10] Advanced Micro Devices (AMD). Rocm communication collectives library, 2021.
- [11] Ammar Ahmad Awan, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K Panda. Optimized broadcast for deep learning workloads on dense-gpu infiniband clusters: Mpi or nccl? In *Proceedings of the 25th European MPI Users’ Group Meeting*, 2018.
- [12] Ammar Ahmad Awan, Khaled Hamidouche, Akshay Venkatesh, and Dhabaleswar K Panda. Efficient large message broadcast using nccl and cuda-aware mpi for deep learning. In *Proceedings of the 23rd European MPI Users’ Group Meeting*, pages 15–22, 2016.
- [13] Ammar Ahmad Awan, Khaled Hamidouche, Akshay Venkatesh, and Dhabaleswar K Panda. Efficient large message broadcast using nccl and cuda-aware mpi for deep learning. In *Proceedings of the 23rd European MPI Users’ Group Meeting*, 2016.
- [14] Ramin Bashizade, Yuxuan Li, and Alvin R. Lebeck. Adaptive simultaneous multi-tenancy for gpus. In Dalibor Klusáček, Walfredo Cirne, and Narayan Desai, editors, *Job Scheduling Strategies for Parallel Processing*, pages 83–106, Cham, 2019. Springer International Publishing.
- [15] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. Groute: An asynchronous multi-gpu programming model for irregular computations. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2017.
- [16] Noah D Brenowitz, Brian Henn, Jeremy McGibbon, Spencer K Clark, Anna Kwa, W Andre Perkins, Oliver Watt-Meyer, and Christopher S Bretherton. Machine learning climate model dynamics: Offline versus online performance. *arXiv preprint arXiv:2011.03081*, 2020.
- [17] V. Carletti, P. Foggia, A. Saggese, and M. Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):804–818, 2018.
- [18] Guoyang Chen, Yue Zhao, Xipeng Shen, and Huiyang Zhou. Effisha: A software framework for enabling efficient preemptive scheduling of gpu. *SIGPLAN Not.*, 52(8):3–16, January 2017.
- [19] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

- [20] Marcus Chow, Kiran Ranganath, Robert Lérias, Mika Shanela Carodan, and Daniel Wong. Energy efficient task graph execution using compute unit masking in gpus. In *2021 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA)*, pages 45–51. IEEE, 2021.
- [21] Tom Deakin, James Price, Matt Martineau, and Simon McIntosh-Smith. Evaluating attainable memory bandwidth of parallel programming models via babelstream. *International Journal of Computational Science and Engineering*, 17(3):247–262, 2018.
- [22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [23] Wenqian Dong, Zhen Xie, Gokcen Kestor, and Dong Li. Smart-pgsim: using neural network to accelerate ac-opf power grid simulation. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.
- [24] Alexandru Duțu, Matthew D Sinclair, Bradford M Beckmann, David A Wood, and Marcus Chow. Independent forward progress of work-groups. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1022–1035. IEEE, 2020.
- [25] Steffen Ernsting and Herbert Kuchen. Algorithmic skeletons for multi-core, multi-gpu systems and clusters. *International Journal of High Performance Computing and Networking*, 7(2):129–138, 2012.
- [26] Facebook. Facebook flexible gpu expander big basin refresh, 2018.
- [27] Iman Faraji, Seyed H Mirsadeghi, and Ahmad Afsahi. Topology-aware gpu selection on multi-gpu nodes. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 712–720. IEEE, 2016.
- [28] Denis Foley. Nvlink, pascal and stacked memory: Feeding the appetite for big data, 2014. Accessed 04-20-2019.
- [29] Geoffrey Fox, James A Glazier, JCS Kadupitiya, Vikram Jadhao, Minje Kim, Judy Qiu, James P Sluka, Endre Somogyi, Madhav Marathe, Abhijin Adiga, et al. Learning everywhere: Pervasive machine learning for effective high-performance computation. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 422–429. IEEE, 2019.
- [30] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users’ Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

- [31] Nitin A. Gawande, Joshua B. Landwehr, Jeff A. Daily, Nathan R. Tallent, Abhinav Vishnu, and Darren J. Kerbyson. Scaling deep learning workloads: Nvidia dgx-1/pascal and intel knights landing. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 399–408, 2017.
- [32] Yiannis Georgiou, Emmanuel Jeannot, Guillaume Mercier, and Adèle Villiermet. Topology-aware job mapping. *The International Journal of High Performance Computing Applications*, 32(1):14–27, 2018.
- [33] Google. Google cloud: Cloud gpus, 2020. Accessed 04-16-2020.
- [34] William D Gropp, William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.
- [35] Khaled Hamidouche, Akshay Venkatesh, Ammar Ahmad Awan, Hari Subramoni, Ching-Hsiang Chu, and Dhabaleswar K. Panda. Exploiting gpudirect rdma in designing high performance openshmem for nvidia gpu clusters. In *CLUSTER*, pages 78–87. IEEE Computer Society, 2015.
- [36] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.
- [37] Jonathan Hines. Stepping up to summit. *Computing in Science & Engineering*, 20(2):78–82, 2018.
- [38] NVIDIA Inc. Nvidia dgx-1: The essential instrument for ai research: Spec sheet, 2019.
- [39] NVIDIA Inc. Nvidia dgx-2: The world most powerful deep learning system for the most complex ai challenges: Spec sheet, 2019.
- [40] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, and Will Tebbutt. A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*, 2019.
- [41] Ali Jahanshahi, Hadi Zamani Sabzi, Chester Lau, and Daniel Wong. Gpu-nest: Characterizing energy efficiency of multi-gpu inference servers. *IEEE Computer Architecture Letters*, 19(2):139–142, 2020.
- [42] Kasra Jamshidi, Rakesh Mahadasa, and Keval Vora. Peregrine: A pattern-aware graph mining system. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys ’20, New York, NY, USA, 2020. Association for Computing Machinery.
- [43] Sylvain Jeaugey. Massively scale your deep learning training with nccl 2.4, 2019.

- [44] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant {GPU} clusters for {DNN} training workloads. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 947–960, 2019.
- [45] Myeongjae Jeon, Shivaram Venkataraman, Junjie Qian, Amar Phanishayee, Wencong Xiao, and Fan Yang. Multi-tenant gpu clusters for deep learning workloads: Analysis and implications. *Tech. Rep.*, 2018.
- [46] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.
- [47] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM ’14, pages 675–678, New York, NY, USA, 2014. ACM.
- [48] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datascenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.
- [49] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao. Machine learning for performance enhancement of molecular dynamics simulations. In *International Conference on Computational Science*, pages 116–130. Springer, 2019.
- [50] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao. Machine learning for parameter auto-tuning in molecular dynamics simulations: Efficient dynamics of ions near polarizable nanoparticles. *The International Journal of High Performance Computing Applications*, 34(3):357–374, 2020.
- [51] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- [52] Shinpei Kato, Karthik Lakshmanan, Aman Kumar, Mihir Kelkar, Yutaka Ishikawa, and Ragnathan Rajkumar. Rgem: A responsive gpgpu execution model for runtime engines. In *2011 IEEE 32nd Real-Time Systems Symposium*, 2011.
- [53] Riaz Ullah Khan, Xiaosong Zhang, Rajesh Kumar, and Emelia Opoku Aboagye. Evaluating the performance of resnet model based on image recognition. In *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, ICCAI 2018*, page 86–90, New York, NY, USA, 2018. Association for Computing Machinery.
- [54] Robin Kobus, Daniel Jünger, Christian Hundt, and Bertil Schmidt. Gossip: Efficient communication primitives for multi-gpu systems. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–10, 2019.

- [55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [56] A. Li, S. Song, J. Chen, J. Li, X. Liu, N. Tallent, and K. J. Barker. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems*, 2019.
- [57] A. Li, S. Song, J. Chen, J. Li, X. Liu, N. Tallent, and K. J. Barker. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems*, 2019.
- [58] Ang Li, Tong Geng, Tianqi Wang, Martin Herbordt, Shuaiwen Leon Song, and Kevin Barker. Bstc: A novel binarized-soft-tensor-core design for accelerating bit-based approximated neural nets. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–30, 2019.
- [59] Ang Li, Weifeng Liu, Linnan Wang, Kevin Barker, and Shuaiwen Leon Song. Warp-consolidation: A novel execution model for gpus. In *Proceedings of the 2018 International Conference on Supercomputing*, pages 53–64, 2018.
- [60] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. Evaluating modern GPU interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *CoRR*, abs/1903.04611, 2019.
- [61] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. Tartan: Evaluating modern gpu interconnect via a multi-gpu benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 191–202, 2018.
- [62] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. Tartan: Evaluating modern gpu interconnect via a multi-gpu benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018.
- [63] Ang Li, Shuaiwen Leon Song, Akash Kumar, Eddy Z Zhang, Daniel Chavarría-Miranda, and Henk Corporaal. Critical points based register-concurrency autotuning for gpus. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1273–1278. IEEE, 2016.
- [64] Zhenhong Liu, Daniel Wong, and Nam Sung Kim. Load-triggered warp approximation on gpu. In *Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED ’18*, 2018.
- [65] Xi Luo, Wei Wu, George Bosilca, Thananon Patinyasakdikul, Linnan Wang, and Jack Dongarra. Adapt: an event-based adaptive collective communication framework. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, 2018.

- [66] Daniel Mawhirter and Bo Wu. Automine: Harmonizing high-level abstraction and high performance for graph mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 509–523, New York, NY, USA, 2019. Association for Computing Machinery.
- [67] Veronica Melesse Vergara and et.al. Scaling the summit: Deploying the worlds fastest supercomputer. *International Workshop on OpenPOWER for HPC*, 6 2019.
- [68] Microsoft. Microsoft azure nc virtual machines—gpu compute, November 2017.
- [69] Saiful A Mojumder, Marcia S Louis, Yifan Sun, Amir Kavvan Ziabari, José L Abellán, John Kim, David Kaeli, and Ajay Joshi. Profiling dnn workloads on a volta-based dgx-1 system. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 122–133. IEEE, 2018.
- [70] J. R. Neely and B. R. de Supinski. Application modernization at llnl and the sierra center of excellence. *Computing in Science Engineering*, 19(5):9–18, 2017.
- [71] Nimbix. Nimbix dgx cloud.
- [72] NVidia. Tuning cuda applications for volta. Accessed 04-20-2019.
- [73] Nvidia. Nvidia dgx-1 with tesla v100 system architecture. <http://images.nvidia.com/content/pdf/dgx1-v100-system-architecture-whitepaper.pdf>, 2017. Accessed 04-20-2019.
- [74] Nvidia. Multi-gpu programming models, 2019.
- [75] Nvidia. Nvidia docker containers, 2021.
- [76] Nvidia. Nvidia multi-instance gpu, 2021.
- [77] Nvidia. Optimized primitives for collective multi-gpu communication, 2021.
- [78] Tesla NVIDIA. P100 white paper. *NVIDIA Corporation*, 2016.
- [79] Tesla NVIDIA. V100 white paper. *NVIDIA Corporation*, 2017.
- [80] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, October 2004.
- [81] Chandandeep Singh Pabla. Completely fair scheduler. *Linux Journal*, 2009(184):4, 2009.
- [82] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. Chimera: Collaborative preemption for multitasking on a shared gpu. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

- [83] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. Chimera: Collaborative preemption for multitasking on a shared gpu. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, page 593–606, New York, NY, USA, 2015. Association for Computing Machinery.
- [84] Sam Partee, Matthew Ellis, Alessandro Rigazzi, Scott Bachman, Gustavo Marques, Andrew Shao, and Benjamin Robbins. Using machine learning at scale in hpc simulations with smartsim: An application to ocean climate modeling. *arXiv preprint arXiv:2104.09355*, 2021.
- [85] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch. *Computer software. Vers. 0.3*, 1, 2017.
- [86] J Luc Peterson, Rushil Anirudh, Kevin Athey, Benjamin Bay, Peer-Timo Bremer, Vic Castillo, Francesco Di Natale, David Fox, Jim A Gaffney, David Hysom, et al. Merlin: enabling machine learning-ready hpc ensembles. *arXiv preprint arXiv:1912.02892*, 2019.
- [87] Sreeram Potluri, Hao Wang, Devendar Bureddy, Ashish Kumar Singh, Carlos Rosales, and Dhabaleswar K Panda. Optimizing mpi communication on multi-gpu systems using cuda inter-process communication. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 2012.
- [88] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.
- [89] Kiran Ranganath, AmirAli Abdolrashidi, Shuaiwen Leon Song, and Daniel Wong. Speeding up collective communications through inter-gpu re-routing. *IEEE Computer Architecture Letters*, 18(2):128–131, 2019.
- [90] Kiran Ranganath, Jesun Firoz, Joshua Suetterlein, Joseph Manzano, Andres Marquez, Mark Raugas, and Daniel Wong. Lc-memento: A memory model for accelerated architectures. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 67–82. Springer, 2022.
- [91] Kiran Ranganath, Joshua D Suetterlein, Joseph B Manzano, Shuaiwen Leon Song, and Daniel Wong. Mapa: Multi-accelerator pattern allocation policy for multi-tenant gpu servers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.
- [92] M Wasiur RASHID, Gary Ward, Wei-Je Robert Huang, and Philip Browning Johnson. Managing copy operations in complex processor topologies, April 30 2019. US Patent 10,275,275.

- [93] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549*, 2019.
- [94] Baidu Research. Baidu all-reduce, 2017.
- [95] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2002.
- [96] Dipanjan Sengupta, Raghavendra Belapure, and Karsten Schwan. Multi-tenancy on gpgpu-based servers. In *Proceedings of the 7th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '13, page 3–10, New York, NY, USA, 2013. Association for Computing Machinery.
- [97] Dipanjan Sengupta, Anshuman Goswami, Karsten Schwan, and Krishna Pallavi. Scheduling multi-tenant cloud workloads on accelerator-based systems. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 513–524. IEEE, 2014.
- [98] Shayan Shams, Richard Platania, Kisung Lee, and Seung-Jong Park. Evaluation of deep learning frameworks over different hpc architectures. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.
- [99] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyounJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh-TensorFlow: Deep learning for supercomputers. In *Neural Information Processing Systems*, 2018.
- [100] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- [101] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [102] Misha Smelyanskiy. Zion: Facebook next-generation large memory training platform. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–22. IEEE Computer Society, 2019.
- [103] Shuaiwen Song, Matthew Grove, and Kirk W Cameron. An iso-energy-efficient approach to scalable system power-performance optimization. In *2011 IEEE International Conference on Cluster Computing*, pages 262–271. IEEE, 2011.
- [104] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In

- Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17*, page 4278–4284. AAAI Press, 2017.
- [105] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
 - [106] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
 - [107] Nathan R Tallent, Nitin A Gawande, Charles Siegel, Abhinav Vishnu, and Adolfo Hoisie. Evaluating on-node gpu interconnects for deep learning workloads. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2017.
 - [108] Nathan R Tallent and Adolfo Hoisie. Palm: Easing the burden of analytical performance modeling. In *Proceedings of the 28th ACM international conference on Supercomputing*, pages 221–230, 2014.
 - [109] Jingweijia Tan, Shuaiwen Leon Song, Kaige Yan, Xin Fu, Andres Marquez, and Darren Kerbyson. Combating the reliability challenge of gpu register file at low supply voltage. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 3–15. IEEE, 2016.
 - [110] Ivan Tanasic, Isaac Gelado, Javier Cabezas, Alex Ramirez, Nacho Navarro, and Mateo Valero. Enabling preemptive multiprogramming on gpus. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014.
 - [111] Ivan Tanasic, Isaac Gelado, Javier Cabezas, Alex Ramirez, Nacho Navarro, and Mateo Valero. Enabling preemptive multiprogramming on gpus. In *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ISCA '14, page 193–204. IEEE Press, 2014.
 - [112] Siamak Tavallaei and Robert Ober. Microsoft project olympus hyperscale gpu accelerator(hgx-1), 2017. Accessed 04-20-2019.
 - [113] Carlos H. C. Teixeira, Alexandre J. Fonseca, Marco Serafini, Georgos Siganos, Mohammed J. Zaki, and Ashraf Abounnaga. Arabesque: A system for distributed graph mining. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP '15*, page 425–440, New York, NY, USA, 2015. Association for Computing Machinery.
 - [114] Devashree Tripathy, Amirali Abdolrashidi, Laxmi Narayan Bhuyan, Liang Zhou, and Daniel Wong. Paver: Locality graph-based thread block scheduling for gpus. *ACM Transactions on Architecture and Code Optimization (TACO)*, 18(3):1–26, 2021.
 - [115] Devashree Tripathy, Amirali Abdolrashidi, Quan Fan, Daniel Wong, and Manoranjan Satpathy. Localityguru: A ptx analyzer for extracting thread block-level locality in

- gpgpus. *Proceedings of the 15th IEEE/ACM International Conference on Networking, Architecture, and Storage*, 2021 (To appear).
- [116] Devashree Tripathy, Hadi Zamani, Debiprasanna Sahoo, Laxmi N Bhuyan, and Manoranjan Satpathy. Slumber: static-power management for gpgpu register files. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 109–114, 2020.
 - [117] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976.
 - [118] Julian R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *ACM J. Exp. Algorithmics*, 15, February 2011.
 - [119] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O’Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC ’13*, New York, NY, USA, 2013. Association for Computing Machinery.
 - [120] Carl A Waldspurger and William E Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, pages 1–es, 1994.
 - [121] Guanhua Wang, Amar Phanishayee, Shivaram Venkataraman, and Ion Stoicat. Blink: A fast nvlink-based collective communication library. In *Conference on Systems and Machine Learning (SysML)*, 2018.
 - [122] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil Devanur, and Ion Stoica. Blink: Fast and generic collectives for distributed ml. *arXiv preprint arXiv:1910.04940*, 2019.
 - [123] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K Panda. Mvapich2-gpu: optimized gpu to gpu communication for infiniband clusters. *Computer Science-Research and Development*, 26(3-4):257, 2011.
 - [124] Lijing Wang, Jiangzhuo Chen, and Madhav Marathe. Defsi: Deep learning based epidemic forecasting with synthetic information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9607–9612, 2019.
 - [125] Jack Wells. Scaling deep learning workloads on the summit supercomputer. <https://www.youtube.com/watch?v=T10ig6NFWg8>, 2018. GTC.
 - [126] Emma White. Optimizing deep learning on p3 and p3dn with efa, 2019.
 - [127] Jeremiah J Wilke, Joseph P Kenny, Samuel Knight, and Sebastien Rumley. Compiler-assisted source-to-source skeletonization of application models for system simulation. In *International Conference on High Performance Computing*, pages 123–143. Springer, 2018.

- [128] Daniel Wong, Nam Sung Kim, and Murali Annavaram. Approximating warps with intra-warp operand value similarity. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 176–187, 2016.
- [129] Bo Wu, Xu Liu, Xiaobo Zhou, and Changjun Jiang. Flep: Enabling flexible and efficient preemption on gpus. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '17*, page 483–496, New York, NY, USA, 2017. Association for Computing Machinery.
- [130] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.
- [131] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 595–610, 2018.
- [132] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh. Modular routing design for chiplet-based systems. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 726–738, 2018.
- [133] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.
- [134] Hadi Zamani, Yuanlai Liu, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. Greenmm: energy efficient gpu matrix multiplication through undervolting. In *Proceedings of the ACM International Conference on Supercomputing*, pages 308–318, 2019.
- [135] Hadi Zamani, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. Saou: Safe adaptive overclocking and undervolting for energy-efficient gpu computing. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 205–210, 2020.