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

Kiran Ranganath
Department of Electrical and Computer Engineering
University of California Riverside
CA, USA
krang006@ucr.edu

Joshua D. Suetterlein
High-Performance Computing Group
Pacific Northwest National Lab
WA, USA
joshua.suetterlein@pnnl.gov

Joseph B. Manzano
High-Performance Computing Group
Pacific Northwest National Lab
WA, USA
joseph.manzano@pnnl.gov

Shuaiwen Leon Song
Future System Architecture Lab
School of Computer Science
University of Sydney
Sydney, Australia
shuaiwen.song@sydney.edu.au

Daniel Wong
Department of Electrical and Computer Engineering
University of California Riverside
CA, USA
danwong@ucr.edu

ABSTRACT

Multi-accelerator servers are increasingly being deployed in shared multi-tenant environments (such as in cloud data centers) in order to meet the demands of large-scale compute-intensive workloads. In addition, these accelerators are increasingly being inter-connected in complex topologies and workloads are exhibiting a wider variety of inter-accelerator communication patterns. However, existing allocation policies are ill-suited for these emerging use-cases. Specifically, this work identifies that multi-accelerator workloads are commonly *fragmented* leading to reduced bandwidth and increased latency for inter-accelerator communication.

We propose Multi-Accelerator Pattern Allocation (MAPA), a graph pattern mining approach towards providing generalized allocation support for allocating multi-accelerator workloads on multi-accelerator servers. We demonstrate that MAPA is able to improve the execution time of multi-accelerator workloads and that MAPA is able to provide generalized benefits across various accelerator topologies. Finally, we demonstrate a speedup of 12.4% for 75th percentile of jobs with the worst case execution time reduced by up to 35% against baseline policy using MAPA.

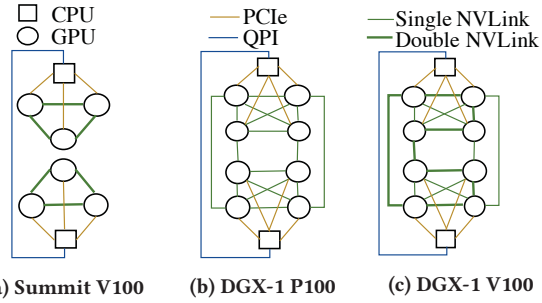
ACM Reference Format:
Kiran Ranganath, Joshua D. Suetterlein, Joseph B. Manzano, Shuaiwen Leon Song, and Daniel Wong. 2021. MAPA: Multi-Accelerator Pattern Allocation Policy for Multi-Tenant GPU Servers. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3458817.3480853

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11.
https://doi.org/10.1145/3458817.3480853

1 INTRODUCTION

The never ending demand for faster computation from data intensive workloads has driven the growth for multi-accelerator servers. Systems equipped with accelerators, such as General Purpose Processing in Graphical Processing Units (GPGPUs) and Tensor Processing Units (TPU) [29] 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.



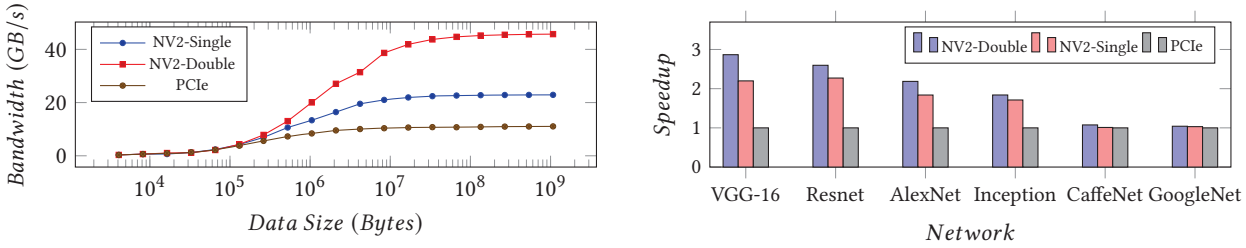
(a) Summit V100 **(b) DGX-1 P100** **(c) DGX-1 V100**

Figure 1: Emerging multi-GPU accelerator topologies are increasingly heterogeneous.

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

Two major challenges of modern large-scale systems are the need for faster collective communication operations [51, 67] and topology-aware scheduling [7, 72]. Recent works like topology-aware scheduling [7] and Gandiva [72] have motivated the importance of optimal placements to improve performance of Machine Learning (ML) workloads within multi-GPU environments by efficiently utilizing inter-accelerator interconnection link. Systems such as Nvidia's DGX-V100, Facebook's Big-Basin [17], and Amazon's P3DN [69] have accelerators connected with many different types of interconnection links.

In this work, we focus on the challenge of inefficient job allocation in a multi-accelerator environment. These sub-par allocations



(a) Bandwidth characterization

(b) Speedup with different links

Figure 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 bandwidth as well as impact on applications such as CNN training. where NV2-Single and NV2-Double are Single and Double NVLink-v2 links respectively.

can lead to 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 1b), Facebook's Big-Basin systems [17], Amazon's P3DN [69] and DGX-1-V (Figure 1c). Even designs with constant access latency such as the DGX-2 exhibit NUMA effects [37] which can lead to allocation inefficiencies. Furthermore, new accelerator designs such as TPUs [29] and multi-chip accelerators [73] 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 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 paper 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 larger scale and complex non-uniform topologies.

2 MOTIVATION

Increasingly more popular cloud [8, 17, 20, 22, 23] and modern HPC systems [21] are accelerator based, and are used to train and to 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 [29]. The following describes some of the challenges posed by these multi-accelerator architectures.

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 1 lists the types of links used to connect accelerators in these systems and their respective bandwidths.

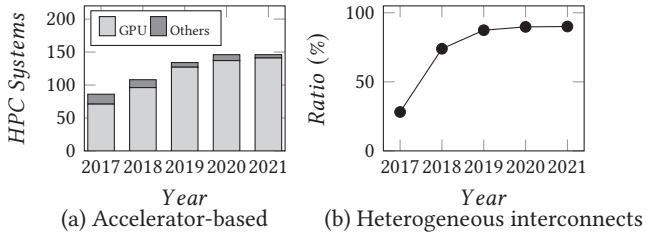
Link	Bandwidth (GBps)
Single NVlink-v1	20
Single NVlink-v2	25
Double NVlink-v2	50
16-lanes PCIe Gen 3 [46]	12

Table 1: Peak Bandwidths per link

In systems like Big basin [17], P3DN [69], Summit [21], DGX-1 V100 [23], and DGX-1 P100 [22], accelerators are not uniformly connected. For example, in earlier generations 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 includes NVidia Collective Communication Libraries (NCCL) [45], AMD's Radeon Collective Communication Library (RCCL) [9], and Baidu All-Reduce [52].

We observe from Figure 3 (a) that supercomputers are increasingly employing discrete GPUs. Figure 3 (b) shows the increased presence of heterogeneous interconnects in such systems. Hence, it is important to identify and explore allocation challenges in such compute environments. Additionally, machine learning-based workloads has recently gained attention in the HPC community, with efforts such as Mesh-tensorflow [54] and Zero [50] 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 [11, 14, 19, 30, 31, 48, 49, 68].

In Figure 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



(a) Accelerator-based (b) Heterogeneous interconnects

Figure 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 are now dominant.

achieve lower bandwidth, the relative performance of each link type to each other remains, with double NVLink being the fastest.

In Figure 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 allocation of high-bandwidth links is critical for workloads with larger data transfers.

Multi-tenant Multi-Accelerator Servers. It was shown in Philly [26] and Gandiva [72] 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) [2] 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 requires a further in-depth exploration to ensure that the loss in these metrics is acceptable for these applications.

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 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 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, 2 Double NVLinks).

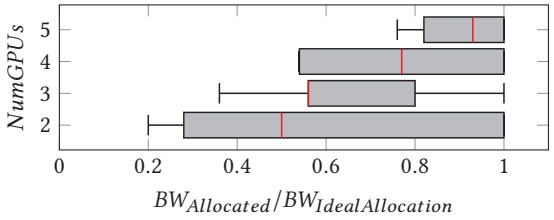


Figure 4: Due to fragmentation of GPU allocations, a large portion of GPU jobs have sub-optimal allocated aggregate bandwidth ($BW_{Allocated}$) compared to the aggregate bandwidth of an ideal allocation ($BW_{IdealAllocation}$).

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 4 for experimental methodology details). The box-plot shows the distribution of bandwidth allocation quality.

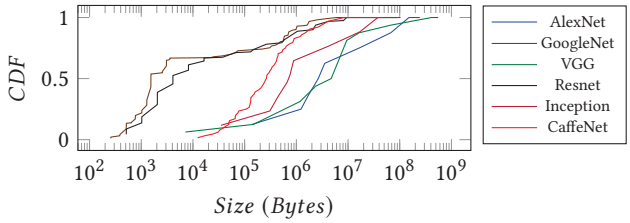
We observe that a large majority of jobs receive suboptimal allocations. It should be noted that smaller jobs with less GPUs suffer more due to the potential for being spread out more across the interconnect topology. For example, with 3 GPU jobs, 75% of jobs experience allocations with 20% less bandwidth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

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 [26, 72]. While these workloads are characterized to be very compute intensive, they have different degrees of sensitivity to the bandwidth provided by the system.

Figure 5(a) shows the distribution of data sizes that are communicated during the synchronization phase of ML training. Figure 5(b) shows the number of collective communication calls per GPU that is employed in training these networks. We can infer from Figure 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 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 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 are 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.

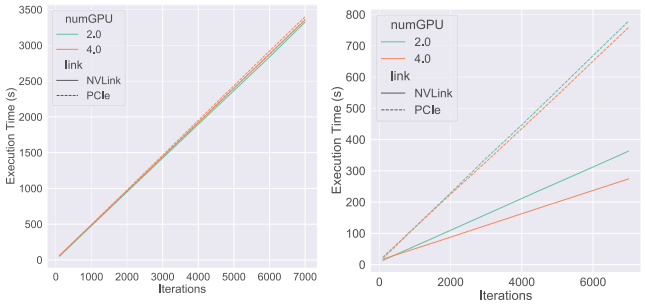


(a) Cumulative distribution of collective communication calls

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) Number of communication calls triggered per GPU per iteration and bandwidth sensitivity

Figure 5: Communication Properties of ML workloads



(a) GoogleNet (Insensitive) **(b) VGG-16 (Sensitive)**

Figure 6: Execution Time trends of Bandwidth Sensitive and Insensitive Networks.

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 number of GPUs and iterations as shown in Figure 6a and Figure 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 slowdown ML training jobs by more than 50% as shown in Figure 2(b). A solution that could potentially avoid the scenarios like this could improve 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 resource fragmented. Existing job allocation polices are unaware of the hardware diversity leading to a misappropriation of bandwidth to jobs. Popular workloads such as ML training can be particularly susceptible to poor allocations. Clearly, 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 MAPA: MULTI-ACCELERATOR PATTERN ALLOCATION

The Multi-Accelerator Pattern Allocation (MAPA) framework introduces a generalized solution towards 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 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.

3.1 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 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.

Application communication patterns can be manually specified by the programmer, or can be automatically extracted through program analysis or profiling [16, 18, 59, 70]. 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 9a illustrates this through a code sample from Caffe which performs the training operation of a layer. In this example, a collective all

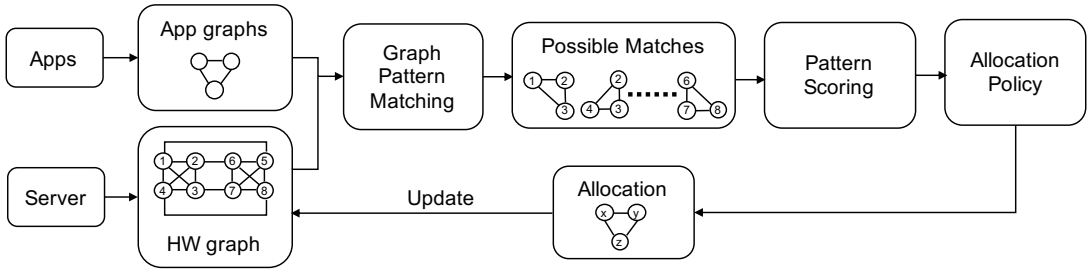


Figure 7: Overview of the Multi-Accelerator Pattern Allocation(MAPA) system

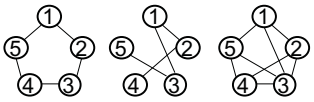


Figure 8: Example application topology for 5-GPU workload utilizing NCCL collective communication for inter-GPU communication. Application topologies can be ring (left), tree (middle), or a combination of both (right).

```
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 9: Examples of identifying application topology through source code analysis and runtime profiling.

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 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 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 application pair a single MPI rank to a single GPU and use MPI calls to communication across ranks. With CUDA-aware MPI [10], these GPU-GPU communication can be handled directly through NVLink without going through the host. While source code analysis of MPI calls can explicitly identify the communication pattern, many recent works have aimed to automatically identify MPI application topologies [16, 18], or automatically identify communication through compiler-assisted skeletons [59, 70].

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` tracks the amount of traffic sent over each NVlink. Figure 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.

3.2 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 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 amount 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 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 [47]. Several well-known algorithm exist in solving this problem, such as Ullmann's algorithm [65, 66], VF2 [47], and VF3 [12]. Since the goal of this paper 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 [61], AutoMine [42], and Peregrine [25]. 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, are currently emerging. For example, GPUs can be virtualized for multi-tenancy [53] 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 [53].

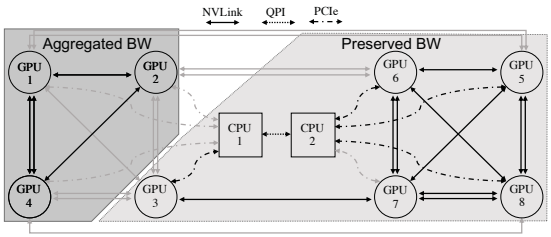


Figure 10: Illustrative example showing bandwidths used for Aggregated Bandwidth score calculation (left) and Preserved Bandwidth calculation (right) given an allocation [1, 2, 4]

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?*

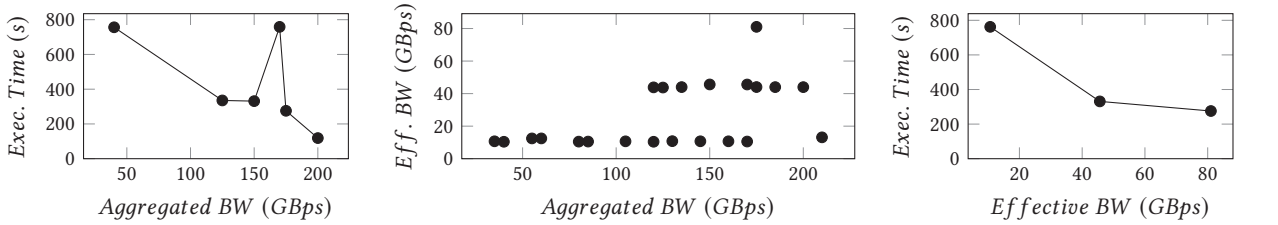
3.4.1 Pattern scoring metrics. To find a suitable pattern scoring metric, we explore two proposed metrics called *Aggregated Bandwidth (AggBW)* and *Effective Bandwidth (EffBW)*.

Aggregated Bandwidth: We define *Aggregated Bandwidth (AggBW)* as the total allocated bandwidth in the matching pattern M that is used by the application pattern 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 1.

$$AggBW = \sum_{e \in (E(P) \cap E(M))} w(e), \tag{1}$$

where $E(P) \cap E(M)$ represents the allocated interconnect in the matching pattern M that are 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 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 GPU 1,2 and 2,4 and 1,4.



(a) VGG-16 training Execution Time (b) Aggregated vs Effective Bandwidth (c) VGG-16 training Execution Time

Figure 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.

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

3.4.2 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 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 11(b) which is collected using microbenchmarks to measure the effective bandwidth of various allocations ranging from 2-5 GPUs. *Therefore, we find that execution time of workloads cannot be predicted by naively aggregating the allocated bandwidth. Instead, execution time of workloads must be predicted by the effective bandwidth.* Figure 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.

3.4.3 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.

$$\begin{aligned} &Predicted\ Effective\ Bandwidth = \\ &\theta_1 x + \theta_2 y + \theta_3 z + \theta_4 \frac{1}{x+1} + \theta_5 \frac{1}{y+1} + \theta_6 \frac{1}{z+1} \\ &+ \theta_7 xy + \theta_8 yz + \theta_9 zx + \theta_{10} \frac{1}{xy+1} + \theta_{11} \frac{1}{yz+1} + \theta_{12} \frac{1}{zx+1} \\ &+ \theta_{13} xyz + \theta_{14} \frac{1}{xyz+1} \end{aligned} \tag{2}$$

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 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 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 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 coefficient is tabulated in table 2.

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 2: Values of Coefficients.

Figure 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 2 we can now directly utilize *EffBW* as our pattern scoring metric without the need for microbenchmarking.

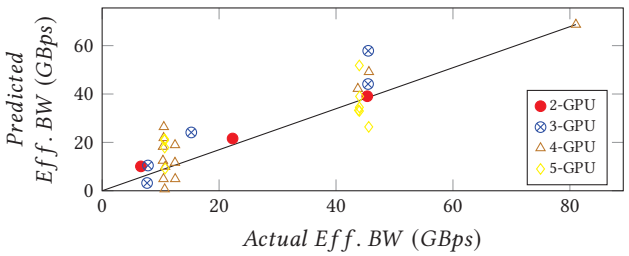


Figure 12: Predicted effective bandwidth correlates well with actual effective bandwidth and generalizes across jobs of different sizes.

3.5 Pattern Selection Allocation Policy

Once the matching patterns are scored, MAPA will then select a matching pattern for allocation. Recall form Section 2.3 and Figure 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) avoid 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 6.

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.

3.5.1 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 10 illustrates the calculation of preserved bandwidth if GPUs 1, 2, and 4 are allocated. Hence, we formally define *Preserved Bandwidth* as follows in Equation 3.

$$Preserved\ Bandwidth = \sum_{e \in E(G \setminus M)} w(e) \tag{3}$$

3.5.2 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 Predicted Effective Bandwidth ($EffBW$). If the job to be allocated is bandwidth insensitive, we allocate the matching pattern that obtains the largest *Preserved Bandwidth*. Meaning, we are preserving the amount of remaining available high-bandwidth links in the hardware graph for bandwidth sensitive allocations to avoid potentially starving these jobs. If the job to be allocated is bandwidth sensitive, we allocate the matching pattern with the highest *Predicted Effective Bandwidth*.

3.6 State Management

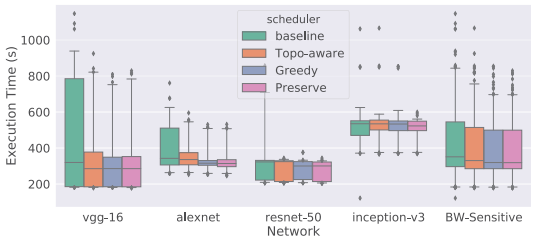
Once a matching pattern is selected for allocation, we then must update the hardware graph G. The hardware graph G is updated whenever there is an allocation (a job is scheduled) and a deallocation (a job is finished). Once an allocation is obtained, we update the hardware graph to remove the unavailable 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 was previously removed.

4 EVALUATION

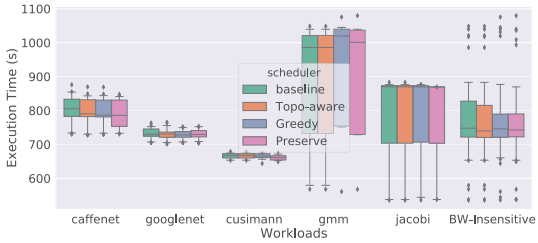
To evaluate MAPA, we use a combination of real-world runs and simulation. Specifically, we first evaluate the effectiveness of MAPA and the impact on performance on an NVIDIA DGX-1 V100 machine running on Ubuntu-16.04 with CUDA 11.3 and NCCL-2.10.3. The DGX-1 V100 hardware topology is shown in Figure 1c. The MAPA framework is built on top of Peregrine [25], 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 work we use Fist-in First-out (FIFO) for scheduling jobs from the queue.

Algorithm 1: Preserve Allocation Policy

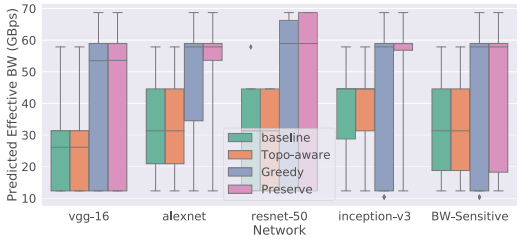
```
Result: Allocation
HWgraph hGraph;
AppGraph aGraph;
Allocation alloc = { };
Patterns possiblePatterns = graphPatternMatching (aGraph,
  hGraph);
if aGraph is bwSensitive then
    foreach pattern in possiblePatterns do
        if EffectiveBW (pattern) > EffectiveBW (alloc) then
            alloc = pattern;
        end
    end
end
else
    foreach pattern in possiblePatterns do
        if PreservedBW (pattern) > PreservedBW (alloc) then
            alloc = pattern;
        end
    end
end
```



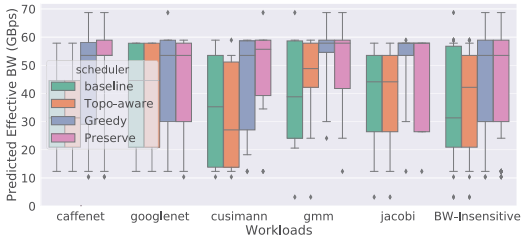
(a) Execution time of bandwidth sensitive jobs



(b) Execution time of bandwidth insensitive jobs



(c) Effective bandwidth of bandwidth sensitive jobs



(d) Effective bandwidth of bandwidth insensitive jobs

Figure 13: Evaluation results on DGX-V

Later in Section 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 [27] training jobs which makes use of multiple GPUs – AlexNet [34], VGG-16 [55], Resnet-50 [32], Inception [57], GoogleNet [58], and CaffeNet [28]. These neural networks are trained using the Image-Net dataset [13]. Each of the evaluated networks have different compute and communication patterns as discussed in Section 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) [39], Gaussian Mixture Model (GMM) [39], and a Jacobi solver [44]. Furthermore, previous works [38, 39] have demonstrated that Cusimann and GMM to have negligible inter-GPU communication during the course of execution. Furthermore, we observed less than 3% execution time improvement with Jacobi solver. Hence, we classify Cusimann, GMM, and Jacobi to be bandwidth insensitive. In this work, 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 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 [26] has shown that the number of request GPUs for multi-GPU jobs in multi-tenant environments tend 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* [7], 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 are done in existing frameworks such as Nvidia Docker [1]. The *Topo-aware* allocation policy [7] 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.

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 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. Figure 13 shows our results, separated by sensitive and insensitive workloads.

Figure 13(a) and 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 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 interconnected GPUs. This results in significantly improved tail execution times, most notably in VGG and Alexnet 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 13(c) and

(d), 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 policy– Greedy and Preserve. The MAPA Greedy policy greedily selects the allocation with the most aggregated bandwidth. Although aggregated bandwidth does not correlated with effective bandwidth, the Greedy policy nevertheless significantly improves the quality of allocation. As shown in Figure 13(c) and (d), 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 aim to preserve bandwidth for future bandwidth sensitive workloads. In Figure 13(c) we see that the Greedy policy has allocations with 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: Summary of results. Normalized execution time speed up and throughput (Tput) observed on DGX-1 V100.

The Preserve policy is able to successfully preserve bandwidth for bandwidth sensitive workloads. This policy is able to 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 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 is able to 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 is able to 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.

5 EXPLORING NOVEL HARDWARE TOPOLOGIES

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.

5.2 Simulation Framework

Details of the simulation framework is shown in Figure 14. 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* 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 the 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 MAPAto 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 15, 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 execution time of the real run for each workload. As shown in Figure 16, 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 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 17a and 17b, respectively. Although 16-GPU topologies exists with crossbar switches

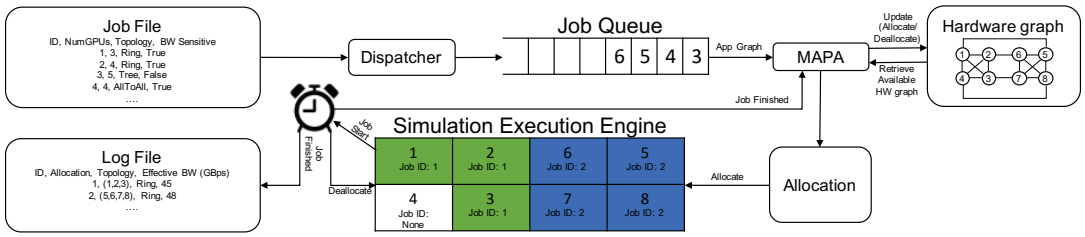


Figure 14: MAPA simulation execution framework.

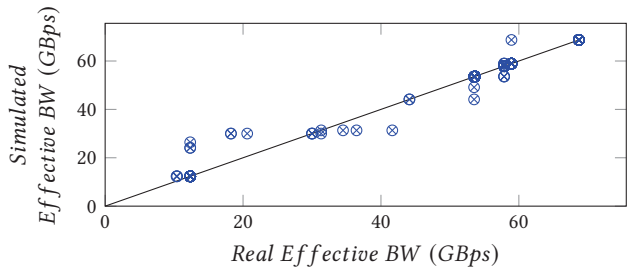


Figure 15: Effective bandwidth measured during DGX-V's real and simulation runs correlated well.

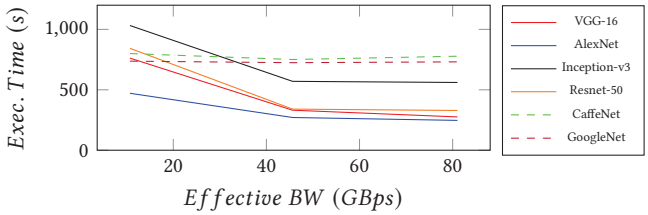
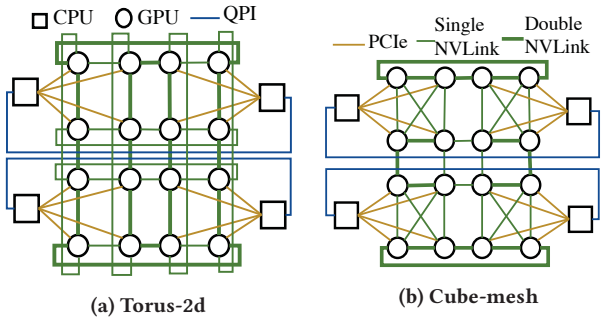


Figure 16: Effective bandwidth vs execution time observed during real runs on DGX-V.



(a) Torus-2d **(b) Cube-mesh**

Figure 17: 16-GPU topologies

(NVSwitch), we aim to explore alternative topologies consisting of cost-effective point-to-point links.

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 16.

For the 16-GPU Torus-2d (Figures 18a), 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 is able to easily select high bandwidth allocations improving the 75th percentile (making fast jobs even faster).

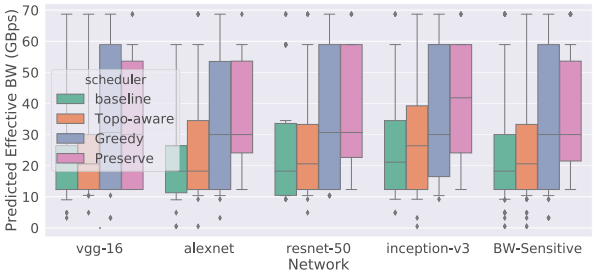
For the Cube-mesh topology (Figure 18b), it is a more irregular network and thus more difficult to greedily select optimal allocations. Here, Preserve performs even better for sensitive workloads. While the minimum effective bandwidth of Preserve is equivalent to the 25th percentile of all other workloads, 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 the all of the jobs with baseline and Topo-aware and the majority of Greedy.**

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

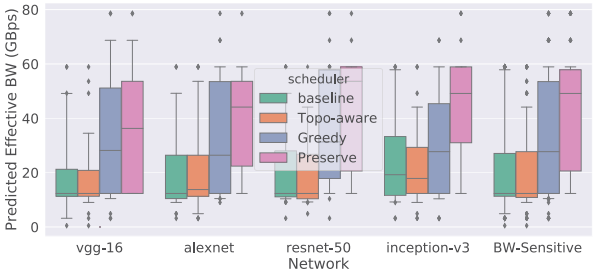
5.4 Overhead of Scheduling

Figure 19 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 size 6, 8, and 16 for Summit, DGX-V and Torus-2d/CubeMesh-16, respectively. Typically, we observe scheduling overheads in the order of milliseconds which is negligible. 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 requires 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.



(a) Torus-2d



(b) Cube Mesh

Figure 18: Simulation results for bandwidth sensitive workloads on 16-GPU topologies. Improvements to lower tail (min and 25th percentile) is better in both .

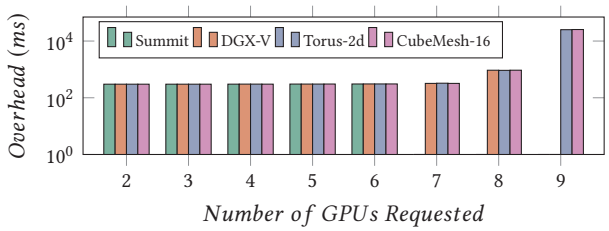


Figure 19: Overhead analysis of MAPA w/ Preserve policy.

6 RELATED WORKS

Scheduling for multi-node GPU clusters: Many works [3–6, 15, 24, 35, 36, 40, 41, 56, 60, 62–64, 71, 74, 75] have proposed optimizations to improve GPU performance and Energy efficiency. Recent works, such as Gandiva [72] and Philly [26], proposes 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* the node itself in a multi-tenant environment due to the heterogeneity of links.

Collective communication: In [33, 67], the authors have proposed techniques towards achieving efficient collective communication. Blink [67] 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 [33] proposes flow-oriented collectives and generates transfer plans to best schedule packets. Works like WOTIR [51] presents 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 sensative jobs.

Multi-GPUs for Machine Learning: From recent works [26, 43, 72], Machine Learning (ML) is one of the primary workloads on multi-GPU systems. Hence, we use ML training as a target workload in this work, as well. We used Caffe [27] framework for Machine Learning in this work. These machine learning workloads use Nvidia Collective Communications Library (NCCL) [45] 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.

7 CONCLUSION

In this work, 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 grow 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.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback and suggestions. This work was partially supported by NSF grants #1815643, #1955650, #2047521, and University of Sydney faculty startup funding and Australia Research Council (ARC) Discovery Project DP210101984.

This work was also partially funded by the U.S. Dept. of Energy's Office of Science Center for Advanced Technology Evaluation (CENATE) project under the Pacific Northwest National Laboratory. Pacific Northwest National Laboratory is operated by Battelle Memorial Institute for the U.S. Department of Energy under Contract DE-AC05-76RL01830.

REFERENCES

[1] 2021. Nvidia Docker Containers. https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

[2] 2021. NVIDIA Multi-instance GPU. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html

[3] Mohammad Abdel-Majeed, Daniel Wong, and Murali Annavaram. 2013. Warped gates: Gating aware scheduling and power gating for GPGPUs. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 111–122.

[4] Mohammad Abdel-Majeed, Daniel Wong, Justin Kuang, and Murali Annavaram. 2016. Origami: Folding Warps for Energy Efficient GPUs. In *Proceedings of the 2016 International Conference on Supercomputing (ICS '16)*.

[5] AmirAli Abdolrashidi, Hodjat Asghari Esfeden, Ali Jahanshahi, Kaustubh Singh, Nael Abu-Ghazaleh, and Daniel Wong. 2021. Blockmaestro: Enabling programmer-transparent task-based execution in gpu systems. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 333–346.

[6] AmirAli Abdolrashidi, Devashree Tripathy, Mehmet Esat Belviranli, Laxmi Narayan Bhuyan, and Daniel Wong. 2017. Wireframe: Supporting data-dependent parallelism through dependency graph execution in gpus. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 600–611.

[7] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. 2017. Topology-aware gpu scheduling for learning workloads in cloud environments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[8] Amazon. 2019. Amazon EC2 elastic GPUs. https://aws.amazon.com/ec2/elastic-gpus/ Accessed 04-20-2019.

[9] Advanced Micro Devices (AMD). 2021. ROCm Communication Collectives Library. https://github.com/ROCmSoftwarePlatform/rccl

[10] Ammar Ahmad Awan, Khaled Hamidouche, Akshay Venkatesh, and Dhabaleswar K Panda. 2016. Efficient large message broadcast using NCCL and CUDA-aware MPI for deep learning. In *Proceedings of the 23rd European MPI Users' Group Meeting*. 15–22.

[11] Noah D Brenowitz, Brian Henn, Jeremy McGibbon, Spencer K Clark, Anna Kwa, W Andre Perkins, Oliver Watt-Meyer, and Christopher S Bretherton. 2020. Machine learning climate model dynamics: Offline versus online performance. *arXiv preprint arXiv:2011.03081* (2020).

[12] V. Carletti, P. Foggia, A. Saggese, and M. Vento. 2018. 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 (2018), 804–818. https://doi.org/10.1109/TPAMI.2017.2696940

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[14] Wenqian Dong, Zhen Xie, Gokcen Kestor, and Dong Li. 2020. Smart-PGSim: using neural network to accelerate AC-OPF power grid simulation. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[15] Alexandru Duțu, Matthew D Sinclair, Bradford M Beckmann, David A Wood, and Marcus Chow. 2020. Independent forward progress of work-groups. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1022–1035.

[16] Steffen Ernsting and Herbert Kuchen. 2012. Algorithmic skeletons for multi-core, multi-GPU systems and clusters. *International Journal of High Performance Computing and Networking* 7, 2 (2012), 129–138.

[17] Facebook. 2018. Facebook Flexible GPU Expander Big Basin Refresh. https://www.opencompute.org/files/OCP2018-Facebook-Flexible-GPU-Expander-Big-Basin-Refresh-v0.7.pdf

[18] Iman Faraji, Seyed H Mirsadeghi, and Ahmad Afsahi. 2016. Topology-aware GPU selection on multi-GPU nodes. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 712–720.

[19] Geoffrey Fox, James A Glazier, JCS Kadupitiya, Vikram Jadhao, Minje Kim, Judy Qiu, James P Sluka, Endre Somogyi, Madhav Marathe, Abhijin Adiga, et al. 2019. Learning everywhere: Pervasive machine learning for effective high-performance computation. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 422–429.

[20] Google. 2020. Google Cloud: Cloud GPUs. https://cloud.google.com/gpu Accessed 04-16-2020.

[21] Jonathan Hines. 2018. Stepping up to Summit. *Computing in Science & Engineering* 20, 2 (2018), 78–82.

[22] NVIDIA Inc. 2019. NVIDIA DGX-1: The essential instrument for AI Research: Spec Sheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-1-rhel-datasheet-nvidia-us-808336-r3-web.pdf

[23] NVIDIA Inc. 2019. NVIDIA DGX-2: The World Most Powerful Deep Learning System For the Most Complex AI Challenges: Spec Sheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-2-datasheet-us-nvidia-955420-r2-web-new.pdf

[24] Ali Jahanshahi, Hadi Zamani Sabzi, Chester Lau, and Daniel Wong. 2020. GPU-NEST: Characterizing Energy Efficiency of Multi-GPU Inference Servers. *IEEE Computer Architecture Letters* 19, 2 (2020), 139–142. https://doi.org/10.1109/LCA.2020.3023723

[25] Kasra Jamshidi, Rakesh Mahadasa, and Keval Vora. 2020. Peregrine: A Pattern-Aware Graph Mining System. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 13, 16 pages. https://doi.org/10.1145/3342195.3387548

[26] Myeongjae Jeon, Shivaram Venkataraman, Junjie Qian, Amar Phanishayee, Wencong Xiao, and Fan Yang. 2018. Multi-tenant GPU clusters for deep learning workloads: Analysis and implications. *Tech. Rep.* (2018).

[27] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. 675–678.

[28] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia* (Orlando, Florida, USA) *(MM '14)*. ACM, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889

[29] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[30] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao. 2019. Machine learning for performance enhancement of molecular dynamics simulations. In *International Conference on Computational Science*. Springer, 116–130.

[31] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao. 2020. 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 (2020), 357–374.

[32] Riaz Ullah Khan, Xiaosong Zhang, Rajesh Kumar, and Emelia Opoku Aboagye. 2018. Evaluating the Performance of ResNet Model Based on Image Recognition. In *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence* (Chengdu, China) *(ICCAI 2018)*. Association for Computing Machinery, New York, NY, USA, 86–90. https://doi.org/10.1145/3194452.3194461

[33] Robin Kobus, Daniel Jünger, Christian Hundt, and Bertil Schmidt. 2019. Gossip: Efficient Communication Primitives for Multi-GPU Systems. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–10.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[35] Ang Li, Tong Geng, Tianqi Wang, Martin Herbordt, Shuaiwen Leon Song, and Kevin Barker. 2019. 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*. 1–30.

[36] Ang Li, Weifeng Liu, Linnan Wang, Kevin Barker, and Shuaiwen Leon Song. 2018. Warp-consolidation: A novel execution model for gpus. In *Proceedings of the 2018 International Conference on Supercomputing*. 53–64.

[37] A. Li, S. Song, J. Chen, J. Li, X. Liu, N. Tallent, and K. J. Barker. 2019. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems* (2019). https://doi.org/10.1109/TPDS.2019.2928289

[38] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. 2019. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *CoRR* abs/1903.04611 (2019). arXiv:1903.04611 http://arxiv.org/abs/1903.04611

[39] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. 2018. Tartan: Evaluating Modern GPU Interconnect via a Multi-GPU Benchmark Suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 191–202. https://doi.org/10.1109/IISWC.2018.8573483

[40] Ang Li, Shuaiwen Leon Song, Akash Kumar, Eddy Z Zhang, Daniel Chavarría-Miranda, and Henk Corporaal. 2016. Critical points based register-concurrency autotuning for GPUs. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1273–1278.

[41] Zhenhong Liu, Daniel Wong, and Nam Sung Kim. 2018. Load-Triggered Warp Approximation on GPU. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '18)*.

[42] Daniel Mawhirter and Bo Wu. 2019. AutoMine: Harmonizing High-Level Abstraction and High Performance for Graph Mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 509–523. https://doi.org/10.1145/3341301.3359633

[43] Saiful A Mojumder, Marcia S Louis, Yifan Sun, Amir Kavyan Ziabari, José L Abellán, John Kim, David Kaeli, and Ajay Joshi. 2018. Profiling dnn workloads on a volta-based dgx-1 system. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 122–133.

[44] Nvidia. 2019. Multi-GPU Programming Models. https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9139-multi-gpu-programming-models.pdf

[45] Nvidia. 2021. Optimized primitives for collective multi-GPU communication. https://github.com/nvidia/nccl

[46] Tesla NVIDIA. 2017. V100 white paper. *NVIDIA Corporation* (2017). https://images.nvidia.com/content/pdf/dgx1-v100-system-architecture-whitepaper.pdf

[47] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 10 (Oct. 2004), 1367–1372. https://doi.org/10.1109/TPAMI.2004.75

[48] Sam Partee, Matthew Ellis, Alessandro Rigazzi, Scott Bachman, Gustavo Marques, Andrew Shao, and Benjamin Robbins. 2021. Using Machine Learning at Scale in HPC Simulations with SmartSim: An Application to Ocean Climate Modeling. *arXiv preprint arXiv:2104.09355* (2021).

[49] 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. 2019. Merlin: enabling machine learning-ready HPC ensembles. *arXiv preprint arXiv:1912.02892* (2019).

[50] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[51] Kiran Ranganath, AmirAli Abdolrashidi, Shuaiwen Leon Song, and Daniel Wong. 2019. Speeding up Collective Communications Through Inter-GPU Re-routing. *IEEE Computer Architecture Letters* 18, 2 (2019), 128–131.

[52] Baidu Research. 2017. Baidu All-Reduce. https://github.com/baidu-research/baidu-allreduce

[53] Dipanjan Sengupta, Anshuman Goswami, Karsten Schwan, and Krishna Pallavi. 2014. 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*. IEEE, 513–524.

[54] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. 2018. Mesh-TensorFlow: Deep Learning for Supercomputers. In *Neural Information Processing Systems*.

[55] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[56] Shuaiwen Song, Matthew Grove, and Kirk W Cameron. 2011. An iso-energy-efficient approach to scalable system power-performance optimization. In *2011 IEEE International Conference on Cluster Computing*. IEEE, 262–271.

[57] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (San Francisco, California, USA) *(AAAI'17)*. AAAI Press, 4278–4284.

[58] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014). arXiv:1409.4842 http://arxiv.org/abs/1409.4842

[59] Nathan R Tallent and Adolfy Hoisie. 2014. Palm: Easing the burden of analytical performance modeling. In *Proceedings of the 28th ACM international conference on Supercomputing*. 221–230.

[60] Jingweijia Tan, Shuaiwen Leon Song, Kaige Yan, Xin Fu, Andres Marquez, and Darren Kerbyson. 2016. Combating the reliability challenge of GPU register file at low supply voltage. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*. IEEE, 3–15.

[61] Carlos H. C. Teixeira, Alexandre J. Fonseca, Marco Serafini, Georgos Siganos, Mohammed J. Zaki, and Ashraf Aboulnaga. 2015. Arabesque: A System for Distributed Graph Mining. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 425–440. https://doi.org/10.1145/2815400.2815410

[62] Devashree Tripathy, Amirali Abdolrashidi, Laxmi Narayan Bhuyan, Liang Zhou, and Daniel Wong. 2021. Paver: Locality graph-based thread block scheduling for gpus. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 3 (2021), 1–26.

[63] Devashree Tripathy, Amirali Abdolrashidi, Quan Fan, Daniel Wong, and Manoranjan Satpathy. 2021 (To appear). 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)).

[64] Devashree Tripathy, Hadi Zamani, Debiprasanna Sahoo, Laxmi N Bhuyan, and Manoranjan Satpathy. 2020. Slumber: static-power management for gpgpu register files. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 109–114.

[65] J. R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *J. ACM* 23, 1 (Jan. 1976), 31–42. https://doi.org/10.1145/321921.321925

[66] Julian R. Ullmann. 2011. Bit-Vector Algorithms for Binary Constraint Satisfaction and Subgraph Isomorphism. *ACM J. Exp. Algorithmics* 15, Article 1.6 (Feb. 2011), 64 pages. https://doi.org/10.1145/1671970.1921702

[67] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil Devanur, and Ion Stoica. 2019. Blink: Fast and generic collectives for distributed ml. *arXiv preprint arXiv:1910.04940* (2019).

[68] Lijing Wang, Jiangzhuo Chen, and Madhav Marathe. 2019. DEFSI: Deep learning based epidemic forecasting with synthetic information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 9607–9612.

[69] Emma White. 2019. Optimizing deep learning on P3 and P3dn with EFA. https://aws.amazon.com/blogs/compute/optimizing-deep-learning-on-p3-and-p3dn-with-efa/

[70] Jeremiah J Wilke, Joseph P Kenny, Samuel Knight, and Sebastien Rumley. 2018. Compiler-assisted source-to-source skeletonization of application models for system simulation. In *International Conference on High Performance Computing*. Springer, 123–143.

[71] Daniel Wong, Nam Sung Kim, and Murali Annavaram. 2016. Approximating warps with intra-warp operand value similarity. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 176–187. https://doi.org/10.1109/HPCA.2016.7446063

[72] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 595–610.

[73] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh. 2018. Modular Routing Design for Chiplet-Based Systems. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 726–738. https://doi.org/10.1109/ISCA.2018.00066

[74] Hadi Zamani, Yuanlai Liu, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. 2019. GreenMM: energy efficient GPU matrix multiplication through undervolting. In *Proceedings of the ACM International Conference on Supercomputing*. 308–318.

[75] Hadi Zamani, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. 2020. 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*. 205–210.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

MAPA is hosted on GitHub and can be accessed on https://github.com/socal-ucr/MAPA. The repo consists of necessary submodules - Peregrine, Caffe, ML models used in the paper, and the JobGenerator.

Follow the instructions on https://github.com/socal-ucr/MAPA/blob/master/README.md to compile and run MAPA software.

We ran our experiments on DGX-1 V100 Machine. DGX-1 V100 has 8 V100 GPUs which are connected via Double and Single NVLinks as described in the paper. We used the Caffe framework for workload evaluation. Peregrine was used for subgraph pattern matching as described in the paper. Compatible Caffe and Peregrine dependencies are provided as submodules in the MAPA repository.

We used the following Caffe-models available in https://github.com/socal-ucr/caffe-models. Our training jobs used the image-net data set available at http://www.image-net.org.

We used GCC 9.3 to compile MAPA. Caffe requires GCC-5.4.0, NCCL-2.5.7, and CUDA-11 to compile. All the experiments were run on Ubuntu-16.04. The system configuration is as follows: CPU: Intel Xeon E5-2698 processors GPUs: Nvidia Volta V100

Author-Created or Modified Artifacts:

```
Persistent ID:
↪ https://zenodo.org/badge/latestdoi/310419676
Artifact name: MAPA
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: CPU-Intel Xeon E5-2698 processors, GPU-Nvidia Volta V100

Operating systems and versions: Ubuntu 16.04 running linux kernel 4.4.0

Compilers and versions: GCC>=9.3.0, G++>=9.3.0, NCCL>=2.5.7, Unittest++, CUDA>=9

Applications and versions: Caffe

Libraries and versions: Peregrine

Input datasets and versions: ImageNet

URL to output from scripts that gathers execution environment information.

```
https://intra.ece.ucr.edu/~kranganath/dgx-v\_environ」
↪ ment.md
```