Norwegian University of
Science and Technology

TDT4501 - Computer Science, Specialization Project

Investigating New GPU Features for Performance

Knut Kirkhorn & Ingunn Sund

Advisor
Dr. Anne C. Elster

January 6, 2020

Abstract

This report compares different GPUs and multi-GPU systems to evaluate performance for new hardware features. Some of these features are Tensor Cores, NVLink and NVSwitch. Multi-GPU systems with special interconnect configurations are benchmarked and compared. The purpose of this evaluation is to which systems or GPUs could be good for which tasks.

The systems and GPUs that are benchmarked is NVIDIA DGX-2 and two versions of the IBM Power System AC922, GeForce GTX 980 and Titan RTX. The benchmarking is done with the benchmark suites SHOC, DeepBench, Tartan and Scope.

During the benchmarking process, some challenges occurred, especially with running the benchmarks inside GPU accelerated Docker containers.

The results from the benchmarking are among other things that DGX-2 is better at GPU-GPU communication than the Power AC922 systems, but the Power AC922 systems are better for CPU-GPU communication. Which system is advisable to use will therefore depend on what kind of application should run on it.

The Power AC922 systems seemed to have worse performance on the second NUMA node than the first. Choosing the right GPUs on this system can be essential for best possible performance, depending on the application.

An interesting result for the DGX-2 is that there were no significant difference in the performance for the GPU-GPU communication over NVSwitches for any GPU combination.

In conclusion, there are a lot of results that can be analyzed further, and there are many possibilities for work that can continue expanding on these results.

Table of Contents

List of abbreviations

Abbreviation	**Explanation**
AI	Artificial Intelligence
CUDA	Compute Unified Device Architecture
CPU	Central Processing Unit
FLOPS	Floating Point Operations Per Second
GCC	GNU Compiler Collection
GEMM	GEnaral Matrix Multiply
GPU	Graphics Processing Unit
GT/s	Giga Transfers per second
HPC	High Performance Computing
MPI	Message Passing Interface
NCCL	NVIDIA Collective Communication Library
PCIe	Peripheral Component Interconnect Express
QPI	Intel QuickPath Interconnect
SSL	Secure Sockets Layer
TC	Tensor Cores

Table 1: Table of abbreviations and explanations

1 Introduction

Every year computer technology advances and contributes to new concepts and more processing power. New GPUs with new exiting cores are launched, such as the NVIDIA Tensor Cores and the NVIDIA Ray Tracing Cores. The interconnects have also improved, NVSwitch is a new and effective way to connect GPUs.

Multi-GPU systems are launching with different interconnect configurations, where DGX-2 and IBM Power System AC922 are two examples.

There is a lot of new technology waiting to be fully utilized by HPC and AI applications. There are many new hardware and software optimization techniques for the computing technology. With the high interest for AI the last years, a lot of new technology are targeting AI. It can be challenging to figure out how this can be used for HPC applications.

This report focuses on benchmarking GPUs and multi-GPU systems, and it will mostly focus on the hardware optimizations. Even though CPUs are important for performance, the benchmarking will target the GPU performance and the interconnects between GPUs and between CPU and GPU. The results will show benchmarks from evaluating how the systems are connected to microbenchmarks for GPUs. There is also benchmarks for some real application kernels and Tensor Cores for both Volta and Turing architecture.

In this report, we present the systems we benchmarked: NVIDIA DGX-2, two versions of IBM Power System AC922, a computer with a GeForce GTX 980 GPU, and a computer with a Titan RTX GPU. The GTX 980 GPU is included to see the differences in performance with a GPU that is more common to have for an ordinary computer user.

The benchmarking process has a main focus on HPC benchmarks, but there are still some deep learning results for easier evaluating Tensor Core performance. The benchmarking comes from four different benchmark suites: SHOC, a relatively old benchmark suite, which makes it interesting to finding out if it is still relevant for new systems. DeepBench is a deep learning benchmark suite, but it still has some benchmarks relevant to HPC. The benchmark suites Tartan and Scope are two that has most focus on benchmarking interconnects.

1.1 Motivation

The motivation for this project comes from the need for investigating new GPU features for performance with NVIDIA DGX-2 and IBM Power System AC922, and exploring the possible improvements using NVIDIA Tensor Cores leads to for HPC applications.

We also wanted to explore new hardware technologies on GPUs and how effective they are.

1.2 Contribution

This project contributes by providing sort of a guide for which systems should be used for which task. Though work has already been done comparing the Power AC922 and DGX-2, we wanted to include comparison with GPUs as the GeForce GTX 980 and the Titan RTX.

The earlier work that has compared the systems, has had more specified tests and not as large scope as this report has. We focus on the single GPUs, the systems and the interconnects.

The report is comparing the systems that the students at NTNU have access to, and finding out what kind of applications or operations to do on which systems. This could be helpful to other users of the systems.

1.3 Outline

This report consists of these chapters with this structure:

2. Background describes the basic concepts and technologies needed for using the benchmark suites, relevant GPUs with their architectures for the tested systems. It also contains details about the systems that are benchmarked, the different benchmark suites that were used in this report and related work.

3. Setup, Approach and Benchmark Suites has a system setup for each benchmarked system, reasons for choosing the benchmark suites, how the benchmarking is conducted, challenges that occurred during this project and the different characteristics and what is wanted to be benchmarked for the systems.

4. Results and Discussion contains the most interesting results from the benchmarking and discusses and analyses these results.

5. Conclusion and Future Work summarizes the results and makes conclusions based on the results. Also, a part with future work is provided to know what could be investigated more.

The report also consists of multiple appendices containing additional information and data for interested readers:

A. Annotated Bibliography has a list of citations and why they are relevant to this project.

B. System Information contains information about how the GPUs are connected to each other and the CPU in a topology, information about the different GPUs in the system and the NVLink status.

C. Setup has information and guides for how to build and run the benchmarks on the different machines with and without Docker.

D. Benchmark Results includes more detailed results than the ones included in 4. Results and Discussion.

2 Background

2.1 Concepts and technologies

2.1.1 GPU

A graphics processing unit (GPU) is a processor specialized in graphic processing and parallel computations. GPUs were originally developed for just graphic processing, but more recently developed GPUs includes extra support for general purpose computing, which is called general-purpose GPU (GPGPU). Operations that traditionally would be done on a CPU are done on the GPU for GPGPU. This can be more efficient because of the GPU architecture which is more suited for parallel computing. GPUs are good at handling specific tasks at a high speed and CPUs can handle multiple different types of operations better and are more versatile. The bus between a CPU and GPU can often be a bottleneck to the performance. [1] [2]

2.1.2 UMA and NUMA

UMA (Uniform Memory Access) and NUMA (Non-uniform Memory Access) are two types of shared memory models for multiprocessor systems. The models describe how memory and hardware are shared between the processors. For the UMA architecture there is one memory for the processors to share. This leads to the same access speed for every processor. It can work fine when there are few processors, but does not scale very well. In a multiprocessor system with NUMA architecture, each processor has its own local memory. The memory access speed will depend on which memory the processor needs, and it is a good architecture for when it mostly needs its own local memory. A NUMA node is a CPU-memory couple and can include hardware like GPUs. [3]

2.1.3 Docker

Docker is an application that makes it possible to run applications with their dependencies inside an isolated container. It is made to be portable such that running the same Docker image on different host platforms will use the same environment settings. It is isolated from the host operating system such that it does not interfere with the files and environment variables or other Docker containers running on the same host machine. [4]

NVIDIA Docker is an extension to Docker and enables the use of GPUs to accelerate applications inside containers. [5]

2.1.4 MPI and NCCL

MPI (Message Passing Interface) is a standardized interface of protocols and functions for passing messages and communicating in a parallel environment with multiple computers. MPI provides a set of functions that are used in the implementations to communicate between the nodes. [6] There exist many different implementations, such as Open MPI [7], Spectrum MPI [8] and MPICH [9].

NCCL (NVIDIA Collective Communication Library) is a library providing MPI similar functions such as AllReduce, AllGather and Broadcast, to the available GPUs within and across multiple nodes. [10] It is however minor differences from normal MPI implementations, for example such that it can have many ranks related to the same process. It can also be used together with an MPI program. This can for example be that the MPI implementation provides the CPU to GPU communication and NCCL provides the GPU to GPU communication. [11]

2.1.5 PCI Express

PCI (Peripheral Component Interconnect) Express, or PCIe for short, is a bus standard that provides communication between connected components in a computer, such as hard drives and graphics cards. The normal connection between the GPU and CPU is done over PCIe. However, this can be a bottleneck due to its maximum transfer rate of 8 GT/s per lane for version 3 and 16 GT/s per lane for version 4. [12] [13]

2.1.6 NVLink 2.0 and NVSwitch

NVIDIA NVLink is a GPU interconnect which offers much faster data transfer and is more scalable than using the PCIe. [14] NVLink can be used for both GPU to GPU and CPU to GPU connection. For each lane in the NVLink it has a transfer rate of 25 GT/s. [15, p. 115] This can reduce the bottleneck caused by transferring over the PCIe bus.

NVSwitch is a switch for connecting NVLinks together. It has 18 ports for connecting NVLinks and each NVLink connected can achieve simultaneously 25 GB/s bandwidth speed in both ways. In total the NVSwitch can therefore achieve a total bandwidth speed of 900 GB/s. [16, p. 3]

In Figure 1 below it is shown the connections consisting of NVLinks between the GPUs and NVSwitches. This is for the NVIDIA DGX-2 system.

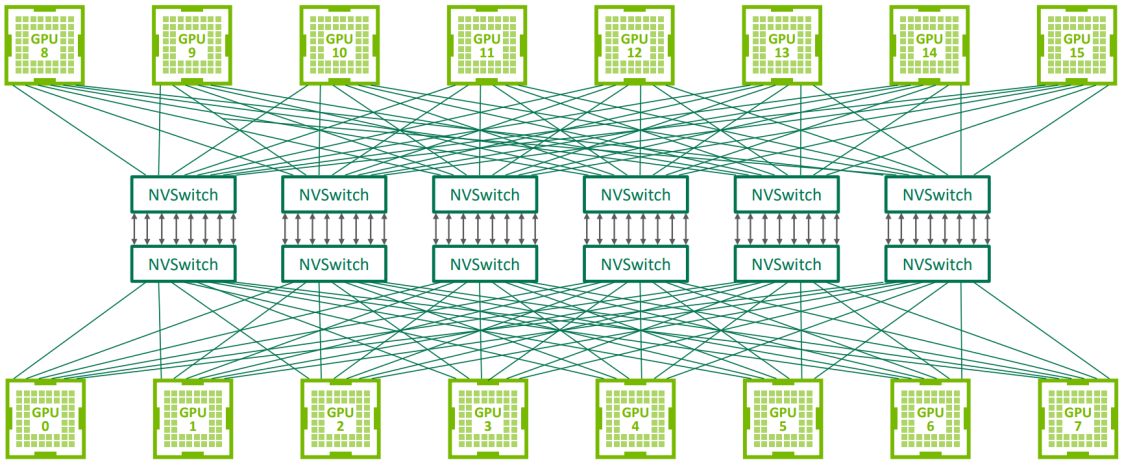

Figure 1: NVSwitch topology on DGX-2 [17, p. 8].

2.2 Project relevant GPUs

2.2.1 NVIDIA GeForce GTX 980

The NVIDIA GeForce GTX 980 is a graphics card from 2014 with the Maxwell 2.0 architecture. It has 4 GB of GDDR5 memory with a bandwidth speed of 224 GB/s. It can achieve performances of 4.9 teraFLOPS for single precision and 155.6 gigaFLOPS for double precision. The GPU is equipped with 2048 CUDA cores. [18]

The Maxwell architecture introduced improved Streaming Multiprocessor (SM) architecture design. The architecture included more power efficient processors in numerous ways, for example by increasing the number of instructions per clock cycle. [19]

2.2.2 NVIDIA Tesla V100

The NVIDIA Tesla V100 is a GPU based on the Volta architecture and there exists versions with 16 GB or 32 GB of the memory type HBM2 (High Bandwidth Memory) with a bandwidth speed of 900 GB/s. It can achieve performances of 125 teraFLOPS for deep learning (mixed precision), 15.7 teraFLOPS for single precision and 7.8 teraFLOPS for double precision. The GPU is equipped with 640 Tensor cores and 5120 CUDA cores. [20, p. 27]

Volta is the first architecture with specialized mixed-precision cores called NVIDIA Tensor Cores.

The Tensor Cores can perform one matrix multiply and accumulate operation in one clock cycle on a 4x4 matrix. Tensor Cores performs operations in mixed precision. The input data is half precision, multiplication is in half precision and accumulation is in single precision. This will lead to some precision loss, which deep neural networks can be tolerant to. HPC applications, on the other hand, cannot always handle the precision loss. [21]

2.2.3 NVIDIA Titan RTX

The NVIDIA Titan RTX is a graphics card based on the Turing architecture. The GPU has 24 GB of GDDR6 GPU memory and with a bandwidth of 672 GB/s. The card can achieve performance of 130 teraFLOPS with its 576 tensor cores made for mixed precision. The GPU also has 4608 CUDA cores. [22]

The Turing architecture provided new and improved Tensor cores. A part of the new design is the added INT8 and INT4 precision modes for inference operations. [23, p. 4]

Another new feature on this graphics card is Ray Tracing cores. These cores came with the Turing architecture.

2.3 Computing systems

2.3.1 IBM Power System AC922

The IBM Power System AC922 is a system designed for giving great performance to data analytics, HPC applications and especially AI training. IBM Power System AC922 will be referred to as Power AC922 from now on. The system has two IBM POWER9 processors, the first chip with PCIe Gen4 which has twice the bandwidth of the previous PCIe generation. [24] [25]

The Power AC922 supports up to 4 or 6 NVIDIA TeslaV100 GPUs depending on the model, where the GPUs can have 16GB or 32GB memory. [20, p. 4-8] The GPUs are split evenly between two POWER9 CPUs. If there are a total of four GPUs, two will be directly connected to the first CPU and the other two will be connected to the second CPU, as can be seen in Figure 2. The GPUs are connected to their CPU and to any siblings with NVLink 2.0. The NVLink 2.0 channels are called NVLink Bricks, and each GPUs and CPUs has six of them. The NVLink Bricks are combined to achieve the highest bandwidth attainable. This means that if the Power AC922 has a total of four GPUs, there will be NVLink Brick groups of three (Figure 2), and with six GPUs there will be groups of two to ensure connection between a CPU and its connected GPUs and the connection between the GPUs connected to the same CPU. [20, p. 12-15]

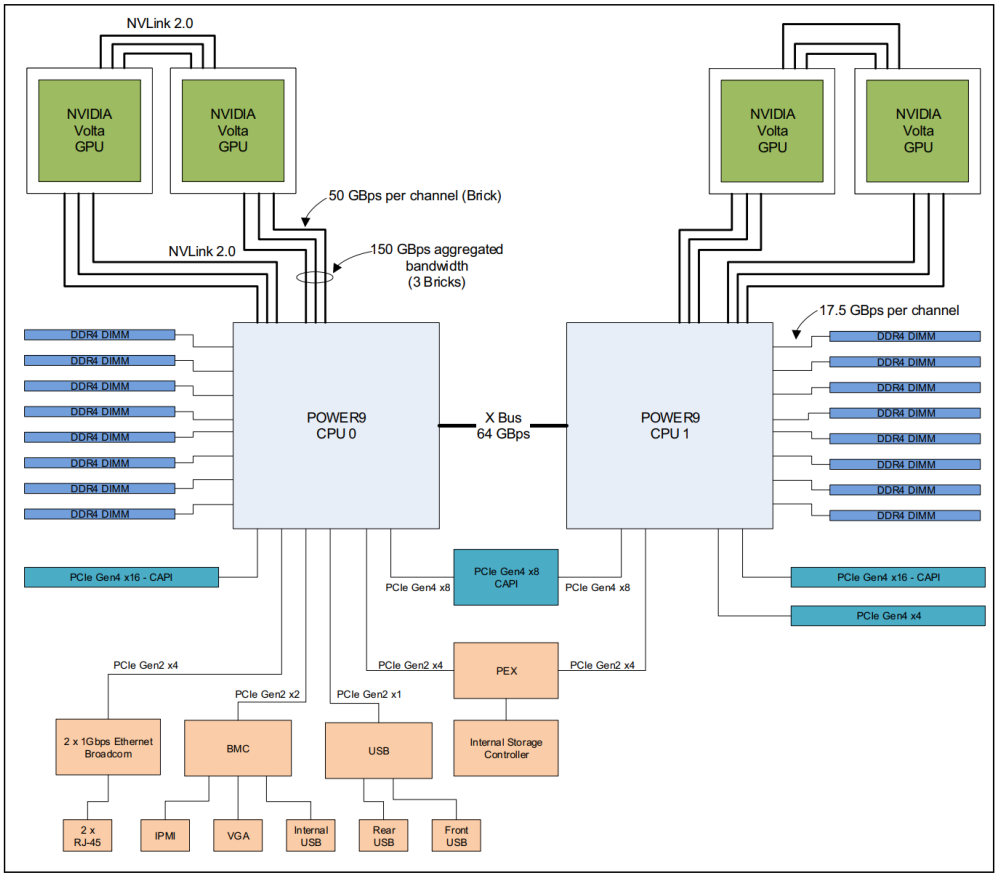

Figure 2: Logical system diagram for IBM Power System AC922 with four GPUs [20, p. 14].

2.3.2 NVIDIA DGX-2

NVIDIA DGX is a series of systems created by NVIDIA for deep learning and complex AI applications. DGX-2 is version two of this system line and is approximately twice as fast as version one (DGX-1). It consists of 16 Tesla V100 GPUs with 32 GB of memory each, which is 512 GB in total. The system has in total 81 920 CUDA cores and 10 240 Tensor cores. [26] The system consists of two baseboards, with each having 8 GPUs. To increase the communication speed between the GPUs, they are connected with 12 NVSwitches, as can be seen in Figure 1. Six NVSwitches belongs to each baseboard, which means that the connection must traverse one NVSwitch if both GPUs are on the same baseboard, and through two NVSwitches if the GPUs are on different baseboards. All GPUs in this system have a bonded set of six NVLinks between each other as shown in Listing 13 in Appendix B.

The system has two Intel Xeon Platinum 8168 CPUs with 24 cores and a base clock frequency of 2.7 GHz. Between the two CPUs there is a QPI connection and each CPU has a PCIe connection with two PCIe switches to each GPU on their baseboard as can be seen in Figure 3. It can achieve the maximum performance for deep learning applications of 2 petaFLOPS which means that this system may be well suited for large workloads.

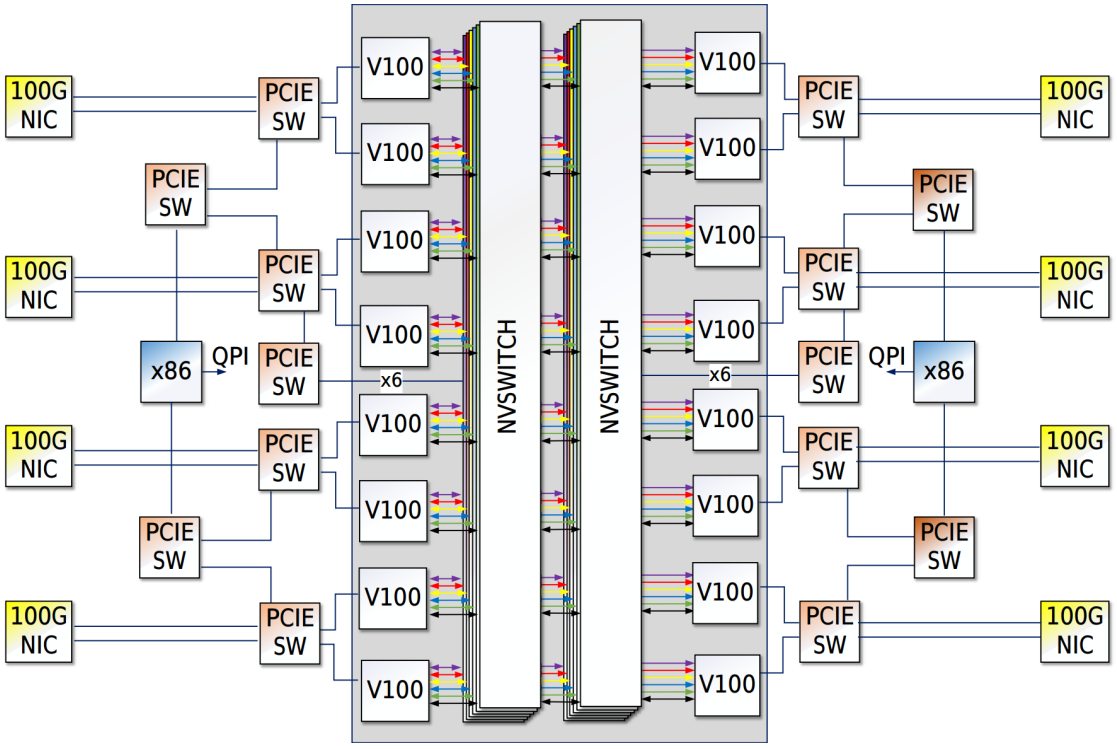

Figure 3: Interconnect diagram for DGX-2 [27, p. 19].

2.4 Benchmark suites

2.4.1 SHOC

The Scalable HeterOgeneous Computing (SHOC) benchmark suite includes a series of benchmark programs for measuring performance and stability on systems. SHOC was made to effectively test systems with multiple GPUs and CPUs. It is almost a decade since this suite was published and since this was before even Tensor Cores was invented, it does not use tensor cores for accelerating the computations of the benchmark applications. The suite provides a set of benchmark applications for measuring the performance of different high performance computing for both clusters and individual computer systems. [28]

As described in "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite" by Anthony Danalis et al.[29], the benchmarking suite is divided into three different levels of benchmark applications. These are named level 0, 1 and 2 respectively. The first level is for measuring hardware interconnect speeds and device characteristics, such as transfer speed on the bus and maximum FLOPS. The second is for measuring performance for basic algorithms such as GEMM and FFT. The third level is for measuring the performance of real world applications.

This benchmarking suite is also categorized into three different types of running the applications. These are EP (embarrassingly parallel), Serial and TP (true parallel). Table 2 has a list of benchmark applications from SHOC that are relevant for this report. All of these benchmarks are related to HPC.

Benchmark	Description
BusSpeedDownload	Measures the bandwidth transfer speed over the PCIe-bus from the host to the device.
BusSpeedReadback	Measures the bandwidth transfer speed of reading back data over the PCIe-bus from the device.
MaxFlops (`maxspflops` and `maxdpflops`)	Measures the maximum floating-point performance for different floating point operations. This does not include PCIe transfer.
Stencil2D (`stencil` and `stencil_dp`)	Measures the performance of single and double precision for 2D 9-point stencil computations.
QTC (`qtc` and `qtc_kernel`)	Measures the performance of the Quality Threshold Clustering algorithm. `qtc` includes the PCIe transfer time and `qtc_kernel` does not.

Table 2: Table of SHOC benchmarks analysed in this report

2.4.2 DeepBench

DeepBench is a benchmarking suite containing deep learning operations made by Baidu Research, which is a company that focuses on AI research. [30] It consists of microbenchmarks for a set of architectures e.g. Intel, ARM and NVIDIA. The benchmarks are for measuring the performance of operations that are used in deep learning frameworks such as Tensorflow [31] and Torch [32]. The benchmarks can only be run on one GPU at the time.

In this report, the focus will be on the NVIDIA benchmarks. The different benchmark applications for the NVIDIA benchmarks are GEMM, convolutional neural network, recurrent neural network, sparse matrix operations multiplication and AllReduce using MPI and NCCL.

Table 3 contains the benchmarks from DeepBench relevant to this report. These are relevant to HPC because they benchmark operations that are common for computation as well as deep learning.

Benchmark	Description
GEMM train (float)	Measures the performance of training with float operations used for GEneral Matrix Multiplication.
GEMM infer (float, int8)	Measures the performance of inference with float or int8 precision operations used for GEneral Matrix Multiplication.
CONV infer (int8)	Measures the performance of inference with integer 8 precision operations used for Convolutional Neural Network. CNNs are mostly used for image and video classification.
MPI NCCL AllReduce	Measure the performance of the MPI communication operation AllReduce. AllReduce is about combining the values of all ranks, performing an operation such as `sum` or `min` and reducing it to all the ranks.

Table 3: DeepBench benchmarks analysed in this report

2.4.3 Tartan

Tartan is a benchmark suite with focus on evaluating interconnects created by Ang Li et al. [33] The creators of Tartan wrote the "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect" [34] paper.

Inside the Tartan benchmark suite it contains microbenchmarks for measuring latency and bandwidth for P2P and Collective Communication (CL), for both intra-node (scale-up) and inter-node (scale-out) systems. [33]

Tartan's benchmark applications are modified version of the "CUDA SDK Code Samples" [35].

When the tests use PCIe mode it means that the communication will go via the host. When the mode is NVLink, the function `cudaDeviceEnablePeerAccess` is enabled, and the communication will go directly from one GPU to another GPU in addition to the possibility of accessing the other device's memory. [36] [37]

Table 4 contains two benchmark applications from Tartan that are relevant for this report and related to HPC.

Benchmark	**Description**
`scale_up_p2p_test` and `scale_up_p2p_packet`	These benchmarks measure the unidirectional and bidirectional bandwidth and latency for intra-node systems. These benchmarks give results for both NVLink mode and PCIe mode.

Table 4: Tartan benchmarks analysed in this report

2.4.4 Scope

Scope is a benchmark framework for both POWER and x86_64 processor architectures. It is created by people at the Center of Cognitive Computing System Research (C3SR) in collaboration with the IMPACT group at the University of Illinois[38]. It provides a framework that can be used for making different benchmark applications that can be ran with this framework. Scope includes multiple benchmark suites that uses the framework, two examples are Comm|Scope [39] and NCCL|Scope [40]. Comm|Scope measures GPU data transfer performance from latency and memory transfer speeds. NCCL|Scope, that measures GPU collective communication performance using NVIDIA's NCCL library.

Table 5 and Table 6 shows the benchmarks relevant to this project from both the Comm and NCCL benchmark suites. These are both relevant to HPC because they both measure the performance of communication, which is important when for example transferring large amount and/or fast data between nodes in systems.

Benchmark	**Description**
`Comm_Memcpy_GPUToWC`	GPU to write-combining host
`Comm_Memcpy_WCToGPU`	Write-combining host to GPU
`Comm_Memcpy_GPUToHost`	GPU to pageable host
`Comm_Memcpy_HostToGPU`	Pageable host to GPU

Table 5: Comm|Scope benchmarks analysed in this report

Benchmark	Description
`NCCL/ops/broadcast`	NCCL Broadcast. Broadcast is an MPI operation that copies a buffer from the root node to all the ranks.

Table 6: NCCL|Scope benchmark analysed in this report

2.5 Related work

Ren et al. wrote a paper called "Performance Analysis of Deep Learning Workloads on Leading-edge Systems" [41] where they also benchmarked the DGX-2 and the IBM Power System AC922. "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect" by Li et al. [34] is the paper that presents Tartan. In this paper the authors describe evaluation DGX-2 and Summit, which contains IBM Power Systems AC922. W. Feng has a presentation about "OpenCL and the 13 Dwarfs" [42] where he writes about the 13 Dwarfs and a benchmarks suite called OpenDwarfs. C. Pearson et al. wrote a paper called "Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects" [43]. This paper writes about the Scope|COMM benchmark suite.

Markidis et al. wrote a paper called "NVIDIA Tensor Core Programmability, Performance & Precision" [21] and it addresses some challenges with using the NVIDIA Tensor Cores for HPC when it comes to precision and performance.

NVIDIA Tensor Cores are hardware units used in GPUs from NVIDIA with Volta architecture or newer architecture. They are specially designed for AI applications and provide better performance for deep learning than when using CUDA cores. Since the Tensor Cores are designed for AI, it is very interesting to see how they perform for HPC applications.

The paper also goes in depth on how to program the Tensor Cores and how to decrease precision loss. The authors used a technique they called precision refinement and noted that optimized versions of this techniques are possible. The version they used were only for an estimation of computational cost. It would be interesting to see how it would have performed with a fully optimized version of refinement. How would the computational cost for ensuring precision be then?

The results show that by using the Tensor Cores there can be achieved a great performance boost, at the cost of less precision. By using the precision refinement technique, there is still a greater improvement in performance compared to not using Tensor Cores, but with only a small amount of error.

Another perspective that could be interesting is exploring the power consumption with and without Tensor Cores, and with and precision refinement for Tensor Cores.

The paper has a thorough evaluation of strength and weaknesses for Tensor Cores and how to best use them for HPC applications. The results show that Tensor Cores will most likely be very helpful in HPC applications in the future and that AI is not the only area who benefits from this new hardware feature.

Another thing that could be interesting when using Tensor Cores is comparing the performance increase for a typical HPC application with the performance increase for an AI application. This will show how much more beneficial they possibly would be to AI than to HPC.

Another paper that discusses the usage of NVIDIA Tensor Cores is "Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers" by Haidar et al. [44] This paper discusses using the NVIDIA Tensor Cores for a well known HPC problem Ax = b. A is a large dense matrix. Tensor Cores provides mixed precision mode, and a double precision result is needed for the problem.

To accelerate solvers for the problem, mixed precision iterative refinement technique is used. The authors made changes to an existing technique to adjust it for the usage of Tensor Cores. For the mixed precision iterative refinement technique, low precision is used for LU factorization, then the factorization is used in a refinement loop, and higher precision can be used for the residual.

This is highly related to the topic in the sense that NVIDIA Tensor Cores in a GPU is used to perform iterative refinement, which is a technique often used in HPC applications.

The results show that by using the framework of algorithms the authors created, there is a great performance increase when using Tensor Cores. It would have been interesting to see a more in-depth error analysis for the technique included in the paper.

The authors used real world matrices in addition to the constructed matrices to illustrate how the results could look like if this was used in practice. This is a great way to show how this technique, used on Tensor cores, can be applied to real cases.

The authors mentioned that they would like to measure the energy efficiency of using the technique on Tensor Cores. They assume that a four times speedup would at least lead to four times energy improvement. This would be interesting to test.

From these two papers on NVIDIA Tensor Cores, it is safe to say that these cores can be used for multiple types of HPC applications, even when precision is important. Another new hardware feature that is interesting is the Ray Tracing (RT) Cores that is included in the new NVIDIA Turing architecture and how these cores can be used in HPC applications.

Salmon et al. found a way to use RT Cores for a HPC purpose in a paper called "Exploiting Hardware-Accelerated Ray Tracing for Monte Carlo Particle Transport with OpenMC" [45]. RT Cores are cores to improve ray tracing algorithms. These algorithms are used for improving graphical rendering, which is great for video games. The paper describes using OpenMC to run on the RT Cores. OpenMC is a particle transport simulation code for running on the CPU.

The authors used techniques to run the particle transport simulation on the RT Cores instead of just the CPU. The simulations are represented as a problem that could be run on the RT Cores.

This paper showed significant performance increase on multiple OpenMC models when using the RT Cores compared to just using CPU.

This research could lead to other HPC applications being tested and used on RT Cores in the future. Utilizing the TR Cores for particle transport simulation is a great way to show that a hardware feature made for graphics can be used for a scientific HPC problem.

So far it is shown how two types of new cores, Tensor Cores and RT Cores, in NVIDIA GPUs can be used for different HPC applications, even though they were not originally designed for this purpose. Another type of hardware that can be used to accelerate HPC performance are new GPU interconnects.

Li et al. writes about different GPU interconnects in the paper "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect" [46]. NVLink and NVSwitch interconnects are relatively new GPU interconnects from NVIDIA, they can be seen as new GPU features for applications for HPC and other areas.

In the last years there have been an increase in the number of GPUs in machines for HPC and AI. This escalation has led to the need for better and better interconnects between the GPUs and between CPU and GPU. This is the authors' reason for wanting to evaluate modern interconnects.

The evaluation consisted of Collective and Peer-To-Peer (P2P) communication patterns. Collective communications are characterized by having multiple senders and receivers. Examples are broadcast, where one GPU sends something to all other GPUs, and allreduce, where reduction is performed, and all GPUs gets the result. The Collective communication were tested using different number of GPUs. P2P communication measures the performance of the interconnects on one GPU to another GPU. There is one sender and one receiver, and the test results shows the performance of

each GPU-GPU combination. The evaluation of Collective and P2P communications in the paper is comprehensive and detailed.

The results of the evaluation show that the specific GPU combination when using a system with multiple GPUs is important and can be essential when trying to achieve best possible performance and GPU communication. These results are therefore useful information for HPC applications. The results also show that new interconnect solutions improve communication speed between GPUs. Better interconnects between GPUs leads to better performance for HPC applications.

This paper only focuses on the performance of interconnects between GPUs, not CPU-GPU communication. Great performance of CPU-GPU interconnects is also important to ensure that HPC and other applications run as fast as possible. Summit, one of the supercomputers evaluated in this paper, uses NVLink 2.0 as interconnect between CPU and GPU. This interconnect could be interesting to compare to another CPU-GPU interconnect, like PCIe.

These new hardware features from NVIDIA have been useful for the performance of HPC applications. Next, two papers which focuses on GPU memory will be discussed.

Sullivan et al. have written a paper called "Buddy Compression: Enabling Larger Memory for Deep Learning and HPC Workloads on GPUs" [47] about memory compression. An issue with GPU memory, is that it is quite small compared to CPU memory. On the other hand, GPU has very high memory bandwidth. When applications have high memory usage, the memory capacity issue is usually solved with more GPUs or extra algorithmic complexity. To increase effective GPU memory and bandwidth, the authors of this paper presented the Buddy Compression scheme.

Buddy Compression involves compressing data and putting the compressed data on the GPU device memory. If this data does not fit in the memory, the remaining part will be placed on a larger and slower buddy-memory connected to the GPU with a high bandwidth interconnect. When changes in compressibility happens, there is no requirement for moving data in the GPU memory.

The paper shows that by utilizing this technique, a 1.5x-1.9x memory compression ratio can be accomplished for HPC workloads. Using the technique compared to GPUs that has larger memory, Buddy Compression has 1-2 % worse performance. This is not a lot of percent worse, and it is probably worth adding a buddy-memory instead of buying a GPU with the highest memory available. Specially if the buyer is a GPU user that needs a lot of memory and want to save money. To know exactly how much can be saved, a cost analysis needs to be done, and this was not included in the paper.

Buddy Compression is a technique that could make HPC applications perform better without increasing GPU memory, but by increasing the buddy-memory. One disadvantage with this scheme is that another memory and interconnect is needed. But since the other memory can be a slower memory, it would not necessary be a big sacrifice to acquire a such buddy-memory.

This technique adds hardware to improve memory capacity for GPU. The next paper will also include discussions about memory for GPUs and is a software feature to improve GPU memory performance for HPC applications.

The last paper that will be introduced in this section is "Performance Evaluation of Advanced Features in CUDA Unified Memory" by Chien et al. [48] The paper discusses the advanced features "prefetch" and "memory advises" from CUDA Unified Memory (UM). CUDA UM is a memory address space that can be accessed by every CPU and GPU in a system. This paper evaluates the performance effects when using these features with applications.

The "memory advises" feature makes it possible for the programmer to advise CUDA UM about data access patterns. The "prefetch" feature makes it possible to prefetch pageable memory with a function, which can reduce page faults. The authors of this paper made a benchmark suite to evaluate performance when using the features. There were not many benchmarking suits for this purpose since these features are relatively new. They compared using CUDA UM alone,

CUDA UM with "memory advises", CUDA UM with "prefetch", CUDA UM with both "memory advises" and "prefetch" and not using CUDA UM. This led to this being a comprehensive analysis of performance when using and not using the advanced features.

The authors concluded with that "memory advises" and "prefetch" were effective and easy to use. The features gave a performance boost for some systems in some conditions. These CUDA UM features can be seen as features to use on the GPU, and since they often improve effectiveness related to memory, they can definitely be used to help HPC applications achieve better performance.

It is challenging for HPC applications that GPUs has significantly less memory than CPUs. This is one of the reasons for the creation of CUDA UM (with the advanced features) and Buddy Compression. These techniques handle the issue of small memory in different ways and it would be interesting to see in-depth comparisons of them with other techniques that improves performance through improving memory handling.

In conclusion, there are many different new ways HPC applications can be boosted in performance. Some of them is using hardware designed for other applications, as for Tensor Cores and RT Cores. When the interconnects improve, it leads to better performance for not only HPC applications, but everything that is run on the GPU. Another way to improve HPC applications that is discussed is improving GPU memory, either by expanding it to fit more data or by using advanced features from CUDA.

3 Setup, Approach and Benchmark Suites

This section starts with describing the systems that were benchmarked and their characteristics. Next, the reasoning for choosing the benchmark suites is described. After this, the benchmarking process is described, and a section of the issues that occurred during the project timeline. Lastly, the different areas that we wanted to benchmark are described.

3.1 System setup

The systems we benchmarked were a computer with a NVIDIA GeForce GTX 980 GPU, a computer with a NVIDIA Titan RTX GPU, two of the IBM Power System AC922 and the NVIDIA DGX-2 system.

NVIDIA GeForce GTX 980 based system

This system has a NVIDIA GeForce GTX 980 GPU with 4 GB memory and an Intel Core i7-6700K CPU. There is a PCIe interconnect between the GPU and the CPU. Additional hardware specifications are shown in Table 7. Appendix B has additional information about the GPU in some listings. Listing 1 shows the topology, Listing 2 shows that the system can not use NVLinks, and Listing 3 shows extra GPU information.

	GeForce GTX 980 system
CPU	Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz with 4 cores. Maximum clock frequency: 4.2 GHz. 2 threads per core, total 8 threads.
RAM	16 GB main memory 4 GB GPU memory
GPU	NVIDIA GeForce GTX 980, 4 GB.
CPU-GPU Interconnect	PCIe
OS	Ubuntu 18.04.3

Table 7: Hardware specification for NVIDIA GeForce GTX 980 based computer

NVIDIA Titan RTX based system

This computer has a NVIDIA Titan RTX graphics card with 24 GB GPU memory. PCIe connects the GPU to the CPU, which is an Intel Core i9-9900K processor. The Titan RTX GPU has 576 Tensor Cores, and more specifications can be seen in Table 8. More information about the GPU can be found in listings in Appendix B. Listing 4 shows the topology, Listing 5 shows that the graphics card can have two NVLinks and Listing 6 displays extra GPU information.

	Titan RTX system
CPU	Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz with 8 cores. Maximum clock frequency: 5 GHz. 2 threads per core, total 16 threads.
RAM	16 GB main memory 24 GB GPU memory
GPU	NVIDIA Titan RTX, 24 GB. 576 Tensor Cores.
CPU-GPU Interconnect	PCIe
OS	Ubuntu 18.04.3

Table 8: Hardware specification for NVIDIA Titan RTX based computer

IBM Power System AC922

The two Power AC922 systems are of the model 8335-GTH, which among other things means that they have air cooling instead of water cooling [20]. The systems have two POWER9 processors each. One of the systems has four NVIDIA Tesla V100-SXM2 GPUs with 32 GB memory each, and the other one has two NVIDIA Tesla V100-SXM2 GPUs with 16 GB memory each, named Yme and Mini Summit respectively. The hardware specifications are shown in Table 3.1, the difference between them is the number of GPUs and their GPU memory. See Listing 9 and Listing 12 in Appendix B for GPU information about the systems. The listings show that the SXM2 model has a power limit of 300W. Another difference between the GPUs on the two systems is that the Tesla V100-SXM2 with 16 GB memory (on Mini Summit) has a minimum power limit of 100W whereas the Tesla V100-SXM2 with 32 GB memory (on Yme) has a minimum limit of 150W. This can be seen when running the command `nvidia-smi -q -i 0`.

	Yme (Power AC922)	**Mini Summit (Power AC922)**
CPU	2x POWER9 CPUs with 16 cores each. Maximal clock frequency: 3.8 GHz. 4 SMT threads per core, total 128 threads.	2x POWER9 CPUs with 16 cores each. Maximal clock frequency: 3.8 GHz. 4 SMT threads per core, total 128 threads.
RAM	512 GB main memory 128 GB GPU memory	512 GB main memory 32 GB GPU memory
GPU	4x NVIDIA Tesla V100-SXM2, 32 GB. 640 Tensor Cores.	2x NVIDIA Tesla V100-SXM2, 16 GB. 640 Tensor Cores.
GPU-GPU Interconnect	NVLink 2.0	NVLink 2.0
CPU-GPU Interconnect	NVLink 2.0	NVLink 2.0
OS	Red Hat Enterprise Linux 7.6	Red Hat Enterprise Linux 7.6

Table 9: IMB Power System AC922 hardware specification

Yme, the system with four GPUs, has two GPUs connected to each processor with three NVLink 2.0 bricks between. There are also three NVLink 2.0 bricks between the two GPUs connected to each CPU. Figure 4 shows the architecture of Yme. For the outprinted topology of Yme see Listing 7 in Appendix B. All four GPUs in this system has all six NVLinks 2.0 bricks active. This can be seen in Listing 8.

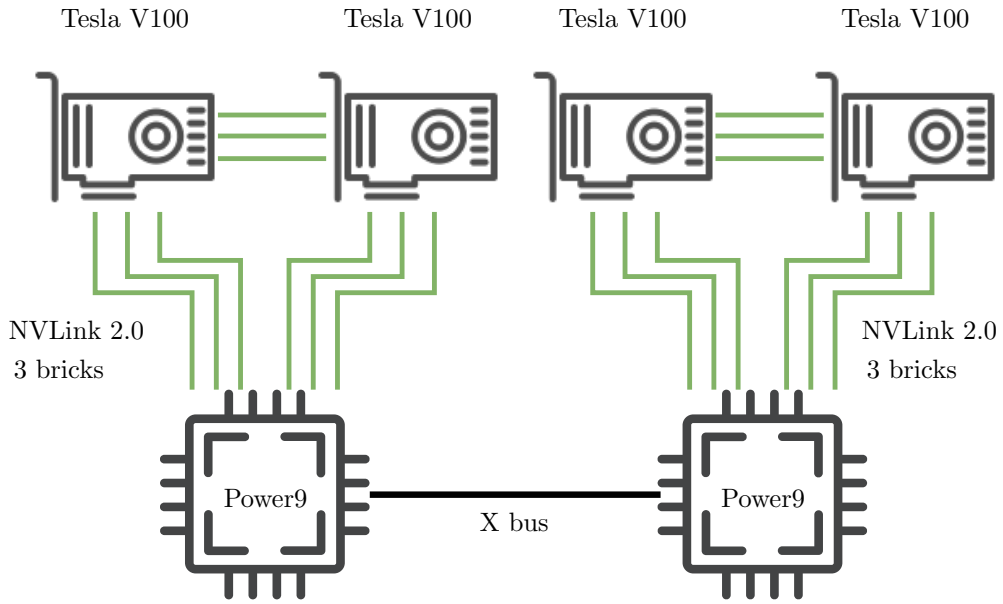

Figure 4: Illustration of how the GPUs and CPUs are connected on Yme (Power AC922 with four GPUs). The illustration is made in *draw.io*.

Mini Summit, the system with two GPUs, has one GPU connected to each processor. This system also has three bricks of NVLink 2.0 connecting the GPUs to their CPUs. Mini Summit's architecture can be seen in Figure 5. See Listing 10 in Appendix B for the outprinted topology. Both the POWER9 CPUs and the Tesla V100 GPUs have possibility for six NVLink 2.0 bricks to another device, but this system does not use the other three bricks. See Listing 11 in the same appendix for the NVLink active status for the GPUs.

Both systems is divided into two NUMA nodes, where in Yme each node has one CPU and two GPUs and in Mini Summit each node has one CPU and one GPU.

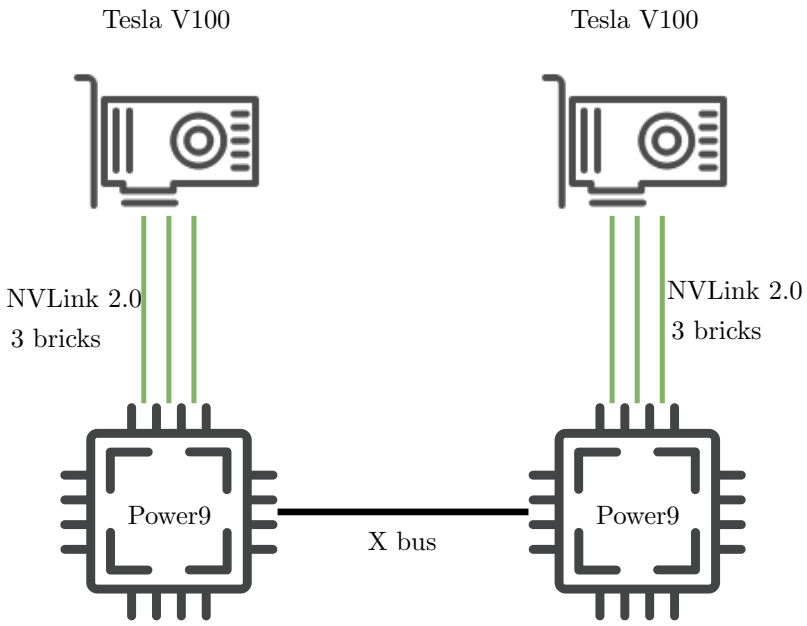

Figure 5: Illustration of how the GPUs are connected to the CPUs on Mini Summit (Power AC922 with two GPUs). The illustration is made in *draw.io*.

NVIDIA DGX-2

The NVIDIA DGX-2 has two Intel Xeon Platinum 8186 processors and 16 NVIDIA Tesla V100-SXM3 GPUs, each with 32 GB memory. These Tesla V100 graphic cards differs from the SXM2 version in that they have different power consumption limits and there are some architectural differences [49]. In Listing 15 from Appendix B there is shown that Tesla V100-SXM3 has a power limit of 350W, 50W more than the SXM2 models. This extra power is dedicated to increasing the clock rate [50] which is about 60-80 MHz higher than for the SXM2 models, depending on the usage. The clock rates can be seen by running the `nvidia-smi -q -i 0` command. This command also shows that the GPUs in the DGX-2 have a minimum power limit of 100W.

This system has NVSwithes between the GPUs and PCIe connection between CPU and GPU. The connection between the GPUs traverses through a bounded set of six NVLinks, the NVLink status can be seen in 14. The system is divided into two NUMA nodes with 8 GPUs per node. This topology can be seen in Listing 13 in Appendix B. The graphic cards has 640 Tensor Cores and more specifications can be seen in Table 10.

	Heid (DGX-2)
CPU	2x Intel(R) Xeon(R) Platinum 8168 CPUs @ 2.70GHz with 24 cores each. Maximum clock frequency: 3.7 GHz. 2 threads per core, total 96 threads.
RAM	1510 GB main memory 512 GB GPU memory
GPU	16x NVIDIA Tesla V100-SXM3, 32GB. 640 Tensor Cores.
GPU-GPU Interconnect	NVSwitch
CPU-GPU Interconnect	PCIe
OS	Ubuntu 18.04.3

Table 10: Nvidia DGX-2 hardware specification

3.2 The benchmark suites

The benchmark suites were chosen for different reasons. There were some challenges finding new and popular benchmark suites that targets purely HPC applications, which lead us to including a benchmark suite with AI focus called DeepBench.

All of the chosen benchmark suites were made by people or organizations that are known in the field or have published papers with multiple citations. Another common trait for the benchmark suites is that they all are tailored for heterogeneous systems.

SHOC

SHOC were chosen because it is a known benchmark suite by well acknowledged people in the HPC field and the creators of SHOC has published a paper describing the benchmark suite that has achieved hundreds of citations [29]. Another reason that we chose this suite is that it claims to be very scalable and could therefore be used to measure performance for systems such as the DGX-2 which has 16 GPUs.

However, the fact that SHOC has not had many updates in the last years might result in the benchmark suite not being optimized for the latest hardware and their features. This lead us to being interested to test if it still worked and showed interesting results on newer architectures, and if it still is relevant.

DeepBench

The DeepBench benchmark suite were chosen to evaluate the different GPUs that supports Tensor Cores and see how the performance differs with and without this GPU feature. The fact that it is possible to choose to run the benchmarks with or without Tensor Cores is great for this purpose. It was also chosen to compare devices that does not have Tensor Cores at all (GTX980) with devices that has the cores. We also wanted to see how the new Turing Tensor Cores (from the Titan RTX) compared to the older Volta Tensor Cores.

Even though DeepBench is a benchmark suite with AI focus, there are still benchmarks that provides results that can be useful to evaluate HPC performance. The GEMM operation is used in multiple benchmarks from the suite, and is very relevant to HPC applications. As mentioned earlier, there is shown that Tensor Cores can be useful for applications in HPC as well as for AI.

This benchmark suite was also chosen because it is relatively widely used, judging by the stars on their GitHub repository, and it has recently been updated. It is also interesting to see how the GPUs preforms with AI operations to see some variations in the results.

Tartan

The paper about evaluating modern GPU interconnects [34] intrigued us and made us look into the benchmark suite they created, Tartan. The benchmark suite is specialized in measuring the performance of interconnects.

Even though they already benchmarked the DGX-2 and Summit supercomputer, which includes multiple IBM Power System AC922, we wanted to test the systems we had access to. They did not do any intra-node benchmarks on Summit, which we wanted to do on the Power AC922 systems.

Judging by the Tartan GitHub repository [33], the benchmark suite is less known than SHOC and DeepBench, but it is created by Ang Li, a scientist from Pacific Northwest National Laboratory (PPNL), which lead us to believe that this was a dependable benchmark suite.

Scope

We found the Scope benchmark suite when researching Tartan. We found a paper called "Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects" by Pearson et al. [43] that were referencing a paper about Tartan. They wrote about how Scope|COMM is different from Tartan especially in how they focuses more on GPU-CPU communication. We found this interesting and chose to use this suite for this purpose.

Since we were going to use the Scope|COMM benchmark suite, we decided to also use the Scope|NCCL benchmark suite for convenience. Scope|NCCL focuses on collective communication.

Scope is a relatively new and because of this not as known as the other benchmarks suites. The papers about Scope seems good, which gives confidence in that the suite is well made.

In the GitHub description the benchmark suite specifies that it is a framework for both POWER and x86_64 processor architectures. This fits our project as they are including all the systems we tested.

3.3 Benchmarking process

The initial plan was to create Dockerfiles for every benchmark suite and alter the Dockerfile to match each separately system that were benchmarked. Then build the Docker images and run them without any clashes with other environment settings. These containers use the NVIDIA Container Toolkit extension with regular Docker to run GPU accelerated Docker containers. To use Docker was not as problem free for all the systems, as described in the next section about challenges during the benchmarking process, which lead to two system not using Docker.

These issues lead to possible differences in the benchmark results with the potential overhead for the systems that ran the benchmarks inside Docker containers compared to those that did not use containers. NVIDIA claims this overhead to possibly be negligible depending on the workload [51].

The benchmark suites that used Docker containers, would use the NVIDIA Container Toolkit to build and run GPU accelerated Docker containers. These containers included their respectively source codes for the benchmark suites and during the building process of the containers they downloaded and installed the needed dependencies. After building a container it can be run many times with the same isolated environment settings without reinstalling software or adjusting anything. For more specific information about building and running Docker containers see Listing 16 and Listing 17 in Appendix C. The software and other prerequisites for building the benchmark suites, with and without Docker, are listed in their respective sub section in Appendix C.

To make sure our results from the benchmarks were correct and could be trusted, we ran the benchmark suites multiple times for different settings and ensured that specific criteria between each run was fulfilled.

We made sure that there were no other significant processes running on the CPUs and no processes running on the GPUs at all. As other processes using the interconnects, CPUs, memory and GPUs can lead to wrong measurements. We used the `nvidia-smi` command line tool to verify that there were no other processes running on the GPUs, and that there were no other major processes on the CPU by using the built in Linux command `top`.

To make the benchmarks consistent for all of the systems, we made sure that the temperature of the GPUs was no more than 40 C°. We are unsure how much this could affect performance, but D. C. Price et al. wrote a paper called "Optimizing performance-per-watt on GPUs in High Performance Computing" [52] that shows differences in performance when the temperature varies.

The benchmarks from SHOC and DeepBench, the benchmark suites that could potentially use Tensor Cores, were profiled to check if Tensor Cores were utilized. For verifying that Tensor cores were used, we used the `nvprof` profiling command and looked for kernels that contained the number `884` [53, p. 12]. SHOC did not use Tensor Cores and DeepBench used them when enabled in the building.

The main software dependencies other than the NVIDIA Docker that was needed to run the benchmark suites were CUDA, MPI, NCCL, CMake/Make, cuDNN, cuBLAS. Before performing the building process of a benchmark suite, the needed dependencies would have to be installed. See the complete list of software dependencies for each benchmark suite in Appendix C.

3.4 Challenges that occurred during the benchmarking

During the benchmarking process we stumbled upon numerous challenges. These occurred in the setup, building and running phase of the benchmark suites and lead to the benchmarking process being more difficult and time consuming than we initially thought.

The benchmark suites

The SHOC benchmarking suite required some work to be able to build and run. One of the issues was that during the project timespan, the CUDA version was updated to 10.2 and was then updated for our systems. In that version the compiler linking options changed order, which lead a script in this benchmark suite to not being able to compile the CUDA-code. A fix for this compilation issue, making the benchmark suite compatible with CUDA version 10.2, is provided in our Git fork with link in Appendix C.

Another issue was that the source code had fairly outdated configuration scripts for detecting the system the benchmarks was running on. This was an issue for the POWER9 architecture and the RHEL 7.6 operating system. This was not that difficult to fix, and the script just needed to be updated from its origin source, which was luckily updated very recently.

Inside the Docker container and when running the benchmark suite, the `mpirun` command was needed to run as root, which lead to another change in the Git fork providing the `--allow-run-as-root` flag.

DeepBench did not lead to any issues when running inside a Docker container.

The running of the Tartan benchmark suite did not result in any major issues either, but there were required some minor changes before building to make it work for the tested systems. This is more explained under the subsection for Tartan in Appendix C.

The benchmark suite Scope, did not require much work for the compiling part of the benchmarks used. This was mostly due to the source code included already provided Docker-files for building it. But for finding the correct commands to use and parameters for them, did result into trial and error on command options.

In the process of choosing and testing benchmark suites, we had the intention of trying out other suites than were used in this report. This led to much time used troubleshooting them. Two of the suites we tried to make work is the HP DLBS and OpenDwarfs benchmark suites. HP DLBS was very problematic to get to work inside Docker container due to it needed to start the benchmark applications in Docker containers. OpenDwarfs was unfortunately unable to compile for our systems, both inside and outside of Docker. We were unable to fix both these suites during the project timespan.

Not using NVIDIA Docker on the Power AC922 systems

The first issue that occurred was with the NVIDIA Docker installation on both Power AC922 systems. The installation did not work with the runtime CUDA and would therefore not recognize the GPU devices connected, making it almost useless for the benchmarking suites, which relies on running the code on the GPUs. This was an issue that occurred early in the project, and there were a lot of back and forth communication with the system administrators of the systems. The conclusion became that in order to fix the NVIDIA Docker, it had to update to a newer version, since the one installed was fairly outdated (version 1.13.1 vs. the current being 19.03 at that time). The newer version of Docker required a newer version of the operating system as well, which in the timeline of this project was not possible to do. This indicated that this would became a harder

problem to solve. After the multiple unsuccessful attempts of updating the Docker version, the solution to this problem was to abandon the running of the benchmarks inside Docker containers and run each of the benchmark suites "bare metal" on both the Power AC922 systems. This made them much more likely to cause trouble during the building and running of the benchmark applications due to the non-isolated environments.

Several of the benchmark suites required an implementation of MPI for building and running. This was very time consuming to fix and were fixed very late in the project timeline. Inside Docker containers this was fairly simple, but without Docker this was more problematic. Since there was installed multiple MPI implementations on the systems already, we tried to make it work with them. At first the system environment variables were misconfigured and did not match the installed MPI installation. This was due to the MPI implementation in the environment variables, named Spectrum MPI needed extra linker options to work. After this was fixed the implementation worked for compilation, but showed error when trying to run it. The error was related to an evaluation period. However, there was installed another MPI implementation, Open MPI, on the Power AC922 systems. After changing the environment variables to match this MPI, both the compilation and the running of the benchmarks worked.

Another time consuming issue in the project was with the installed CUDA on the Power AC922 machines. This was something a system administrator needed to fix.

Other than the challenges with MPI, CUDA and the general challenges not related to running without Docker for SHOC and Tartan, these benchmark suites did not cause any more problems.

DeepBench required additional dependencies such as NCCL, cuDNN and cuBLAS to compile and run. This was also something that the system administrator needed to install.

Scope was more complicated to fix. It needed NCCL as well as a newer version of CMake than was already installed on the systems. After some troubleshooting, we chose to install a new version locally from source for our own user. After installing the newer version and trying to `make` the benchmark suite, it did not complete compiling the code. It relied on Hunter, a package manager used for downloading and installing dependencies [54]. Hunter was required to retrieve files with cURL [55] with SSL enabled, which it was not for our locally compiled CMake. This was a known issue for the benchmark suite and the package manager, but was still difficult to solve.

After trying to compile cURL from source with SSL enabled, it led to having to compile OpenSSL from source as well. At that point we had compiled OpenSSL, cURL and CMake and was ready to compile and download the dependencies using Hunter for the benchmark suite. However, after another attempt of compiling the code it still did not complete. This time due to the fact that some of the source code in one of the benchmark applications relied on newer code syntax and therefore needed a newer version of GCC. At first, we tried installing the latest version of GCC at that point (9.2) and compiled from source for our user. However, the current version of NVIDIA CUDA compiler (NVCC) did not support newer version than 8. After installing the same version (7.4.0) that was used in the Docker containers for the non Power AC922 systems, setting this version in the environment variables paths and specifying to use this version in the compiling, the compilation was successful.

3.5 Benchmark areas

The characteristics of the GPUs and the architecture of the systems affected what we wanted to benchmark and compare.

Benchmarking DGX-2

For the DGX-2 system we wanted to collect benchmark results to see if there were any significant performance differences when the GPUs that are communicating are on one baseboard compared to them being on separate baseboards. If the GPUs are on separate baseboards, the connection would have to traverse two NVSwitches.We also wanted to see if the distance between the GPUs could be relevant, and if there were any differences in which GPU to use when communicating with host.

Benchmarking the Power AC922 systems

When benchmarking the Power AC922 systems we wanted to look into how the systems are different. We wanted to see if it would matter for our tests that Mini Summit (2 GPUs) has 16 GB memory on its GPUs and Yme (4 GPUs) has 32 GB memory per GPU.

We also wanted to see if there were any significant difference when choosing which GPUs to use on the systems, and if some pairs of GPUs would be better for some types of communications.

The main comparisons between the systems were done in this part, but some tests when the number of GPUs were significant were done in the next part.

Comparing DGX-2 and the AC922 Power Systems

This part focuses on the differences of the interconnects of the systems. We also wanted to see how the performance would change when using different amounts of GPUs for the systems.

Comparing single GPUs

This part focuses on comparing the NVIDIA GeForce GTX 980, the NVIDIA Tesla V100 and the NVIDIA Titan RTX graphic cards. We had three different versions of the Tesla V100 cards, where two of them has 32GB GPU memory but are different models (SXM2 and SXM3) and the third is an SXM2 with 16 GB GPU memory.

This comparison consists of testing general performance with different benchmarks, and testing how much the Tensor Cores improves performance on Tesla V100 with Volta architecture and Titan RTX with Turing architecture.

4 Results and Discussion

This section will present and discuss the most interesting results from the areas we wanted to benchmark. There will also be introduced causes or theories for why the results are how they are. Appendix D contains most of the results used for the graphs, and all of the results are found in a GitHub repository referred to in the appendix. In this repository, Python scripts to produce the charts in this chapter are included.

At the end of the section, a summary with the results can be found in a table, and lastly there is an evaluation of the benchmark suites that were used.

4.1 NVIDIA DGX-2

When analyzing and benchmarking the DGX-2 system we performed tests within the system to find out if there were any significant differences between the GPU-CPU communication for different NUMA nodes. We also wanted to investigate if there were any differences in performance for GPU-GPU communication when the GPUs are on the same baseboard compared to them being on different baseboards and needing to traverse two NVSwithes.

GPU-GPU communication with Tartan

The GPU-GPU communication were tested by using the Tartan benchmark suite and its `scale_up_p2p_packet` benchmark. This benchmark provided the results for different data sizes in matrices, as can be seen in Listing 32 in Appendix D. For this section we analyzed the latency and both unidirectional and bidirectional bandwidth for data size `33554432`. The latency were measured in microseconds and the bandwidth in GB/s.

The benchmarking was done for two types of transfer types, for the one called PCIe the connection will go via the host, and for the one called NVLink the connection will go directly from GPU to GPU if possible. When the latency or bandwidth is measured for a GPU to itself, the connection will not go to host and back to GPU.

Figure 6 shows a heat map of the latency between GPU devices when using PCIe connection. The dark purple color represent the worst latency, with a maximum value of around 4000 μs, and the lowest latency is represented by the dark green color. The absolute lowest latency is the latency when a GPU is communicating with itself at around 80 μs and does not traverse interconnects.

The results from the PCIe latency test shows that the latency from a GPU on one baseboard (half of the GPUs) to a GPU on the other baseboard is around the same for all combinations. This makes sense since the transfer data from the sender needs to pass through the senders CPU over the QPI to the receivers CPU and lastly to the receiver GPU. This transfer path is the same for all of these combinations.

When the latency is measured for a GPU on one baseboard to a GPU on the same baseboard, there were some bigger differences than the previous observation. We can see that the latency between a GPU on one half of a baseboard to a GPU on the same half is way worse (with a latency of around 3400 μs to 4000 μs) compared to the latency between a GPU on one half of a baseboard to a GPU on the other half of the same baseboard (which has a latency of around 3000 μs to 3200 μs). These results are somewhat different from the results Li et al. presented in their paper "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect" [34]. In the paper they stated that they could see this effect on the bandwidth results, but it was not observed for the latency results. Our hypothesis is that they did not use a high enough data size

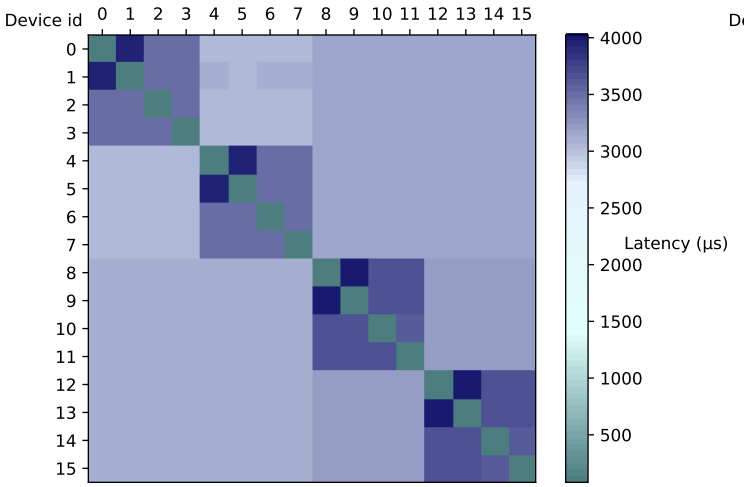

Figure 6: P2P PCIe latency for DGX-2

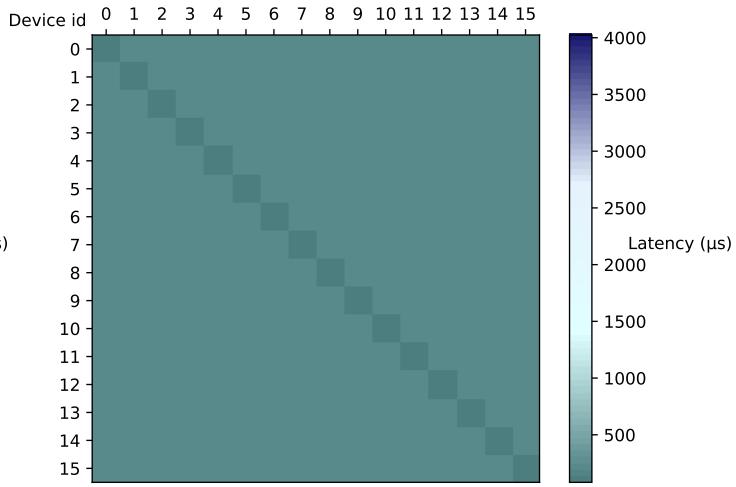

Figure 7: P2P NVLink latency for DGX-2

to see this effect for latency. Their paper call this NUMA effect "anti locality" and the definition is that when data transfer between two GPUs has worse performance when the GPUs are closer compared to when they are further apart. The authors of the paper believes that the anti locality could come from the PCIe-swiches with unbalanced physical signal paths on the chipsets. As we can see in Figure 3 from 2 Background, the connection has to traverse though the same highest level PCIe switch twice, once on the way to the CPU and one back to the other GPU. If the GPUs also share the same lowest level switch, the connection has to traverse two of the switches twice. The fact that the switches has to be used multiple times, strengthen the hypothesis that this could be a PCIe-switch signaling complication.

This type of anti locality will not be relevant to where the GPUs are on different baseboards. This is because the connection will not traverse the same PCIe switches multiple times.

Another difference with our results compared to the results from the paper is that in our chart we can see worse performance for the first pair in these halves of the baseboards. The pairs with worst latency are GPU 0 and GPU 1, GPU 4 and GPU 5, GPU 8 and GPU 9, and GPU 12 and GPU 13. The reason this is not shown in the paper could also be that the data size is too small. Further tests need to be done to identify the reason for this result.

Another interesting observation is that there is lower latency on the first baseboard when a GPU from one half of the baseboard communicates with one from the other half of the baseboard, compared to the second baseboard. For example, there is lower latency when GPU 4 and GPU 0 communicates, compared to when GPU 12 and GPU 8 communicates. This could be such a small difference that it is not necessarily important. If this result were to appear more, it could potentially be that the first CPU is the "main" CPU, and communication has to go through it.

Figure 7 shows the same type of heat map, with the same colour scale as the previous chart, but for NVLink connection. This result shows us that there is negligible difference in latency for the GPU-GPU communication thorugh NVLink, even when the connection crosses two NVSwitches to transfer data from one baseboard to the other baseboard.

Since the two heat maps uses the same colour scale, we can see that the latency is much lower for the NVLink communication than for the PCIe communication. This is expected because NVSwitch is supposed to be much faster than PCIe.

Figure 8 shows the P2P unidirectional bandwidth for PCIe and NVLink in a logarithmic scale because of the enormous value differences. In Figure 9 the bididirectional bandwidth for PCIe and NVLink is shown in the same scale. These results are similar to the latency results, but not as prominent.

The results shows that there are better performance for the bandwidth when the communication is bidirectional than when it is unidirectional. This is probably because when the communication goes both ways, there are more data transferred.

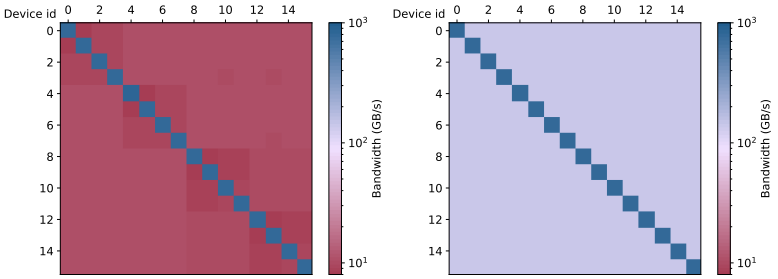

Figure 8: P2P unidirectional bandwidth for DGX-2. Right: PCIe. Left: NVLink

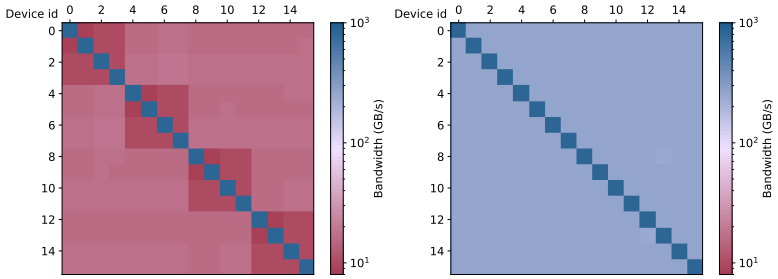

Figure 9: P2P bidirectional bandwidth for DGX-2. Right: PCIe. Left: NVLink

CPU-GPU communication with Scope

Scope was used for evaluating the communication between CPU and GPU. The results in Table 11 shows some results from CPU-GPU Memcpy with four different GPUs. There are not any significant differences between the GPUs when the data transfers from GPU to CPU. There are however some differences between the GPUs when host is transferring data to the GPU. There could be many different reasons why this is the case. More benchmark data must be collected to find the reason.

	GPU 0	GPU 1	GPU 14	GPU 15
GPUToHost	1.47e+09	1.48e+09	1.47e+09	1.48e+09
HostToGPU	4.66e+09	4.79e+09	4.74e+09	4.81e+09
GPUToWC	1.31e+10	1.32e+10	1.32e+10	1.32e+10
WCToGPU	1.15e+10	1.19e+10	1.19e+10	1.15e+10

Table 11: Scope|COMM Memcpy results for data size 30. The unit is bytes per second.

4.2 IBM Power System AC922

The Power AC922 systems were benchmarked by performing tests on each of the systems. Tests were performed to reveal if there were any performance differences on when using different GPUs. This section also contains some comparisons between the systems regarding which GPUs to use for communication.

GPU-GPU communication with Tartan

Tartan was used to evaluate the GPU-GPU communication for the Power AC922 systems in the same way, with the same data size, as for the DGX-2 system. Listing 30 and Listing 31 from Appendix D shows the values from the benchmark from Yme (4 GPUs) and Mini Summit (2 GPUs) respectively.

Figure 10 shows the results from the latency benchmark for Yme (4 GPUs). We can immediately see that the latency between GPU 0 and GPU 1 (both ways) are much lower than the latency between GPU 2 and GPU 3 (both ways). The first two GPUs are directly connected to the first CPU, and the other two are connected to the second CPU. A possible reason for these results could be that CPU 0 is the "main" CPU, and all communication has to go through it.

The NVLink latency between two GPUs are shown in Figure 11. We can see that the lowest latency, not taking the communication with itself into account, is between the GPUs with direct NVLink connection between them, GPU0-GPU1 and GPU2-GPU3, in their own NUMA node. The highest latency is when a GPU communicates with a GPU on the other NUMA node. This makes sense, since the connection has to traverse through two CPUs.

Since the same colour scale is used for both heat maps, we can see that there are differences in latency for the maps when a GPU on one NUMA node is communicating with a GPU on the other NUMA node. The communication will traverse though both CPUs. The reason why the latency is lower for the NVLink mode is probably because the function that is used to make the connection traverse through NVLinks, `cudaDeviceEnablePeerAccess`, enables direct access to memory allocations on a peer device, and for PCIe communication the GPU has to access memory from CPU.

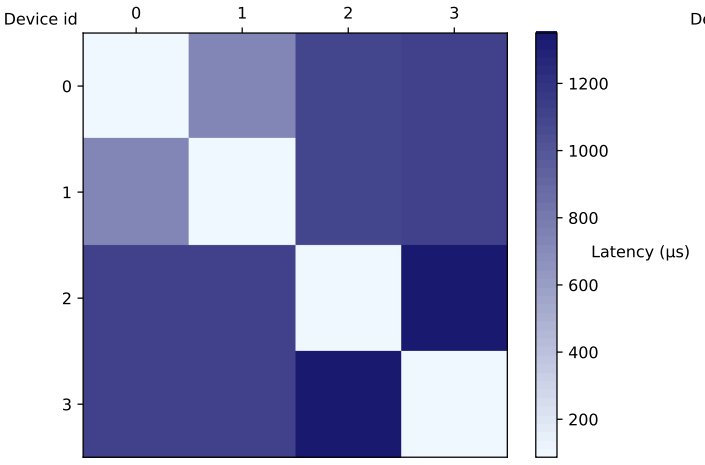

Figure 10: P2P PCIe latency for Yme (Power AC922, 4 GPUs)

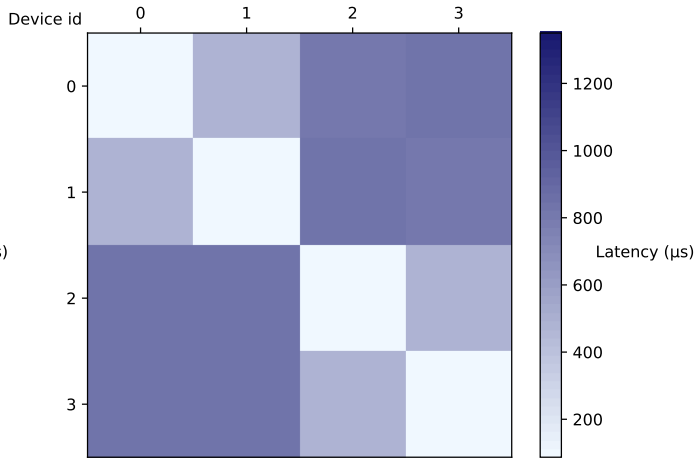

Figure 11: P2P NVLink latency for Yme (Power AC922, 4 GPUs)

Figure 12 and Figure 13 shows the latency for Mini Summit (2 GPUs) with PCIe and NVLink respectively. Since these charts has the same colour scale as the latency charts for Yme, we can see that the latency is slightly lower for Mini Summit compared to Yme when two GPUs on different NUMA nodes are communicating. For PCIe the latency is around 1090 μs for Mini Summit, and around 1110 μs for Yme. The NVLink latency is around 815 μs for Mini Summit, and around 822 μs for Yme. Since the difference is so minimal, we will wait to discuss this result until any other similar results appear.

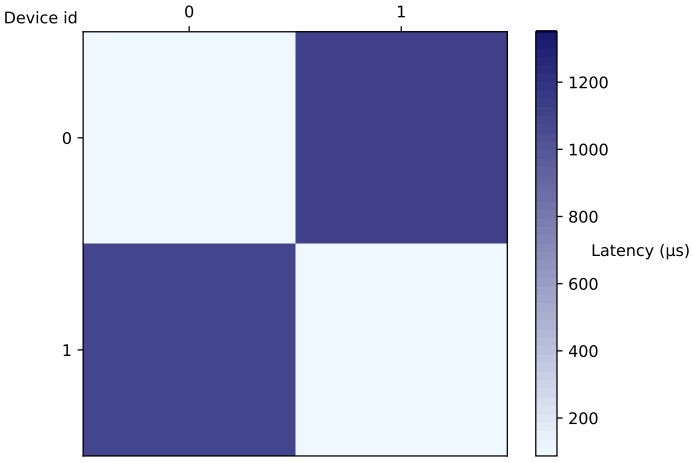

Figure 12: P2P PCIe latency for Mini Summit (Power AC922, 2 GPUs)

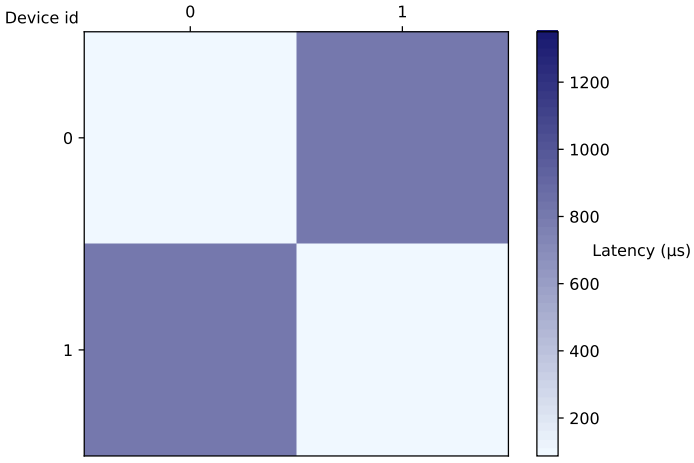

Figure 13: P2P NVLink latency for Mini Summit (Power AC922, 2 GPUs)

Figure 14 and Figure 15 shows the unidirectional and bidirectional bandwidth for Yme respectively. Figure 16 and Figure 17 also shows the unidirectional and bidirectional bandwidth, but for Mini Summit respectively. These results shows the same results that latency does.

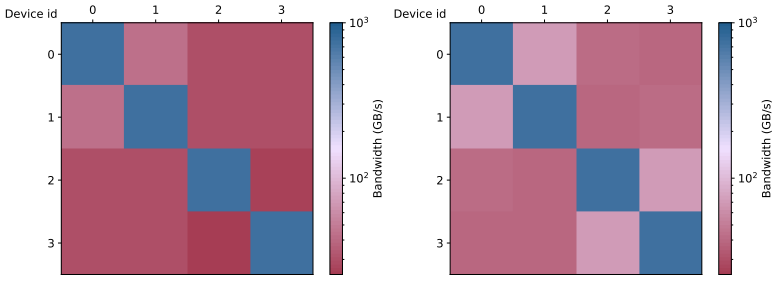

Figure 14: P2P unidirectional bandwidth for Yme (4 GPUs)
Right: PCIe. Left: NVLink

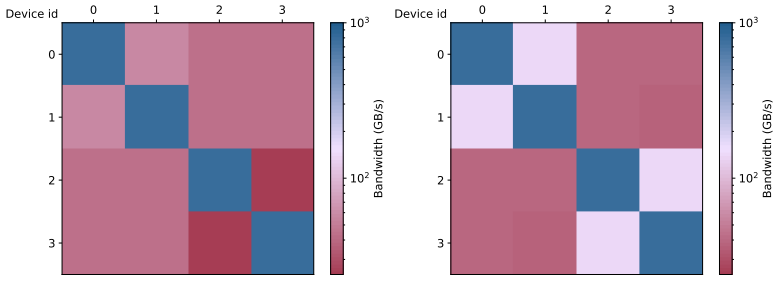

Figure 15: P2P bidirectional bandwidth for Yme (4 GPUs)
Right: PCIe. Left: NVLink

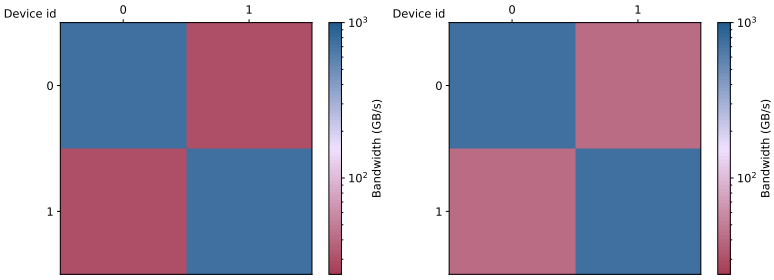

Figure 16: P2P unidirectional bandwidth for Mini Summit (2 GPUs)
Right: PCIe. Left: NVLink

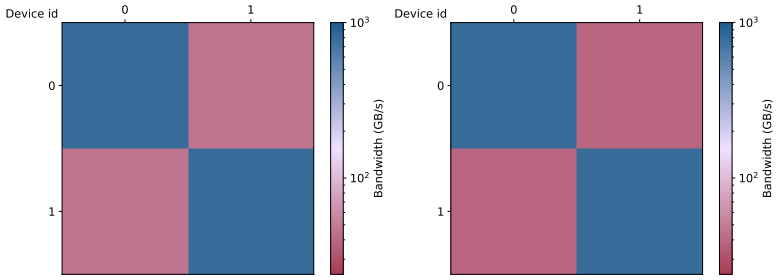

Figure 17: P2P bidirectional bandwidth for Mini Summit (2 GPUs)
Right: PCIe. Left: NVLink

CPU-GPU communication with Scope

Scope|COMM with data size 30 was used to evaluate the communication between specific GPUs and host. In Figure 18 and Figure 19 shows the Memcpy operation from GPU to host and host to GPU respectively for each specific GPU. Yme (4 GPUs) has two NUMA nodes, the two first GPUs are on the first NUMA node, and the two second GPUs are on the second NUMA node. Mini Summit (2 GPUs) also has two NUMA nodes with one GPU each. Both of these bar charts are quite similar in the way that the performance is better when using GPUs from the first NUMA node compared to using GPUs from the second NUMA node, this applies to both Power AC922 systems.

The performance of the first GPU on each NUMA node for both systems is slightly better for Mini Summit than for Yme. One possible reason for this could be that Mini Summit has one CPU for each GPU, but Yme has to share the CPU between two GPUs.

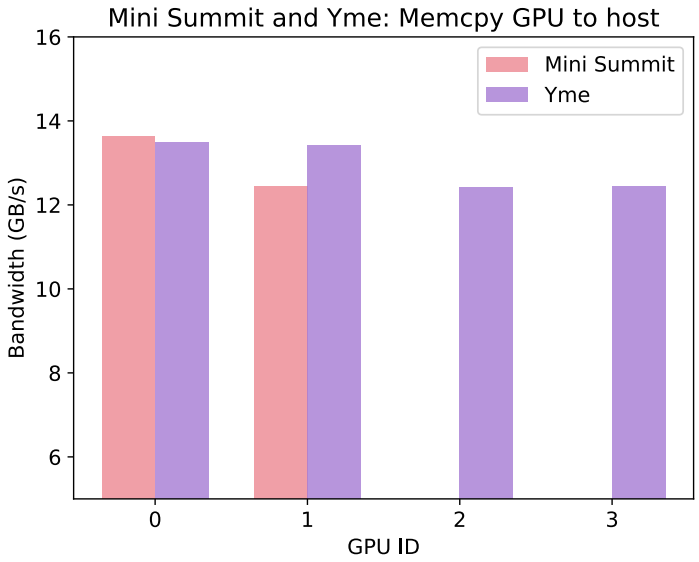

Figure 18: Memcpy GPU to host: Mini Summit and Yme

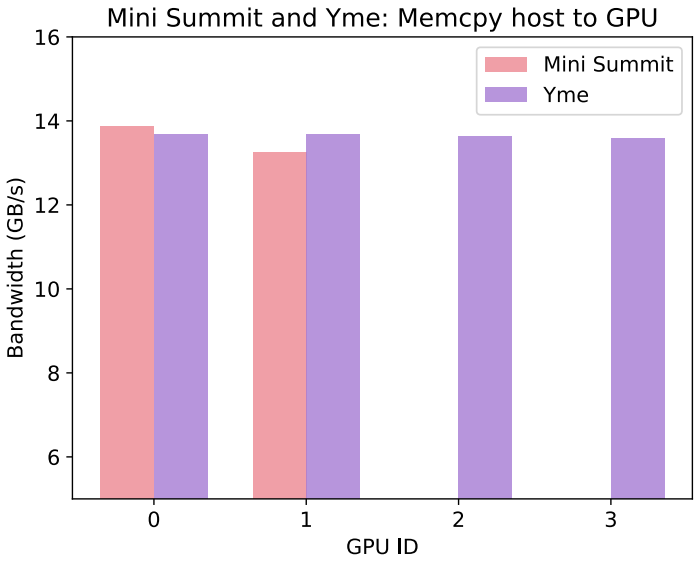

Figure 19: Memcpy host to GPU: Mini Summit and Yme

Figure 20 and Figure 21 shows the differences between the NUMA nodes better than the last figures. We can see that the bandwidth when performing Memcpy from a GPU on the first NUMA node to Write combining host is around double the bandwidth of the Memcpy from a GPU on the second NUMA node. Most of the data sizes shows these results. We did not have enough time to analyze why these benchmarks got these results, but our hypothesis is that the first CPU is the "main" CPU, and all communication has to go through it.

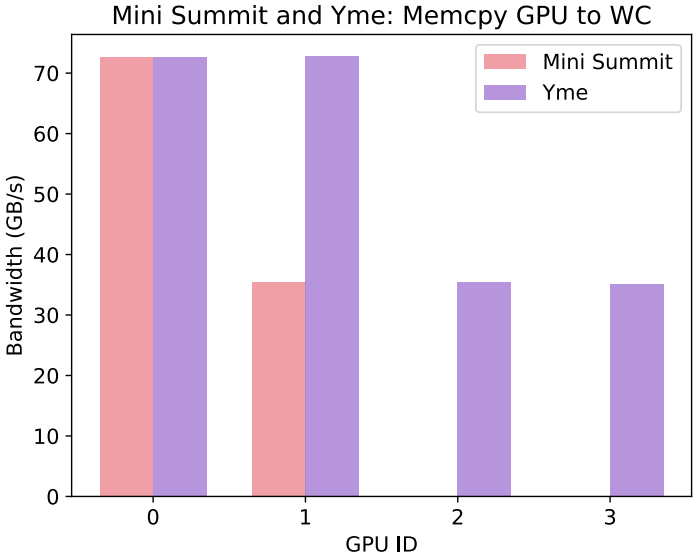

Figure 20: Memcpy GPU to WC: Mini Summit and Yme

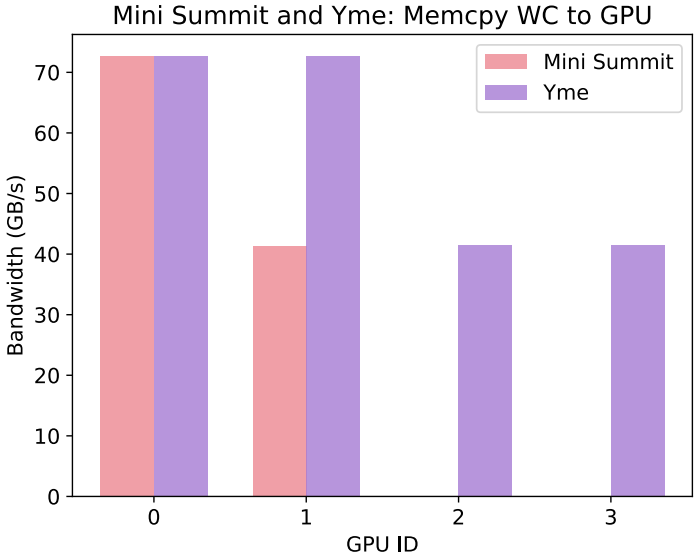

Figure 21: Memcpy WC to GPU: Mini Summit and Yme

4.3 NVIDIA DGX-2 and IBM Power System AC922 comparison

In this section, results that compares NVIDIA DGX-2 and the IBM Power System AC922 machines are presented. The focus is on showing how the performance changes when using different amounts of GPUs for the different systems.

GPU-GPU communication on one NUMA node with Tartan

Tartan was used for measuring the unidirectional bandwidth from a GPU to another GPU in the same NUMA node. This comparison was only done with the DGX-2 and Yme, the Power AC922 with four GPUs, because Mini Summit only has two GPUs where each of them are on different NUMA nodes.

Figure 22 and Figure 23 shows the unidirectional bandwidth for PCIe and NVLink communication respectively. The values are averages of all the measurements of same NUMA node communication. For the PCIe communcation, the bandwidth for Yme is on average around 35 GB/s. This is because the bandwith between the two GPUs on the first NUMA node is around 45 GB/s, and on the second NUMA node around 25 GB/s. The DGX-2 shows a beandwidth of around 10.5 GB/s on average. This is way worse than the Power AC922 results, and shows the significance of using NVLink interconnects to host instead of PCI Express. When using the PCIe mode, the connection will go via host.

The NVLink communication shows how the performance turns, this time the DGX-2 has the best perfomance with an average of around 137 GB/s. The bandwidth Yme presents is around 71 GB/s. We can see that the performance for the DGX-2, with NVSwitch, is around double the performance for the Power AC922, which uses NVLink between the GPUs. The reason why the performance doubles is probably because for the DGX-2 the connection traverses a bounded set of six NVLinks, whereas for Yme it is used three NVLink bricks between the GPUs on the same NUMA node.

The results shows that using NVLink as connection between GPU and CPU is better than PCIe, and using NVSwitch between GPUs is better than NVLink.

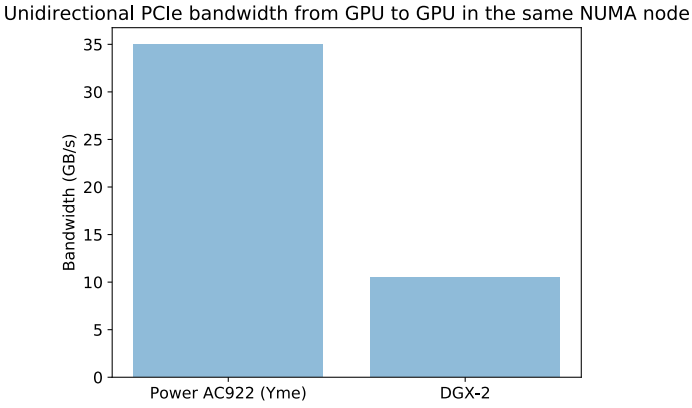

Figure 22: Unidirectional PCIe bandwidth from GPU to GPU in the same NUMA node

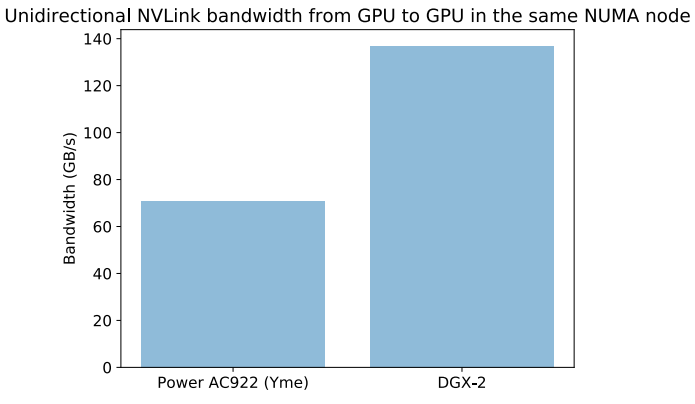

Figure 23: Unidirectional NVLink bandwidth from GPU to GPU in the same NUMA node

CPU-GPU bus speed with SHOC

To further investigate the CPU-GPU bus, SHOC was used to measure the speed between the GPU and the CPU.

Figure 24 and Figure 25 shows the bandwidth of GPU to one CPU for download and readback respectively. The value is the average of the measurements for each GPU-CPU bus speed. These results shows again how much faster the CPU-GPU connection is with NVLinks compared to PCIe. Mini Summit shows again that its performance is slightly better than Yme.

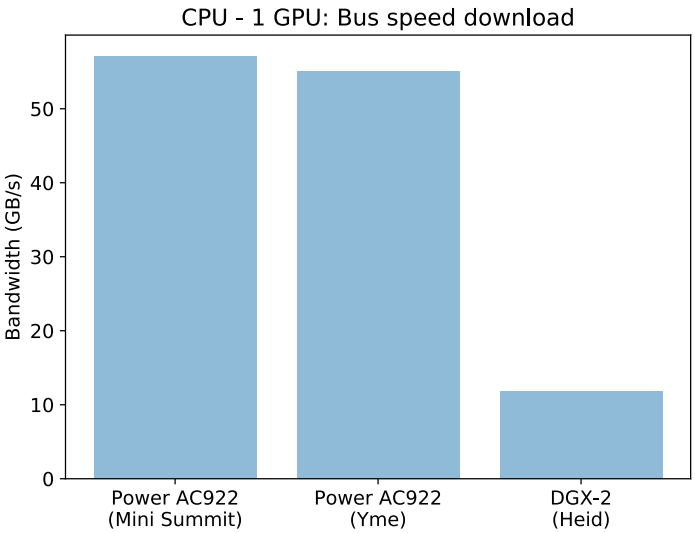

Figure 24: Bus speed download CPU-GPU

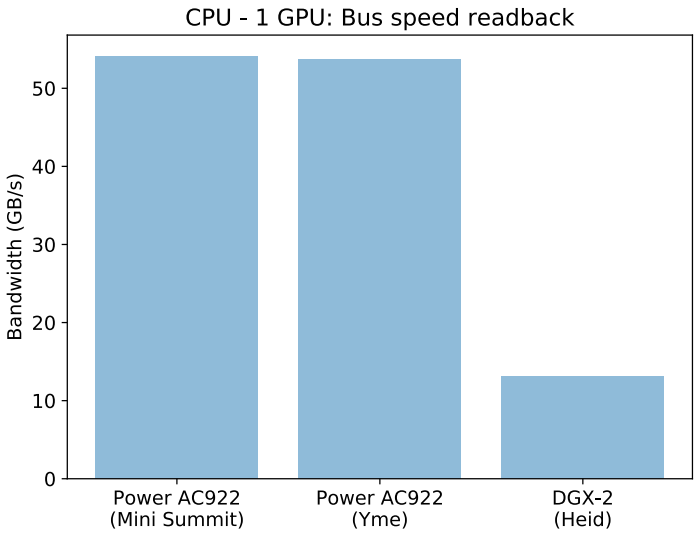

Figure 25: Bus speed readback CPU-GPU

NCCL Multi-GPU communication with DeepBench and Scope

To evaluate the multi-GPU communication we used DeepBench and Scope with NCCL. The GPUs are chosen in order, which means that if two GPUs are used, they are the two first GPUs.

Figure 26 shows a graph with time in ms when performing AllReduce with NCCL MPI. The three systems are shown with values up to their total number of GPUs. Mini Summit, the Power AC922 with 2 GPUs, has a much higher time usage with two GPUs than the other two systems. This is because it does not have a direct connection between the GPUs. Yme, the Power AC922 with 4 GPUs, has much better performance when using two GPUs compared to Mini Summit. The performance decreases significantly when three or four GPUs are used. This happens for the same reason as for the other Power AC922 system, the connection has to traverse through two CPUs. For Heid, the DGX-2 system, we can see that the performance is better than the other system for every number of GPU. There is a slight time increase for every time the GPU number increases. These results shows that the DGX-2 is more scalable.

In these results we can see the same as the previous result, that using NVSwithes with six NVLink bricks between all GPUs is faster than using three NVLink bricks between GPUs in both halfes of the system, when it comes to GPU-GPU communication without the need to involve CPUs.

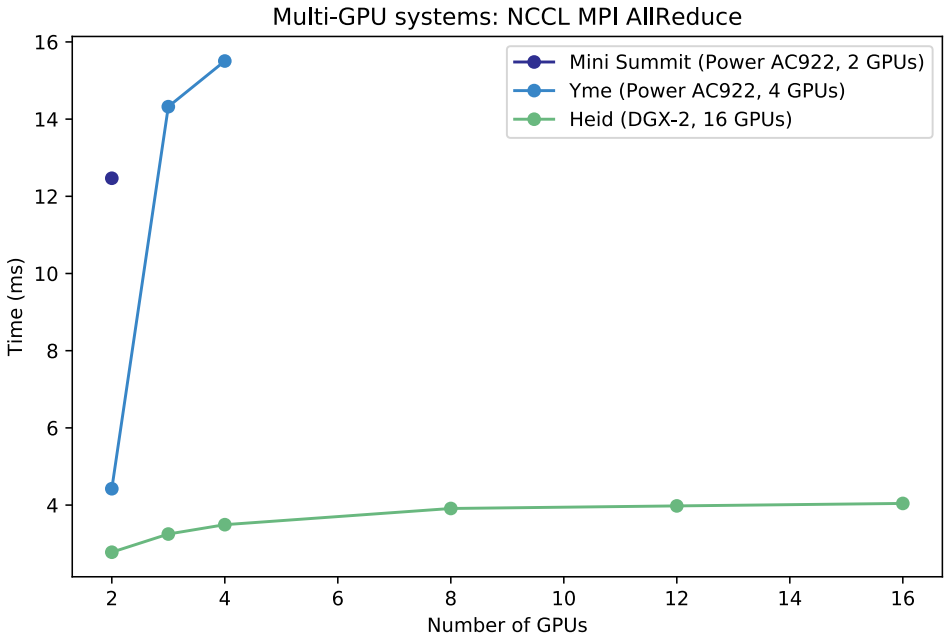

Figure 26: NCCL MPI AllReduce with DeepBench

In Figure 27 the broadcast operation with NCCL is evaluated with the Scope benchmark suite. These results also includes numbers for using only one GPU. The DGX-2 system has better performance than the other systems for every amount of GPUs. We can see the same trend in this graph as the previous, when looking at how the performance changes for the Power AC922 systems when using GPUs where the connection has to traverse through two GPUs.

Another point to notice is that Mini Summit has a slightly better performance than Yme when using one GPU.

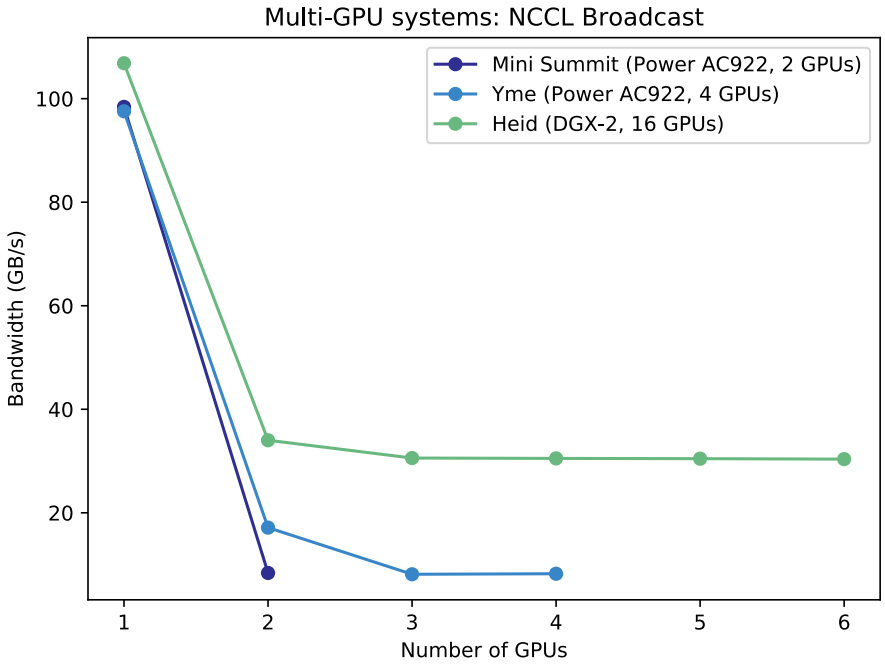

Figure 27: NCCL Broadcast with Scope

Multi-GPU communication with SHOC

SHOC was also used for evaluating the multi-GPU communication for the systems. Size 4 was used. In Figure 28 the time in seconds is shown for the Quality Threshold Clustering (QTC) algorithm. This benchmark is a real application kernel and PCIe transfer is included in the time.

Since the number of GPUs starts on two, the communication has to traverse through the CPUs for Mini Summit. Mini Summit is still the system with the best performance in this instance, which can be surprising considering the previous results that shows the opposite. Our hypothesis is that it has better performance because it uses one GPU on each NUMA node, where both has its own CPU. For the other two systems the two first GPUs are chosen, which means that they are on the same NUMA node and share the same CPU.

For Yme and the DGX-2 we can see that the time is improving when GPUs are added. A difference between this results and the previous results is that Yme and DGX-2 has almost the same performance when using four GPUs. This leads us to believe that this benchmark uses a lot of CPU interaction instead of just GPU communication. Maybe because this is a real application kernel benchmark.

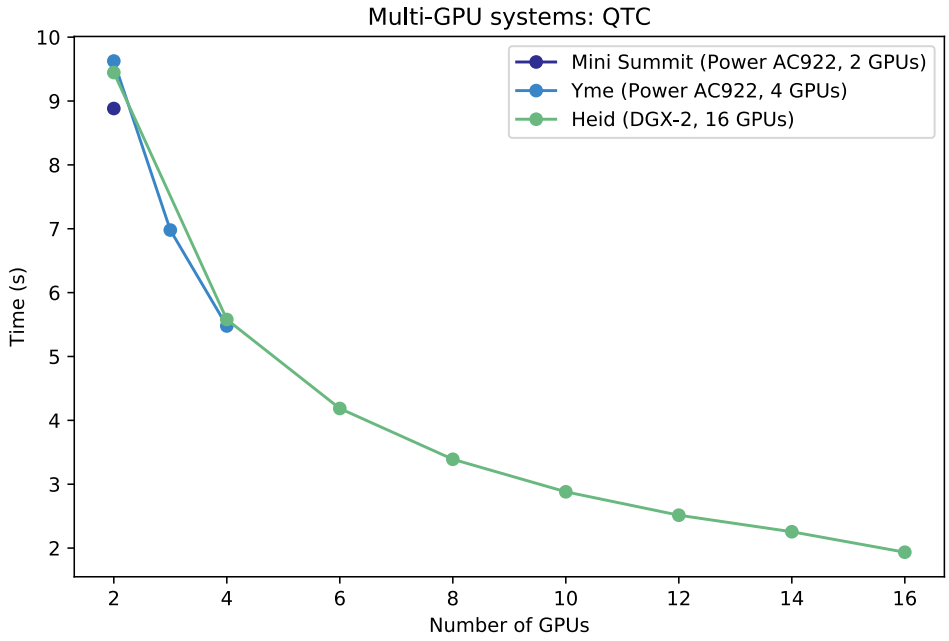

Figure 28: QTC for multi-GPU systems

Figure 29 illustrates the performance in GFLOPS when performing the single precision Stencil 2D test. It shows graphs for two different scenarios for the systems. The solid lines shows the performance when the chosen GPUs are on the same NUMA node if possible. The dashed lines illustrates the performance when the GPUs are on different NUMA nodes. The performance from dashed line for the DGX-2 system on four GPUs are GPU IDs 0, 1, 14 and 15, two on each NUMA node. But the solid line shows the performance for GPU IDs 0, 1, 3 and 4, all on the same NUMA node. When the number of GPUs increase to over half of the total, the results will be the same for both type of lines, since the GPUs will be on different NUMA nodes.

These results shows the same results about Mini Summit's performance as the last graph.

When comparing the performance for different NUMA nodes compared to the same NUMA nodes for Yme and DGX-2, we can see that the performance when using different NUMA nodes is often slightly better. For the DGX-2, the performance when using different NUMA nodes are better on 2 and 4 GPUs, and almost the same on 6 and 8 GPUs. Since Mini Summit has the best result for two GPUs, we can assume that there are some CPU communication involved. If there is a lot of CPU communication, it makes sense that using GPUs on different NUMA nodes leads to better performance because of the anti locality principle for the DGX-2.

This line chart also shows some interesting results regarding the DGX-2 lines. The performance drops between 2 GPUs and 8 GPUs. A drop in performance is also the case for Yme, and then the performance exceeds the performance of the DGX-2 for four GPUs. We did not have enough time to analyze why these benchmarks got these results.

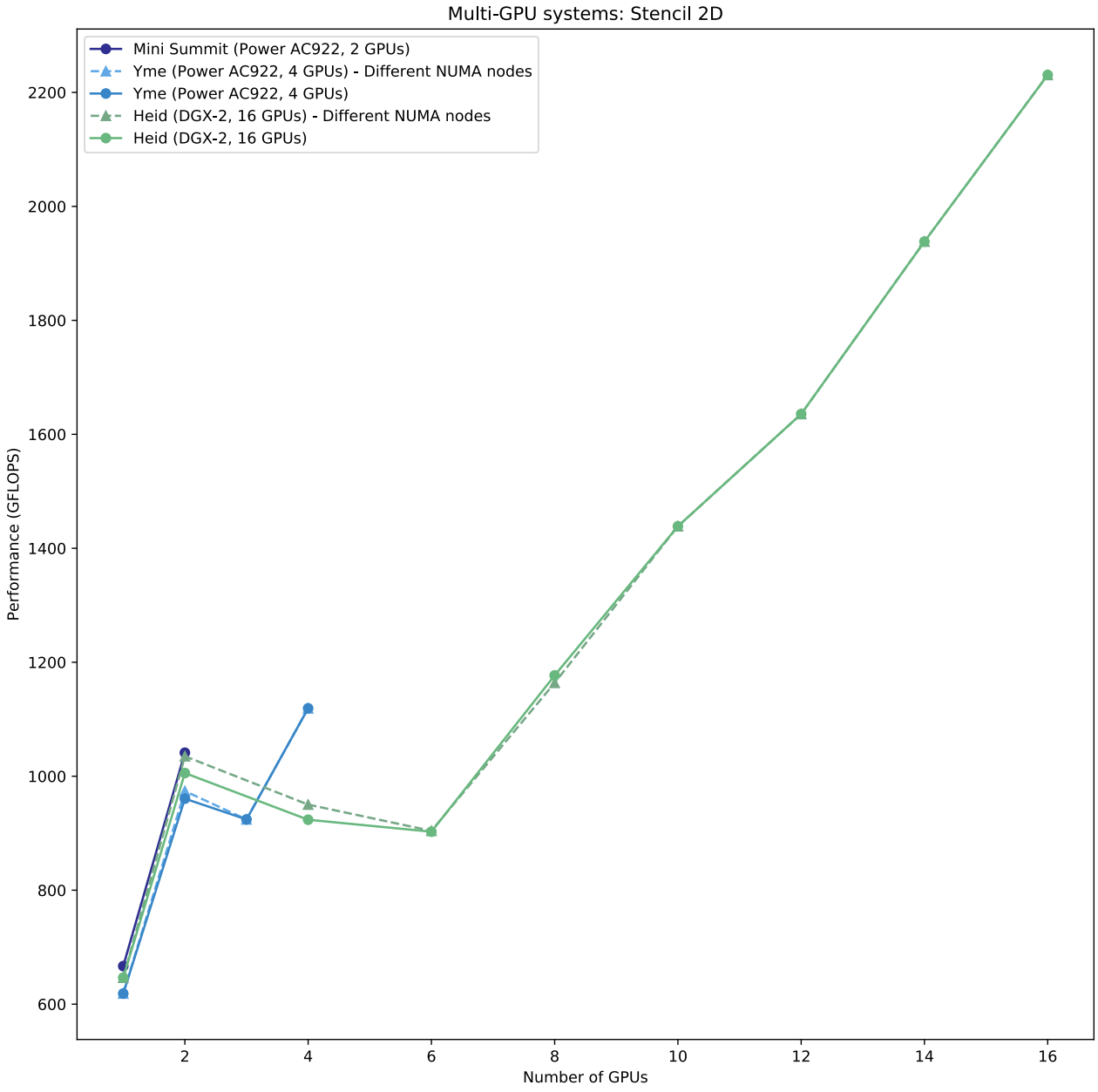

Figure 29: Stencil 2D

4.4 GeForce GTX 980, Tesla V100 and Titan RTX comparison

This section focuses on comparing the general performance and NVIDIA Tensor Core performance of GeForce GTX 980, Tesla V100 and Titan RTX. There is also focus on comparing the different Tesla V100 graphic cards: Tesla V100 SXM3 with 32 GB memory from DGX-2, Tesla V100 SXM2 with 32 GB memory from Yme (Power AC922, 4 GPUs) and Tesla V100 SXM2 with 16 GB memory from Mini Summit (Power AC922, 2 GPUs).

General performance with SHOC

SHOC was used to generate results for general performance, and MaxFlops with single and double precision were used as the benchmark.

Figure 30 and Figure 31 shows the performance in GFLOPS for all of the GPUs when performing the MaxFlops test in single and double precision respectively. The results shows that for single precision Titan RTX is quite a bit better than the rest of the GPUs. For double precision, on the other hand, the performance is remarkably worse than the Tesla V100 cards, and it is not that much better than the GTX 980 card. This is probably because the Titan RTX card is adapted to have great deep learning performance. For deep learning applications the precision is not always as important as it can be for HPC applications [21], and thus lower precision is prioritized.

The GeForce GTX 980 graphics card has the worst performance for both bar charts, as expected considering its specifications.

The MaxFlops benchmark does not include PCIe transfer, which means that the Tesla V100 from the Power AC922 systems does not get an advantage when it comes to this performance comparison. The Tesla V100 SXM3 (32 GB) from the DGX-2 has the greatest performance of the Tesla V100 cards for both precisions. This is probably not because of the memory size, since both of the Tesla V100 SXM2 with same memory has almost identical performance. The fact that the tests on the SXM2 cards were not run using Docker is probably not a reason for the SXM2 cards to perform worse than the SXM3 card. The reason could in all likelihood be because of the different model versions.

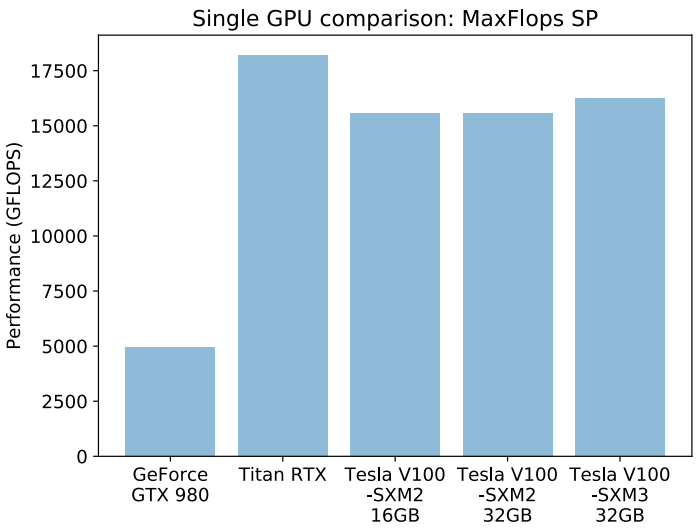

Figure 30: Single GPU comparision: MaxFlops SP

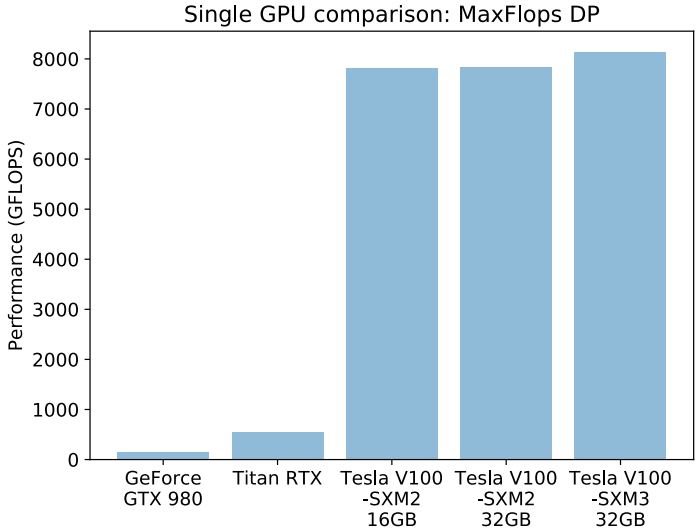

Figure 31: Single GPU comparision: MaxFlops DP

Tensor Core performance with DeepBench

To evaluate how the GPUs utilized their Tensor Cores we used the DeepBench benchmark suite. The Titan RTX card has 576 Tensor Cores, the Tesla V100 cards has 640 Tensor Cores and the GTX 980 does not have any Tensor Cores.

Figure 32 and Figure 33 are both bar charts with results from GEMM Training with FP32 precision. The difference between the charts are the matrix sizes used for the GEMM operation. The first chart uses the values (M, N, K, a_t, b_t) = (7680, 48000, 2560, 0, 0) and the second chart uses (M, N, K, a_t, b_t) = (7680, 5481, 2560, 0, 1). The difference between the values are N (one of the dimensions of the second matrix) and b_t (the second matrix is transposed). The difference between these sizes is that the N value for the second chart is not dividable by four, but the N value for the first chart is.

Figure 32 uses the matrix sizes that is dividable by four and all of the GPUs that has Tensor Cores shows a great decrease in time when the cores are enabled. Figure 33 shows that using a matrix size that is not dividable by four leads to a very small time decrease when Tensor Cores are enabled for the GPUs that has Tensor Cores. This leads us to believe that by using matrix dimensions that will be dividable by four, the Tensor Cores will be more useful. This makes sense because the Tensor Cores can only perform operations on 4x4 matrices.

Titan RTX cards have new Turing Tensor Cores that are better for inferencing and has modes for int8 and int4 precision. For the benchmark in Figure 32, the Titan RTX card does not improve as much as the other graphic cards that has Tensor Cores when the cores are enabled. A reason for this might be that the card is not optimized for higher precision and it has less Tensor Cores than the Tesla V100 cards.

An observation from Figure 32 is that the performance for both of the Tesla V100 SXM2 cards are worse than the performance for the SXM3 card when Tensor Cores are not enabled. But when the Tensor Cores are enabled, the SXM2 cards have slightly better performance than the SXM3 card and notably better improvement percentage. Since the SXM2 cards are better in only one of the scenarios it leads us to believe that the reason can not be anything external of the GPU. The

main difference between SXM2 and SXM3 is that the SXM3 has higher clock rate, which could normally lead to better performance. There should be performed more tests to identify the reason for this result.

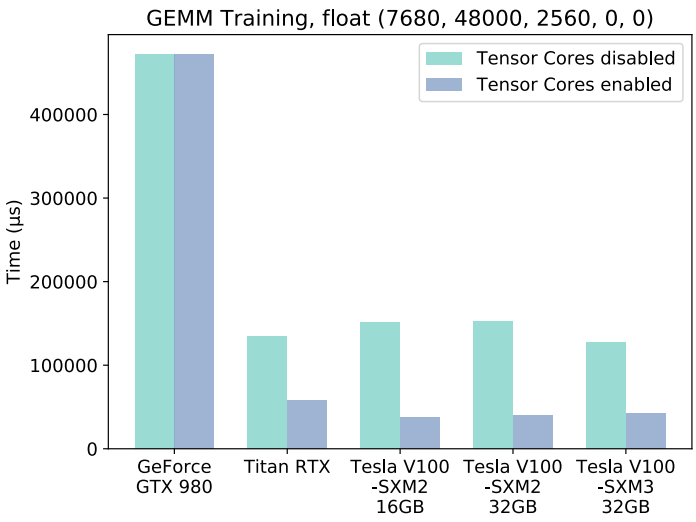

Figure 32: GEMM Training FP32. (7680, 48000, 2560, 0, 0)

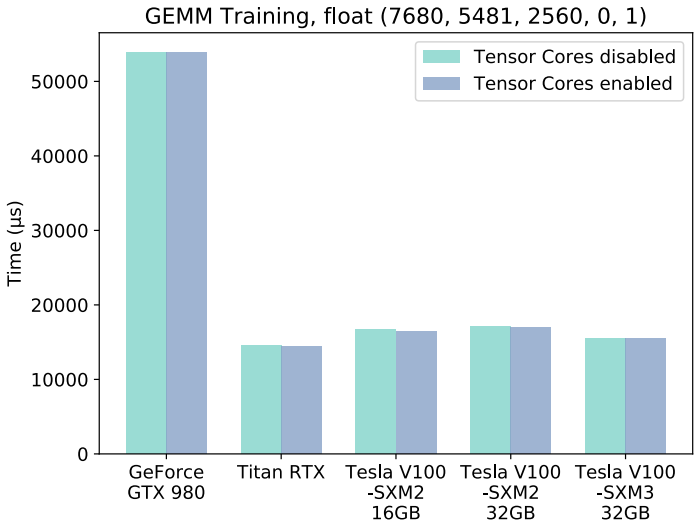

Figure 33: GEMM Training FP32. (7680, 5481, 2560, 0, 1)

Since there are plenty of benchmark results for different matrix sizes, we wanted to see how the GPUs performed as a total for each category. We added up all of the times to get a total time for all the operations.

Figure 34 and Figure 35 shows the time for all of the operations added for GEMM Training FP32 and GEMM Inference FP32 respectively. The GeForce GTX 980 value is way above the chart for both figures. For both of these bar charts we can see the same result with the Tesla V100 cards. The SXM2 cards starts off worse than the cards with Tensor Cores and has the best performance when Tensor Cores are enabled.

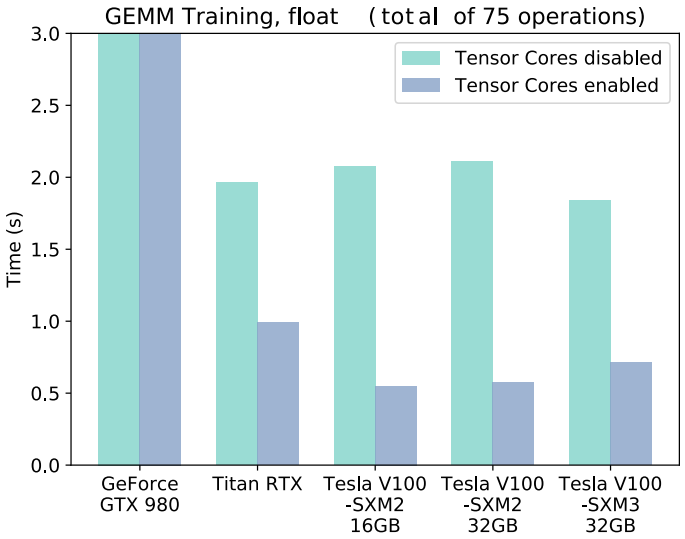

Figure 34: GEMM Training FP32.
Total sum of 75 operations.

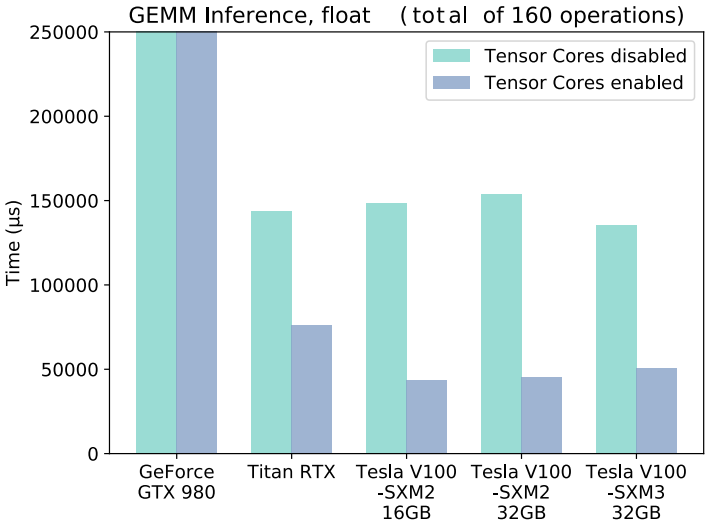

Figure 35: GEMM Inference FP32.
Total sum of 160 operations.

Since the Titan RTX card has new Turing Tensor Cores that should be better at INT8 operations, we added up the values for two more categories. Figure 36 shows GEMM Inference INT8 and Figure 37 shows Convolutions Inference INT8. The benchmark suite did not work for the GTX 980 cart with INT8 precision, which is why it is not included in the new charts.

Figure 36 shows no difference in when Tensor Cores are enabled or disabled. Figure 37 shows that Titan RTX is the only graphics card with improvement when enabling Tensor Cores, it is almost half of the time with Tensor Cores.

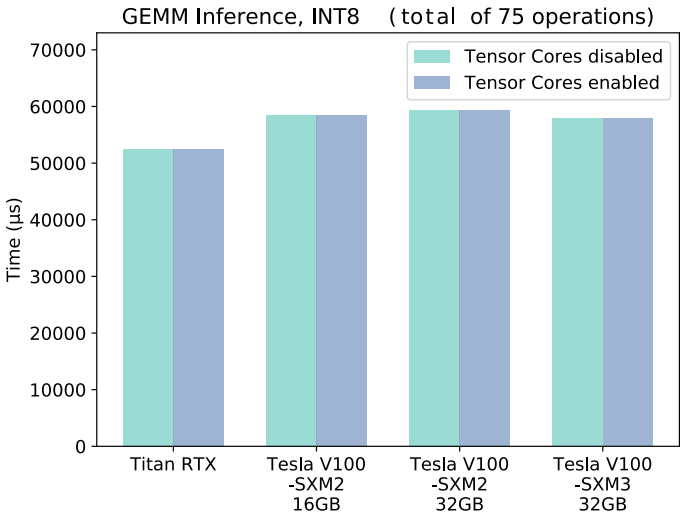

Figure 36: GEMM Inference INT8.
Total sum of 75 operations.

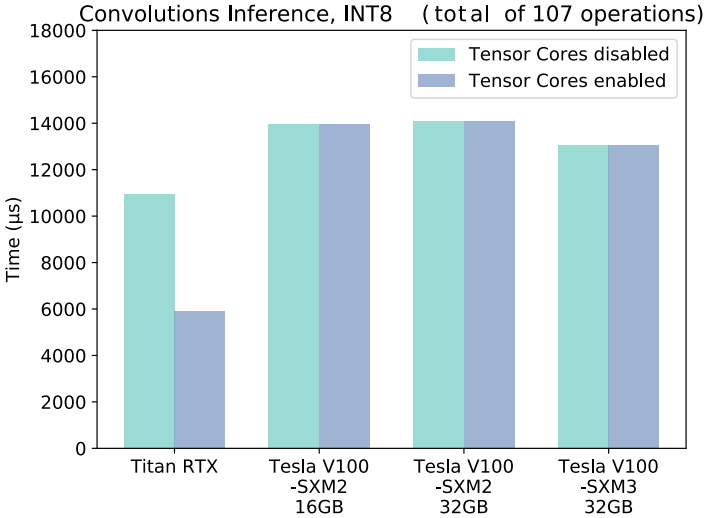

Figure 37: Convolutions Inference INT8.
Total sum of 107 operations.

When we look at Tensor Core performance, it is the improve percentage that is the most interesting part, since the general performance is different for the GPUs anyway.

To give the Titan RTX a fair chance with its 576 Tensor Cores, we calculated what the performance would be if it would have had the same amount of Tensor Cores as the Tesla V100 cards, 640 cores. After this, we found the percentage of time that using Tensor Cores lead to when comparing usage and no usage of Tensor Cores. One other thing to note is that Titan RTX and Tesla V100 is not the same type of cards for different generations, which makes the comparison even harder.

Figure 38 shows the percentage of the time when using Tensor Cores compared to not using Tensor Cores. We can see that for this figure that shows GEMM Training (FP32), Titan RTX improved the non-Tensor Core time to around 45 % when the Tensor Cores are enabled. It is still the SXM2 cards that has the best improvement, where when the Tensor Cores are activated it leads to almost a quarter of the original time.

Figure 39 shows that when the Titan RTX has activated Tensor Cores for Convolutions Inference (INT8), it is around 40 % of original time.

All of these results shows that Titan RTX is great at lower precision calculations, and the Tesla V100 cards are better at the higher precision calculations.

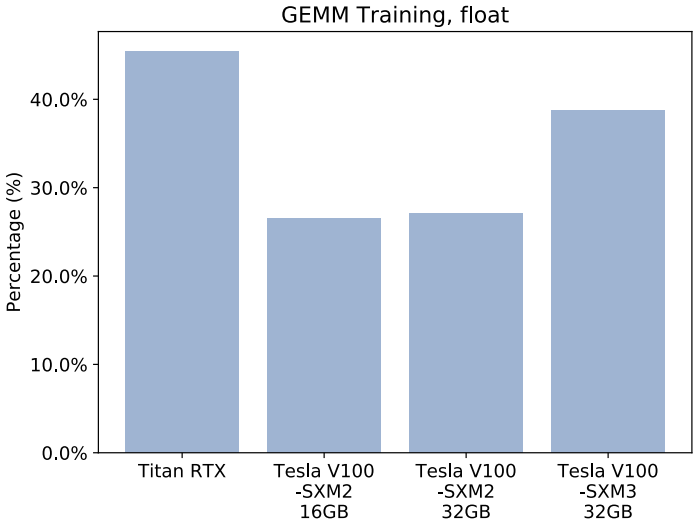

Figure 38: GEMM Training (FP32): Time percentage with Tensor Cores compared to not using Tensor Cores

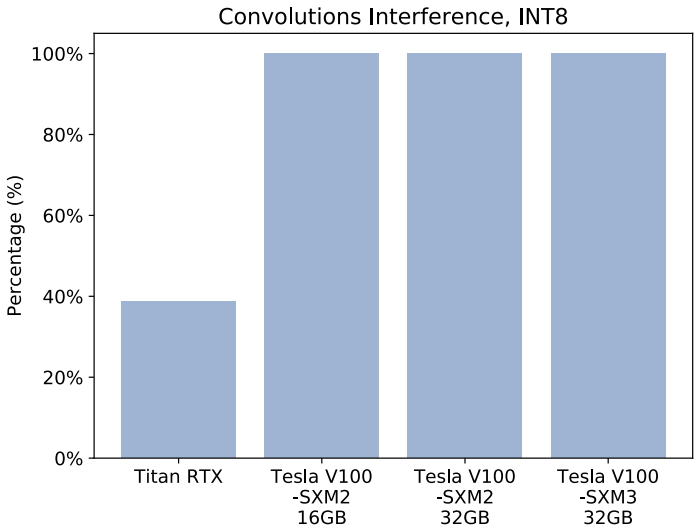

Figure 39: Convolutions Inference (INT8): Time percentage with Tensor Cores compared to not using Tensor Cores

4.5 Result summary

This section contains four tables with summaries of the results found earlier and a small discussion part for each of the results.

Type	Result	Discussion
NVIDIA DGX-2	PCIe communication: GPUs that are closer show worse performance communicating than GPUs that are further apart.	This is because of the anti locality effect, that could be because of PCIe-switch signaling complications.
	PCIe communication: Every other GPU neighbour pair shows worse performance than the rest of the neighbour pairs.	More tests needs to be done to identify a reason for this.
	PCIe communication: Lower latency on the first baseboard when a GPU from one half of the baseboard communicates with one from the other half of the baseboard, compared to the second baseboard.	More tests needs to be done, but this could possibly be because of a "main" CPU.
	No significant difference in performance when two GPUs communicates with each other through NVSwitches no matter which pair.	This shows that data transfer over two NVSwithes are just as effective as over one NVSwitch.
	Connection through NVSwitches is much faster than connection through PCIe and host.	This is expected since NVSwitches are faster than PCIe interconnects.
	Bandwidth results shows similar performance as for latency.	This is since both benchmarks communicates over interconnects.
	There is better performance for the bandwidth when the communication is bidirectional than when it is unidirectional.	This is probably because when the communication goes both ways, there are more data transferred.
	There are not any significant differences between the specific GPUs when the data transfers from GPU to CPU. There are however some differences between the GPUs when host is transferring data to the GPU.	There could be many different reasons why this is the case. More benchmark data must be collected to find the reason.

Table 12: Summarized results and discussion for DGX-2

Type	Result	Discussion
IBM Power System AC922 comparison	Yme: The PCIe latency between GPU 0 and GPU 1 (both ways) are much lower than the latency between GPU 2 and GPU 3 (both ways).	A possible reason for these results could be that CPU 0 is the "main" CPU, and all communication has to go through it.
	Yme: The lowest NVLink latency is between the GPUs with direct NVLink connection between them. The highest latency is when a GPU communicates with a GPU on the other NUMA node.	This is because the GPUs on the same NUMA node have a NVLink connection between them. When the GPUs are on different NUMA nodes the connection has to traverse through two CPUs.
	Yme and Mini Summit: When a GPU on one NUMA node is communicating with a GPU on the other NUMA node there is a difference in latency depending on if PCIe mode or NVLink mode is used. The latency is lower for the NVLink mode	The reason why the latency is lower for the NVLink mode is probably because the function that is used to make the connection traverse through NVLinks, `cudaDeviceEnablePeerAccess`, enables direct access to memory allocations on a peer device, and for PCIe communication the GPU has to access memory from CPU.
	Yme and Mini Summit: Memcpy results shows huge differences in bandwidth when performing Memcpy from a GPU on the first NUMA node compared to from a GPU on the second NUMA node to WC host.	We did not have enough time to analyze why these benchmarks got these results, but our hypothesis is that the first CPU is the "main" CPU, and all communication has to go through it.
	Mini Summit is generally slightly better in performance than Yme	One possible reason for this could be that Mini Summit has one CPU for each GPU, but Yme has to share the CPU between two GPUs.

Table 13: Summarized results and discussion for the Power AC922 systems

Type	Result	Discussion
Power AC922 and DGX-2 comparison	The bandwidth for GPU to GPU via host on the same NUMA node on DGX-2 is way worse than the bandwidth for the same scenario for Power AC922.	This shows the significance of using NVLink interconnects to host instead of PCI Express.
	The GPU-GPU NVLink mode performance for the DGX-2, with NVSwitch, is around double the performance for the Power AC922, which uses NVLink between the GPUs.	The reason why the performance doubles is probably because for the DGX-2 the connection traverses a bounded set of six NVLinks, whereas for Yme it is used three NVLink bricks between the GPUs on the same NUMA node.
	Bus speed is better for Mini Summit and Yme than for the DGX-2.	This is because Mini Summit and Yme uses NVLink between GPU and CPU, but DGX-2 uses PCIe.
	DGX-2 has better performance than the other systems for NCCL MPI AllReduce and for NCCL Broadcast for any GPU size.	This is because the DGX-2 only has NVSwitches between GPUs. It is more scalable than the others.
	For the QTC algorithm, Mini Summit is the system with best performance for two GPUs, which can be surprising considering the previous results that shows the opposite (since it is two different NUMA nodes).	Our hypothesis is that it has better performance because it uses one GPU on each NUMA node, where both has its own CPU. For the other two systems the two first GPUs are chosen, which means that they are on the same NUMA node and share the same CPU.
	For the QTC algorithm, Yme and DGX-2 has almost the same performance when using four GPUs.	This leads us to believe that this benchmark uses a lot of CPU interaction instead of just GPU communication. Maybe because this is a real application kernel benchmark.
	Stencil 2D: When comparing the performance for different NUMA nodes compared to the same NUMA nodes for Yme and DGX-2, we can see that the performance when using different NUMA nodes is often slightly better.	If there is a lot of CPU communication, it makes sense that using GPUs on different NUMA nodes leads to better performance because of the anti locality principle for the DGX-2.
	Stencil 2D: The performance drops between 2 GPUs and 8 GPUs. A drop in performance is also the case for Yme, and then the performance exceeds the performance of the DGX-2 for four GPUs.	We did not have enough time to analyze why these benchmarks got these results.

Table 14: Summarized results and discussion for DGX-2 and Power AC922 comparisons

Type	Result	Discussion
Single GPU comparison	Titan RTX has much better performance for single precision than double precision	This is most likely because of the increasing deep learning interest, which leads to more focus on developing graphics cards with more support for lower precision computations.
	GeForce GTX 980 provides worse performance than all of the other cards.	Probably because it is older and have worse specifications.
	Tesla V100 SXM3 has better performance than the Tesla V100 SXM2 cards for MaxFlops.	Probably because of the model type.
	When using a matrix size that is dividable by four, all of the Tensor Cores shows a great decrease in time when the cores are enabled. When using a matrix size that is not dividable by four there is a very small time decrease when Tensor Cores are enabled for the GPUs that has Tensor Cores.	This leads us to believe that by using matrix dimensions that will be dividable by four, the Tensor Cores will be more useful. This makes sense because the Tensor Cores can only perform operations on 4x4 matrices.
	The performance for both of the Tesla V100 SXM2 cards are worse than the performance for the SXM3 card when Tensor Cores are not enabled. But when the Tensor Cores are enabled, the SXM2 cards have slightly better performance than the SXM3 card and notably better improvement percentage.	Since the SXM2 cards are better in only one of the scenarios it leads us to believe that the reason can not be anything external of the GPU. The main difference between SXM2 and SXM3 is that the SXM3 has higher clock rate, which could normally lead to better performance. There should be performed more tests to identify the reason for this result.
	Results shows that Titan RTX is good for lower precision calculations and Tesla V100 is better for higher precision calculations	This makes sense for the Titan RTX card, which is specialized for deep learning operations.

Table 15: Summarized results and discussion for the single GPUs

4.6 Benchmark suite evaluation

During this project we have experienced the strengths and the weaknesses for each benchmark suite used in this report. They differed in the quality of the documentation, being updated recently and complexity with configuring, setting up and running the suite.

The benchmark suite SHOC is a very popular benchmark suite in both stars on GitHub, as well as citations for the associated paper. However, it has not been updated for some years and both the documentation at the readme-file and the wiki-page at the GitHub repository page were outdated. It is not immediately understandable what command to run and which parameters to use. It is not the case for this benchmark suite that each benchmark application was documented well either. This was something that also was wanted. The code for the benchmark suite was not recently updated resulting in the code needed to be changed in order to run and compile for the tested systems.

We profiled the SHOC commands when benchmarking and found out that the NVIDIA Tensor Cores were not used in the benchmarks. This is due to the code not being updated to the latest GPU features for performance.

We would like to claim that the SHOC benchmark suite is still relevant since we got some interesting results from it. It could however need an update. The largest problem sizes are probably approaching being too small.

Tartan was a benchmark suite that was easy to use. The code was recently updated and was okay documented. The benchmark results should however have provided the results on a format that could be easier to use in scripts and plotting of graphs.

DeepBench was good documented both for compiling and running, which lead to having less problems with the benchmark suite itself. The benchmark should have better results for plotting graphs using scripts as well. Some of the results could also been easier to interpret. When we profiled the DeepBench commands when benchmarking we found out that the NVIDIA Tensor Cores was being used in the benchmarks. This was good and showed that the benchmark application was updated to newer GPU features. This makes the benchmark application able to compare the performance of operations used in state of the art applications for High Performance Computing and AI. It was also very useful that one could choose to enable or disable the use of Tensor Cores. It is a negative part that it is not possible to run the benchmarks on more than one GPU at a time. This is something that should have been implemented. Another issue was that the benchmark suite did contain too many different possibilities, making it hard to find the result that one wants.

The Scope benchmark suite generally needed more documentation as well, but especially for running the benchmark applications. An issue for this benchmark suite was that when trying to run the benchmark suite with more than 7 GPUs at a time, it would return out of memory error.

Commonly for all benchmark suites tested was that it would have been nice if the they had documented a good data flow. This means documentation about where the data flows between the CPU and GPU or GPU and GPU. Documentation such as that would have strengthen evidence about the benchmark suites performance for the tested systems. An issue that could have been fixed with multiple of the benchmark suites is that the benchmark suite should be able to detect its own architecture without providing it with a parameter.

5 Conclusion and Future Work

This section will summarize the results and contain a conclusion for each system and GPU. Since a huge part of the project timeline was used to troubleshoot and fix issues, this led to us getting the results quite late in the process, and therefore getting a short period of time for analyzing results. After the conclusion for the systems, there will be a part with future work.

The results showed that when a GPU on the NVIDIA DGX-2 communicates with another GPU, the NVSwitch interconnect is obviously more efficient than going via the CPU and the PCI Express. There are no significant differences when communicating through NVSwitches when it comes to which specific GPUs to use. Some interesting results from the PCIe interconnect evaluation is the anti locality effect, which is when GPUs that are close shows worse performance when communication compared to GPUs that are further away. The results also showed that there were slightly worse performance on the second baseboard than the first baseboard in some scenarios. Our hypothesis is that the first CPU is the "main" CPU. A result that needs further work to decipher is the one where every other CPU neighbour pair shows worse performance than the others when using PCIe connections. For when the CPU is transferring data to the GPU, there are some performance differences in for specific GPUs. This needs further work and data to figure out the reason.

The conclusion is that when using the DGX-2 for multiple GPUs, neighbours should not be chosen if there is need for much CPU-GPU communication because of the anti locality effect.

One results for Yme (IBM Power System AC922, 4 GPUs) the performance between two GPUs on the first NUMA node compared to the two GPUs on the second NUMA node is way better. We have a hypothesis that the first CPU is the "main" CPU. Since the GPUs on each NUMA node has a three bricked NVLink between them, the communication between these GPUs will be faster than going via two CPUs, even though there is NVLink connection from the GPU to the CPU too.

A GPU-CPU result for both of the Power AC922 systems is that the GPUs on the systems' first NUMA node have much better performance compared to the second NUMA node. Through almost all of the tests that compared Mini Summit and Yme, Mini Summit has had the best results. This might be because it has one CPU for each GPU.

It is a good idea to use Mini Summit if there is only need for one GPU, and Yme if there is a need for two or more. If two GPUs are needed, it can be beneficial to use only one NUMA node (the first), depending on the operations.

When comparing the Power AC922 systems to the DGX-2 we got some interesting results. The DGX-2 is better for communication from GPU to GPU, but the Power AC922 systems are better if you only need one GPU and a lot of GPU-CPU communications. The results showed that there were different scenarios where the systems were best, but if a multi-GPU application is to be run, there could be an advantage to use the DGX-2. The DGX-2 is also more scalable.

For a real application kernel benchmark (Stencil 2D) there were a drop in performance for DGX-2 and Yme. We need more data to interpret this result.

For the single GPU comparison GeForce GTX 980 had the worst performance. This is not surprising considering the specifications compared to the other GPUs. The Titan RTX card were best on operations with lower precisions, and the Tesla V100 cards were better if higher precisions were needed. All of the cards with possibility for Tensor Cores showed performance improvement when it was enabled. The Tesla V100-SXM2 cards were the ones with the highest improvement percentage when using Tensor Cores.

A result we did not have an explanation for is that the Tesla V100-SXM2 cards had worse performance (with exception of GTX 980) before Tensor Cores were enabled, and best performance

after they were enabled.

Since the Titan RTX had best performance for lower precision operations, we can assume that this is because the cards should appeal to deep learning applications, which does not necessarily need high precision. For HPC applications the Tesla V100 might be more suited because it has better performance for higher precision operations.

Comparing the Titan RTX card to the Tesla V100 card is not ideal. We should have compared a Volta and a Turing card from the same series to make the analysis fairer.

Another not fair comparison was when we compared the GeForce GTX 980 to the other graphics cards. This card is older and cheaper than the other cards, it is also a consumer card. Tesla V100 is generally a professional graphics card. The cards are made for different purposes and are therefore hard to compare in a fair way.

When comparing the Tesla V100 cards, we assumed that the memory difference would lead to some performance differences. This was not really the case, since Tesla V100-SXM2 with 32 GB memory (from Yme) was almost never better than Tesla V100-SXM2 with 16 GB memory (from Mini Summit). On the contrary, the card with 16 GB memory were most often better than the card with 32 GB. For future work we would like to find more benchmarks that pushes the memory limits. The other main difference between the Tesla V100 cards is the clock frequency. This is also something that could be investigated further in the future.

Next we will discuss additional future work. After receiving the results for our project, it opened doors to much more possibilities to explore even further. This included other benchmark suites, and even the new system "Selbu", that arrived recently at NTNU with 20 NVIDIA Tesla T4 GPUs. The machine could be compared to other GPUs with Turing architecture, such as the RTX Titan. It could also be compared to the Tesla V100 to get a fairer analysis of the Tensor Core improvements.

For future work we would like to run the benchmarks for the two Power AC922 systems inside Docker containers to minimize the potential overhead and making the benchmarking process more similar. An alternative to this is to do extensive testing on Docker overhead.

We would also like to explore the RT Cores on Tesla RTX, and how these can be used for HPC applications. For all of the results we have already collected, we would want to analyze them more and find reasons for deviation behaviour. To inspecting more of the code to see why results are the way they are would be beneficial. Other interesting things to explore is how to use new hardware, that not necessarily is made for HPC, for HPC-applications.

We would like to especially explore more of the Titan RTX card to see what other application it could perform well on. More evaluation of multi-GPU performance, especially with real life applications would be interesting to do.

In the future we would like to see how the performance could have been on Mini Summit if all 6 NVLink connections were used between the GPU and CPU, instead of only the 3 that were used now.

We would also like to make our own benchmark suite specialized for HPC applications with all of the interesting functionality. Then we could also explore how to get better performance and precision with Tensor Cores.

Two benchmark suites that was explored in this report, but was unsuccessfully tested due to challenges with compiling and running, was the OpenDwarfs and HPs DLBS benchmark suites. These two were both interesting from a HPC perspective because of their interesting benchmark tests. Other benchmark suites that looked promising and relevant for this work were xtensor-benchmark, Mirovia, PPT-GPU, benchfriend and hpcgarage/spatter.

In conclusion, there are a lot of work that could be done in the future related to this report.

References

[1] I. M. Liseter and A. C. Elster, "Grafikkprosessor." https://snl.no/grafikkprosessor, September 2019. [Accessed Dec. 10, 2019].

[2] OmniSci, "CPU vs GPU — Definition FAQs." https://www.omnisci.com/learn/resources/technical-glossary/CPU-to-GPU, December 2019. [Accessed Dec. 10, 2019].

[3] TechDifferences, "Difference Between UMA and NUMA." https://techdifferences.com/difference-between-uma-and-numa.html, October 2018. [Accessed Dec. 10, 2019].

[4] D. Inc., "What is a Container? — App Containerization — Docker." https://www.docker.com/resources/what-container, December 2019. [Accessed Dec. 15, 2019].

[5] N. Corporation, "NVIDIA/nvidia-docker: Build and run Docker containers leveraging NVIDIA GPUs." https://github.com/NVIDIA/nvidia-docker, December 2019. [Accessed Dec. 15, 2019].

[6] M. Forum, "MPI Forum." https://www.mpi-forum.org/, 2019. [Accessed Dec. 12, 2019].

[7] T. O. M. Project, "Open MPI: Open Source High Performance Computing." https://www.open-mpi.org/, 2019. [Accessed Dec. 12, 2019].

[8] NVIDIA Corporation, "IBM Spectrum MPI — NVIDIA Developer." https://developer.nvidia.com/ibm-spectrum-mpi, 2019. [Accessed Dec. 5, 2019].

[9] MPICH, "MPICH — High-Performance Portable MPI." https://www.mpich.org/, 2019. [Accessed Dec. 12, 2019].

[10] NVIDIA, "NVIDIA Collective Communication Library (NCCL) — NVIDIA Developer." https://developer.nvidia.com/nccl, November 2019. [Accessed Dec. 12, 2019].

[11] NVIDIA, "NCCL and MPI - NCCL 2.5.6 documentation." https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/mpi.html, December 2019. [Accessed Dec. 16, 2019].

[12] C. Ramseyer, "PCI Express 4.0 Brings 16 GT/s And At Least 300 Watts At The Slot." https://www.tomshardware.com/news/pcie-4.0-power-speed-express,32525.html, August 2016. [Accessed Dec. 10, 2019].

[13] HowStuffWorks, "How PCI Express Works — HowStuffWorks." https://computer.howstuffworks.com/pci-express.htm, 2019. [Accessed Dec. 16, 2019].

[14] N. Corporation, "NVLink High-Speed GPU Interconnect — NVIDIA Quadro." https://www.nvidia.com/en-us/design-visualization/nvlink-bridges, 2019. [Accessed Dec. 10, 2019].

[15] T. N. Z. S. A. S. U. Muhammad Nufail Farooqi, "Asynchronous AMR on Multi-GPUs." https://link.springer.com/chapter/10.1007/978-3-030-34356-9_11, 2019. [Accessed Dec. 15, 2019].

[16] NVIDIA Corporation, "NVIDIA NVSwitch: The World's Highest-Bandwidth On-Node Switch." https://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf, May 2018. [Accessed Nov. 7, 2019].

[17] G. Dearth, V. Venkataraman (NVIDIA), "S8688: INSIDE DGX-2." http://on-demand.gputechconf.com/gtc/2018/presentation/s8688-extending-the-connectivity-and-reach-of-the-gpu.pdf, March 2018. [Accessed Nov. 12, 2019].

[18] TechPowerUp, "NVIDIA GeForce GTX 980." https://www.techpowerup.com/gpu-specs/geforce-gtx-980.c2621. [Accessed Dec. 11, 2019].

[19] N. Corporation, "Maxwell Architecture — NVIDIA Developer." https://developer.nvidia.com/maxwell-compute-architecture. [Accessed Dec. 16, 2019].

[20] R. Nohria, G. Santos (IBM Corporation), "IBM Power System AC922: Technical Overview and Introduction." https://www.redbooks.ibm.com/redpapers/pdfs/redp5494.pdf, July 2018. [Accessed Nov. 5, 2019].

[21] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA Tensor Core Programmability, Performance & Precision." https://arxiv.org/pdf/1803.04014.pdf, March 2018. [Accessed Nov. 5, 2019].

[22] NVIDIA Corporation, "TITAN RTX Ultimate PC Graphics Card with Turing — NVIDIA." https://www.nvidia.com/en-us/deep-learning-ai/products/titan-rtx/, 2019. [Accessed Dec. 10, 2019].

[23] NVIDIA Corporation, "NVIDIA TURING GPU ARCHITECTURE." https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, 2018. [Accessed Dec. 20, 2019].

[24] IBM Corporation, "IBM Power System AC922." https://www.ibm.com/downloads/cas/6PRDKRJ0, 2019. [Accessed Nov. 5, 2019].

[25] IBM Corporation, "IBM Power System AC922 - Details." https://www.ibm.com/us-en/marketplace/power-systems-ac922/details, 2018. [Accessed Nov. 5, 2019].

[26] NVIDIA Corporation, "NVIDIA DGX-2 Datasheet." https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-2-datasheet-us-nvidia-955420-r2-web-new.pdf, July 2019. [Accessed Nov. 5, 2019].

[27] A. Ishii, D. Foley, E. Anderson, B. Dally, G. Dearth, L. Dennison, M. Hummel, and J. Schafer, "NVSWITCH AND DGX-2: NVLINK-SWITCHING CHIP AND SCALE-UP COMPUTE SERVER." https://www.hotchips.org/hc30/2conf/2.01_Nvidia_NVswitch_HotChips2018_DGX2NVS_Final.pdf, 2018. [Accessed Nov. 11, 2019].

[28] A. Danalis, G. M. C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "GitHub - The SHOC Benchmark Suite." https://github.com/vetter/shoc, 2014. [Accessed Nov. 12, 2019].

[29] A. Danalis, G. M. C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite." https://www.researchgate.net/publication/220938804_The_Scalable_HeterOgeneous_Computing_SHOC_benchmark_suite, 2010. [Accessed Nov. 12, 2019].

[30] Baidu Research, "DeepBench: Benchmarking Deep Learning operations on different hardware." https://github.com/baidu-research/DeepBench, May 2018. [Accessed Dec. 12, 2019].

[31] Google, "TensorFlow." https://www.tensorflow.org/, November 2015. [Accessed Dec. 12, 2019].

[32] Torch, "Torch — Scientific computing for LuaJIT.." http://torch.ch/, October 2002. [Accessed Dec. 12, 2019].

[33] uuudown, "Tartan." https://github.com/uuudown/Tartan, September 2018. [Accessed Dec. 13, 2019].

[34] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. Tallent, and K. Barker, "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect." https://arxiv.org/pdf/1903.04611.pdf, March 2019. [Accessed Nov. 1, 2019].

[35] NVIDIA, "CUDA Code Samples." https://developer.nvidia.com/cuda-code-samples, 2019. [Accessed Dec. 13, 2019].

[36] NVIDIA, "CUDA Runtime API :: CUDA Toolkit Documentation." https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PEER.html, 2019. [Accessed Dec. 13, 2019].

[37] NVIDIA, "NVLINK - NVIDIA Developer Forums." https://devtalk.nvidia.com/default/topic/1033056/nvlink/, 2019. [Accessed Dec. 13, 2019].

[38] c3sr, "Scope." https://github.com/c3sr/scope, July 2019. [Accessed Dec. 13, 2019].

[39] C. Pearson and S. Hashash, "Comm|Scope." https://github.com/c3sr/comm_scope, April 2019. [Accessed Dec. 13, 2019].

[40] C. Pearson and S. Hashash, "NCCL|Scope." https://github.com/c3sr/comm_scope, February 2019. [Accessed Dec. 13, 2019].

[41] Y. Ren, S. Yoo, and A. Hoisie, "Performance Analysis of Deep Learning Workloads on Leading-edge Systems." https://arxiv.org/pdf/1905.08764.pdf, October 2019. [Accessed Nov. 1, 2019].

[42] W. Feng, "OpenCL and the 13 Dwarfs." http://developer.amd.com/wordpress/media/2013/06/2155_final.pdf, June 2011. [Accessed Dec. 2, 2019].

[43] C. Pearson, A. Dakkak, S. Hashash, C. Li, I.-H. Chung, J. Xiong, and W.-M. Hwu, "Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects." https://dl.acm.org/doi/10.1145/3297663.3310299, April 2019. [Accessed Nov. 12, 2019].

[44] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers." http://www.netlib.org/utk/people/JackDongarra/PAPERS/haidar_fp16_sc18.pdf, November 2018. [Accessed Nov. 11, 2019].

[45] J. L. Salmon and S. M. Smith, "Exploiting Hardware-Accelerated Ray Tracing for Monte Carlo Particle Transport with OpenMC." https://sc19.supercomputing.org/proceedings/workshops/workshop_files/ws_pmbsf102s2-file1.pdf, October 2019. [Accessed Dec. 1, 2019].

[46] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. Tallent, and K. Barker, "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect." https://arxiv.org/pdf/1903.04611.pdf, March 2019. [Accessed Nov. 18, 2019].

[47] M. B. Sullivan, M. O'Connor, M. Erez, J. Pool, D. Nellans, and S. W. Keckler, "Buddy Compression: Enabling Larger Memory for Deep Learning and HPC Workloads on GPUs." https://arxiv.org/pdf/1903.02596.pdf, April 2019. [Accessed Dec. 1, 2019].

[48] S. W. D. Chien, I. B. Peng, and S. Markidis, "Performance Evaluation of Advanced Features in CUDA Unified Memory." https://arxiv.org/pdf/1910.09598.pdf, October 2019. [Accessed Dec. 1, 2019].

[49] Inspur Systems, "AGX-5 - Inspur Systems." https://www.inspursystems.com/product/agx-5/, 2018. [Accessed Dec. 15, 2019].

[50] P. Alcorn, "Inside The World's Largest GPU: Nvidia Details NVSwitch." https://www.tomshardware.com/news/nvidia-dgx-2-worlds-largest-gpu-nvswitch,37661.html, August 2018. [Accessed Dec. 20, 2019].

[51] NVIDIA, "nvidia-docker - Frequently Asked Questions." https://github.com/NVIDIA/nvidia-docker/wiki/Frequently-Asked-Questions, July 2019. [Accessed Nov. 19, 2019].

[52] D. C. Price, M. A. Clark, B. R. Barsdell, R. Babich, and L. J. Greenhill, "Optimizing performance-per-watt on GPUs in High Performance Computing." https://arxiv.org/pdf/1407.8116.pdf, October 2015. [Accessed Dec. 1, 2019].

[53] M. Carilli, C. Sarofeen, M. Ruberry, and B. Barsdell, "Training Neural Networks with Mixed Precision." https://on-demand.gputechconf.com/gtc-taiwan/2018/pdf/5-1_Internal%20Speaker_Michael%20Carilli_PDF%20For%20Sharing.pdf, May 2018. [Accessed Dec. 13, 2019].

[54] cpp pm, "Hunter: CMake driven cross-platform package manager for C/C++. ." https://github.com/cpp-pm/hunter, December 2019. [Accessed Dec. 16, 2019].

[55] curl, "curl." https://curl.haxx.se/, 2019. [Accessed Dec. 16, 2019].

[56] E. Smistad, T. L. Falch, M. Bozorgi, A. C. Elster, and F. Lindseth, "Medical image segmentation on GPUs – A comprehensive review." https://www.sciencedirect.com/science/article/pii/S1361841514001819, October 2014. [Accessed Dec. 2, 2019].

[57] S. Li, T. Hoefler, and M. Snir, "NUMA-Aware Shared-Memory Collective Communication for MPI." https://htor.inf.ethz.ch/publications/img/li-hoefler-snir-hpdc13-numa_collectives.pdf, June 2013. [Accessed Dec. 3, 2019].

[58] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High Performance Discrete Fourier Transforms on Graphics Processors." https://dl.acm.org/citation.cfm?id=1413373, November 2008. [Accessed Dec. 3, 2019].

[59] B. R. de Supinski, T. R. W. Scogland, A. Duran, M. Klemm, S. M. Bellido, S. L. Olivier, C. Terboven, and T. G. Mattson, "The Ongoing Evolution of OpenMP." https://ieeexplore.ieee.org/document/8434208, November 2018. [Accessed Dec. 3, 2019].

[60] K. Ahmed, J. Liu, S. Eidenbenz, and J. Zerr, "Scalable Interconnection Network Models for Rapid Performance Prediction of HPC Applications." https://ieeexplore.ieee.org/document/7828492, December 2016. [Accessed Dec. 1, 2019].

[61] J. Dongarra, M. A. Heroux, and P. Luszczek, "HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems." http://www.netlib.org/utk/people/JackDongarra/PAPERS/HPCG-benchmark.pdf, January 2015. [Accessed Dec. 3, 2019].

[62] J. C. Meyer and A. C. Elster, "Performance Modeling of Heterogeneous Systems." https://folk.idi.ntnu.no/elster/pubs/meyer-elster-ipdps2010.pdf, May 2010. [Accessed Dec. 3, 2019].

A Annotated Bibliography

Li et al. "Evaluating Modern GPU Interconnect: PCIe,NVLink, NV-SLI, NVSwitch and GPUDirect" [46]

This paper inspiring us to use Tartan, and to display the results in heat maps for nicely displaying the results. It also provided us with valuable insight when it came to anti locality.

S. Markidis et al. "NVIDIA Tensor Core Programmability, Performance & Precision" [21]

This paper was very helpful for figuring out how Tensor Cores can benefit HPC applications, they thoroughly explained how the experiments were done and we got a good idea of the downsides and upsides of using these Tensor Cores.

A. Danalis et al. "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite. [29]

This paper was useful for our project to learn how the SHOC benchmarking suite worked and to get a documentation of how to use the benchmarks. SHOC is not new, but can still provide relevant benchmark results for tested systems.

Y. Ren et al. "Performance Analysis of Deep Learning Workloads on Leading-edge Systems." [41]

This paper provided us with useful knowledge about the DGX-2 and IBM Power System AC922. It was also interesting to see which results they got.

Carl Pearson et al. "Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects" [43]

This paper presented the Scope|COMM benchmark and included very useful for benchmarking interconnects.

D. C. Price et al. "Optimizing performance-per-watt on GPUs in High Performance Computing." [52]

This paper gave us insight in how temperature can be vital to performance. It was also interesting to read about the optimizing.

M. Carilli et al. "Training Neural Networks with Mixed Precision." [53]

This presentation helped us learn how to profile benchmark applications and see if an application benchmarked was running using Tensor Cores. It showed that the number 884 contained in the kernels in the profiling meant that the application used Tensor Cores.

W. Feng, presentation about "OpenCL and the 13 Dwarfs" [42]

The Berkeley View of parallel computing is about designing hardware for future applications, instead of making the applications adjust to existing hardware and models. It is found 13 "dwarfs" that represent computational patterns that is relevant now and probably in the future. Those dwarfs can be kept in mind for developing new hardware.

This presentation is related to the topic because new GPU hardware features are often made with specific computational patterns and methods in mind. An example of this is the NVIDIA Tensor Cores. The Tensor Cores are specifically designed for matrix-multiply-and-accumulate operations and will therefore be related to the "Sparse Linear Algebra" and "Dense Linear Algebra" dwarfs. Another example is the NVIDIA Ray Tracing Cores, they can be used for calculation with the Monte Carlo method. This is related to the "MapReduce" dwarf, which the Monte Carlo method is a part of.

The presentation also contains information about a benchmark suite called "OpenCL and the 13 Dwarfs" which includes benchmarks for the dwarfs. This can be used to evaluate hardware like Tensor Cores and RT Cores.

E. Smistad et al. "Medical image segmentation on GPUs – A comprehensive review" [56]

This paper is about evaluating medical image segmentation methods on GPUs. Image segmentation is a method to divide an image into segments (image objects). Segmentation of medical images consists of segmenting CT, MRI and ultrasound images. This will transform the images to something that will be easier to analyse.

Different segmentation algorithms are evaluated for GPUs and several of them are parallelisable. This means that they also can run on larger systems with multiple GPUs and they are related to HPC.

A limiting factor of performance discussed in the paper is memory usage. Since this paper was written in 2014, there are multiple new improvement methods for GPU memory. Two techniques for this are described in the literature review and are Buddy Compression and "memory advises" and "prefetch" from CUDA Unified Memory. If techniques like these are considered, the memory usage might not be such a critical issue anymore.

S. Li et al. "NUMA-Aware Shared-Memory Collective Communication for MPI" [57]

This paper describes models for MPI Collective communication (described in literature review) for NUMA node clusters, and how to optimize them. To improve the communication, multi-threading was used, and experimental results showed that it provided a significant performance improvement.

Improving MPI communication for NUMA nodes is very relevant to HPC applications. Multiple NUMA nodes can collaborate to get computational results faster.

This is partly related to GPUs and GPU performance because NUMA nodes can have GPUs. A better collective communication leads to better overall performance, and if the computations that needs to be done needs GPUs, this communication performance will also be boosted. Some of the techniques about MPI Collective communication for NUMA node clusters could also possibly be used for Collective communication between GPUs.

N. K. Govindaraju et al. "High Performance Discrete Fourier Transforms on Graphics Processors" [58]

This paper presents algorithms to compute Discrete Fourier Transforms (DFT) on GPUs. The algorithms are from a class called Fast Fourier Transform (FFT). FFT algorithms are often used to compute HPC problems. The result from the paper is that their implementation on GPU had much better performance than existing FFT algorithms on CPU and GPU. This paper is from 2008, which is quite old in the computer science domain. There could be better implementations now.

To improve the performance of the FFT algorithms, new GPU techniques and features could be utilized. NVIDIA Tensor Core could be used to accelerate FFT, and other new hardware features could also possibly be used. It was mentioned that a bottleneck for the untried scenario of performing the computation on multiple GPUs would be the interconnects. Since 2008 there has been a great progress and improvement in the GPU interconnect area, with interconnects like NVLink and NVSwitch. Utilizing newer interconnects could help lessen the bottleneck.

Like mentioned earlier, memory optimizations have advanced since this paper was written, and could also possible boost the performance of FFT on GPUs.

B. R. de Supinski et al. "The Ongoing Evolution of OpenMP" [59]

This paper examines the past, present and future of the multi-platform shared memory multiprocessing API OpenMP. Since the topic of this paper is "New GPU features for HPC applications", the main focus will be on the recent extensions of OpenMP.

Some recent additions are directives to indicate that something should be executed with SIMD (Single Instruction, Multiple Data) parallelism, "target" directive to address hardware devices like GPUs, and a "depend" clause to set dependencies between tasks. By using the "depend" clause, the programmer can make sure that the tasks happen in the right order. Another addition is cancellation. This acts like a break statement in loops that is run in parallel, which, among other things, makes error handling more efficient. These recent additions show that OpenMP focuses more on parallelism and GPUs.

There are also plans for future extensions which partly focuses on memory allocations. This paper is related to the topic because OpenMP is an important part of parallelism inside a node, and thus an important part of HPC applications. OpenMP has support for GPUs, which means that when OpenMP advances, the way OpenMP utilizes the GPU will improve, which could again lead to better performance for HPC applications.

K. Ahmed et al. "Scalable Interconnection Network Models for Rapid Performance Prediction of HPC Applications" [60]

Performance Prediction Toolkit (PPT) is a simulator used to predict performance on existing and future HPC architectures. The simulator is based on the topology of systems. There are not any models capable of simulating an entire system in detail. A reason for this is that when there are uncertainties in the simulation, it can lead to huge modelling errors.

PPT differs from other similar simulators because it can among other things integrate large applications with full-scale architecture models, and it can scale and achieve high performance using parallelization. The results show that the PPT interconnect model is a pretty accurate model. By using PPT for HPC applications, performance can be predicted and if the model is accurate enough, there is no need in testing the applications on a real system.

Modelling and simulation toolkits generally can be great for figuring out which hardware that could work for different applications. This could be important for developing new hardware or architectures for GPUs to improve HPC and AI applications.

J. Dongarra et al. "High Performance Conjugate Gradient Benchmark: a New Metric for Ranking High Performance Computing Systems" [61]

The High Performance Conjugate Gradient (HPCG) benchmark is focused on evaluating machines ability to run scientific applications, which can be HPC applications. The benchmark tests common computations from scientific applications, and the systems are ranked based on a simple additive Schwarz, symmetric Gauss-Seidel preconditioned conjugate gradient solver.

HPCG benchmark has result lists that ranks the best systems in the world. The lists are updated twice a year and are great for checking which systems that will be good for running big HPC applications. This ranking is more based on HPC applications than AI applications, which is important when there is currently so much focus on AI systems.

The HPCG benchmark is also a great way for hardware companies and makers of computer systems to see what types of technology that ranks the highest and what specific features and architecture they have. This way they can get tips on how to improve future computer systems by looking at parts that improve current systems.

J. C. Meyer et al. "Performance Modeling of Heterogeneous Systems" [62]

This paper describes a model for predicting performance of heterogeneous systems. The model uses BSPlib, which is a library with implementation of the Bulk-Synchronous Parallelism (BSP) model. MPI and GASnet are used for communication. The model gets characteristics from applications and architectures and produces results for the expected performance.

Models for predicting performance on heterogeneous systems are important for HPC applications, which often runs on heterogeneous systems. GPUs are also currently a big part of heterogeneous systems. By predicting performance for different architectures with GPUs, techniques that needs improvement and techniques that works well can be detected. This way changes can be made to create optimal systems for different programs.

B System Information

B.1 GTX 980 based system

```
knutakir@hpclab12:~$ nvidia-smi topo -m
      GPU0  CPU Affinity
GPU0    X     0-7

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA
    nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host
    Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
    CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe
    Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```

Listing 1: Topology for GTX 980 system

```
knutakir@hpclab12:~$ nvidia-smi nvlink --status -i 0
```

Listing 2: NVLink status for GTX 980 system. No results because the GPU does not have the possibility for NVLink connections

```
knutakir@hpclab12:~$ nvidia-smi
Wed Dec 18 19:50:42 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 980     On   | 00000000:01:00.0 Off |                  N/A |
| 27%   29C    P8    13W / 180W |     17MiB /  4041MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Listing 3: Information about the GTX 980 GPU when running the nvidia-smi command

B.2 Titan RTX based system

```
ingunsu@hpclab15:~$ nvidia-smi topo -m
  GPU0  CPU Affinity
GPU0    X  0-15

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA
    nodes (e.g., QPI/UPI)
```

```
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host
    Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
    CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe
    Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```

Listing 4: Topology for Titan RTX system

```
ingunsu@hpclab15:~$ nvidia-smi nvlink --status -i 0
GPU 0: TITAN RTX (UUID: GPU-8583ed85-a5e2-eeb3-178a-5921ab72dcf3)
   Link 0: <inactive>
   Link 1: <inactive>
```

Listing 5: NVLink status for Titan RTX system. The system has a possibility for two NVLink connections, but this is not used on this particular computer.

```
ingunsu@hpclab15:~$ nvidia-smi
Mon Dec 16 23:52:04 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  TITAN RTX           On   | 00000000:01:00.0 Off |                  N/A |
| 41%   37C    P8    15W / 280W |    172MiB / 24217MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Listing 6: Information about the Titan RTX GPU when running the nvidia-smi command

B.3 IBM Power System AC922

B.3.1 Yme (4 GPUs)

```
-bash-4.2$ nvidia-smi topo -m
        GPU0  GPU1  GPU2  GPU3  CPU Affinity
GPU0     X    NV3   SYS   SYS    0-63
GPU1    NV3    X    SYS   SYS    0-63
GPU2    SYS   SYS    X    NV3    64-127
GPU3    SYS   SYS   NV3    X     64-127

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA
    nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host
    Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
    CPU)
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe
     Host Bridge)
  PIX  = Connection traversing a single PCIe switch
```

```
  NV#  = Connection traversing a bonded set of # NVLinks

-bash-4.2$ nvidia-smi topo -mp
        GPU0  GPU1  GPU2  GPU3  CPU Affinity
GPU0     X    PIX   SYS   SYS 0-63
GPU1    PIX    X    SYS   SYS 0-63
GPU2    SYS   SYS    X    PIX 64-127
GPU3    SYS   SYS   PIX    X  64-127
```

Listing 7: Topology for Yme (Power AC922, 4 GPUs). First matrix is the direct communication matrix, the second is PCI only.

```
-bash-4.2$ nvidia-smi nvlink --status -i 0
GPU 0: Tesla V100-SXM2-32GB (UUID: GPU-8be05466-6c9b-4d91-a8c8-4ed680df64a5)
   Link 0: 25.781 GB/s
   Link 1: 25.781 GB/s
   Link 2: 25.781 GB/s
   Link 3: 25.781 GB/s
   Link 4: 25.781 GB/s
   Link 5: 25.781 GB/s

...

-bash-4.2$ nvidia-smi nvlink --status -i 3
GPU 3: Tesla V100-SXM2-32GB (UUID: GPU-ef24cc4c-5f06-bbb6-6dac-fe62e170aa75)
   Link 0: 25.781 GB/s
   Link 1: 25.781 GB/s
   Link 2: 25.781 GB/s
   Link 3: 25.781 GB/s
   Link 4: 25.781 GB/s
   Link 5: 25.781 GB/s
```

Listing 8: NVLink status on Yme (Power AC922, 4 GPUs). The listing shows only GPU 0 and 3 because the command will print the same for every GPU in the system.

```
-bash-4.2$ nvidia-smi
Sun Dec 15 16:45:04 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000004:04:00.0 Off |                    0 |
| N/A   38C    P0    39W / 300W |      0MiB / 32510MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Listing 9: Information about the first GPU when running the nvidia-smi command on Yme (Power AC922, 4 GPUs).

B.3.2 Mini Summit (2 GPUs)

```
-bash-4.2$ nvidia-smi topo -m
        GPU0  GPU1  CPU Affinity
GPU0     X    SYS    0-63
GPU1    SYS    X     64-127
```

```
Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA
    nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host
    Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
    CPU)
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe
     Host Bridge)
  PIX  = Connection traversing a single PCIe switch
  NV#  = Connection traversing a bonded set of # NVLinks
```

Listing 10: Topology for Mini Summit (Power AC922, 2 GPUs)

```
-bash-4.2$ nvidia-smi nvlink --status -i 0
GPU 0: Tesla V100-SXM2-16GB (UUID: GPU-d5cfaea6-0aca-7f9b-5ed1-957950b4f8f8)
   Link 0: <inactive>
   Link 1: 25.781 GB/s
   Link 2: <inactive>
   Link 3: 25.781 GB/s
   Link 4: <inactive>
   Link 5: 25.781 GB/s

-bash-4.2$ nvidia-smi nvlink --status -i 1
GPU 1: Tesla V100-SXM2-16GB (UUID: GPU-4822d295-9751-d6b8-bd93-7739510f189e)
   Link 0: <inactive>
   Link 1: 25.781 GB/s
   Link 2: <inactive>
   Link 3: 25.781 GB/s
   Link 4: <inactive>
   Link 5: 25.781 GB/s
```

Listing 11: NVLink status on Mini Summit (Power AC922, 2 GPUs)

```
-bash-4.2$ nvidia-smi
Sun Dec 15 16:49:09 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000004:04:00.0 Off |                    0 |
| N/A   37C    P0    35W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Listing 12: Information about the first GPU when running the nvidia-smi command on Mini Summit (Power AC922, 2 GPUs).

B.4 DGX-2

```
ingunsu@DGX-2:~$ nvidia-smi topo -m
      GPU0 G1   G2   G3   G4   G5   G6   G7   G8   G9   G10 G11 G12 G13 G14 G15   CPU Affinity
```

```
GPU0    X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU1   NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU2   NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU3   NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU4   NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU5   NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU6   NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU7   NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6    0-23,48-71
GPU8   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6 NV6    24-47,72-95
GPU9   NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6 NV6    24-47,72-95
GPU10 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6 NV6    24-47,72-95
GPU11 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6 NV6    24-47,72-95
GPU12 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6 NV6    24-47,72-95
GPU13 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6 NV6    24-47,72-95
GPU14 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X   NV6    24-47,72-95
GPU15 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6 NV6  X     24-47,72-95

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA
    nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host
    Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU
    )
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe
    Host Bridge)
  PIX  = Connection traversing a single PCIe switch
  NV#  = Connection traversing a bonded set of # NVLinks

ingunsu@DGX-2:~$ nvidia-smi topo -mp
     G0    G1    G2    G3    G4    G5    G6    G7    G8    G9    G10   G11   G12   G13   G14   G15
G0    X    PIX   PXB   PXB   NODE NODE NODE NODE SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G1   PIX    X    PXB   PXB   NODE NODE NODE NODE SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G2   PXB   PXB    X    PIX   NODE NODE NODE NODE SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G3   PXB   PXB   PIX    X    NODE NODE NODE NODE SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G4   NODE NODE NODE NODE  X    PIX   PXB   PXB   SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G5   NODE NODE NODE NODE PIX    X    PXB   PXB   SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G6   NODE NODE NODE NODE PXB   PXB    X    PIX   SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G7   NODE NODE NODE NODE PXB   PXB   PIX    X    SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS
G8   SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS    X    PIX   PXB   PXB   NODE NODE NODE NODE
G9   SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   PIX    X    PXB   PXB   NODE NODE NODE NODE
G10 SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   PXB   PXB    X    PIX   NODE NODE NODE NODE
G11 SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   PXB   PXB   PIX    X    NODE NODE NODE NODE
G12 SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   NODE NODE NODE NODE  X    PIX   PXB   PXB
G13 SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   NODE NODE NODE NODE PIX    X    PXB   PXB
G14 SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   NODE NODE NODE NODE PXB   PXB    X    PIX
G15 SYS   SYS   SYS   SYS   SYS   SYS   SYS   SYS   NODE NODE NODE NODE PXB   PXB   PIX    X
```

Listing 13: Topology for DGX-2 (G=GPU). First matrix is the direct communication matrix, the second is PCI only.

```
ingunsu@DGX-2:~$ nvidia-smi nvlink --status -i 0
GPU 0: Tesla V100-SXM3-32GB (UUID: GPU-ad78d3a5-0a4f-ac16-0ea4-e02b88404047)
   Link 0: 25.781 GB/s
   Link 1: 25.781 GB/s
   Link 2: 25.781 GB/s
   Link 3: 25.781 GB/s
```

```
   Link 4: 25.781 GB/s
   Link 5: 25.781 GB/s

...

ingunsu@DGX-2:~$ nvidia-smi nvlink --status -i 15
GPU 15: Tesla V100-SXM3-32GB (UUID: GPU-4e5a4b58-56fc-114b-5de2-ee41540cc549)
   Link 0: 25.781 GB/s
   Link 1: 25.781 GB/s
   Link 2: 25.781 GB/s
   Link 3: 25.781 GB/s
   Link 4: 25.781 GB/s
   Link 5: 25.781 GB/s
```

Listing 14: NVLink status on DGX-2. The listing shows only GPU 0 and 15 because the command will print the same for every GPU in the system.

```
ingunsu@DGX-2:~$ nvidia-smi
Sun Dec 15 16:51:35 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM3...  On   | 00000000:34:00.0 Off |                    0 |
| N/A   33C    P0    49W / 350W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Listing 15: Information about the first GPU when running the nvidia-smi command on the DGX-2

C Setup

This appendix describes prerequisites and how to run the different benchmarks.

How the Docker images were built and ran:

How to build Docker image

```
# Build from the Dockerfile
docker build -t <image_name> .

# Or build from a specific file
docker build -f <path_to_dockerfile> -t <image_name> .
```

Listing 16: Build Docker image

How to run the Docker container

This command will remove the container after use and enable interactive terminal output making it easy for running multiple commands after each other.

```
nvidia-docker run --rm -ti <image_name>
```

Listing 17: Run Docker container

Note, the Dockerfiles provided for each benchmark suite below is for x86_64 architecture. If NVIDIA Docker had worked during this project for the tested POWER9 IBM systems, the Dockerfiles for them would not be that much different from the ones provided below for each benchmark suite. The main difference would be the `FROM`-image that it would be built from. This would be changed from the x86_64 architecture image `nvidia/cuda` to `nvidia/cuda-ppc64le`.

C.1 SHOC

To build the SHOC benchmark suite it is needed to:

- **Clone the code** from our Git repository fork: our fork contains modifications that are needed to run the benchmark suite with CUDA version 10.2 and on IBM Power AC922 systems. The fork is located on GitHub at `https://github.com/Knutakir/shoc`.
- Have a working **CUDA** toolkit: this is needed for running the benchmarks used in this report, as the used ones are written using CUDA-code.
- Specify the **target architecture** in the configure command: because the configure command does not automatically detect the given system architecture to compile the code for.
- Specify the ***--with-mpi*** flag in the configure command: if one needs to run the benchmark suite on multiple GPUs with MPI.
- Have a working implementation of **MPI**: this needs to be an implementation that is included in the system environment variable paths so that SHOC finds it.
- Have a working **Perl interpreter**: this is needed to run the script that combines all the benchmark applications and gives an output in the terminal. Note, this is not needed to run the benchmarks separately.
- Have a working **NVIDIA Docker**: if the benchmark suite is ran inside a Docker container. Example of Dockerfile is shown below in Listing 18.

- Run the `configure` script with its needed parameters and the `make` commands as on line 15 to 18 in Listing 18.

To run the SHOC bencmark suite it is needed to:

- Run the command as below on line 21 in Listing 18, or run the individual benchmarks with their respective executable files.
- Specify the ***-s*** parameter for selecting the problem size to run the benchmarks with.
- Specify the ***-cuda*** parameter for running the CUDA benchmarks.
- Specify the ***-d*** parameter for selecting which device to run on. If this is not specified, it will default to device 0.

```
1  # Same version as nvidia/cuda:latest as of 20.11.2019, just more specified
   FROM nvidia/cuda:10.1-devel-ubuntu18.04
3
   WORKDIR /usr/src/shoc
5
   # Install dependencies
7  RUN apt-get update && apt-get install -y \
       openmpi-bin \
9      openssh-client \
       libopenmpi-dev
11
   COPY . .
13
   # Configure with mpi for the target architecture and compile the sources
15 RUN ./configure CUDA_CPPFLAGS="-gencode=arch=compute_70,code=sm_70" --with-mpi
   RUN make clean
17 RUN make
   RUN make install
19
   # Example command to run benchmark on two GPUs with size 4
21 CMD perl ./tools/driver.pl -s 4 -cuda -d 14,15
```

Listing 18: SHOC Dockerfile for the DGX-2 system with Volta GPUs.

C.2 DeepBench

To build the NVIDIA benchmarks for DeepBench it is needed to:

- **Clone the code** from the authors Git repository.
- Have a working **CUDA** toolkit: this is needed as the benchmarks from this benchmark suite used in this report, are written using CUDA-code.
- Have a working **cuDNN**: this is needed as the benchmarks used in this report uses it.
- Have a working **cuBLAS**: this is needed as the benchmarks used in this report uses it.
- Have a working **NCCL**: this is needed as NCCL is used to communicate between multiple nodes in this benchmark suite.

- Have a working implementation of **MPI**: this needs to be an implementation that is included in the system environment variable paths so that DeepBench finds it, or specified using the ***MPI_INCLUDE_PATH*** and the ***MPI_PATH*** parameters as in the build command on line 19 in Listing 19. The ***nccl_mpi*** Makefile target inside the Makefile in the *nvidia*-directory needs to match the installed location of the MPI installations *lib*-directory. This is the linking part to the *MPI_PATH lib*-directory and was needed to be changed to `lib64` from `lib` for building this benchmark on the IBM POWER9 systems.
- Specify the **target architecture** in the build command. This is done by providing the ***ARCH*** parameter and is needed because the benchmark suite does not automatically detect the given system architecture to compile the code for.
- Specify to use **Tensor Cores** in the build command. This is to enable the Tensor Cores for the benchmarked operations. This is specified by the ***USE_TENSOR_CORES*** parameter.
- Have a working **NVIDIA Docker**: if the benchmark suite is ran inside a Docker container. Example of Dockerfile is shown below in Listing 19.
- Run the `make nvidia` command with the needed parameters as on line 19 in Listing 19.

To run the training and inference benchmarks for DeepBench it is needed to:

- Run either of the following commands from the *code*-directory with specified training or inference and the given precision for the benchmarks:
  - `./bin/conv_bench <train|inference> <float|half|int8>`
  - `./bin/gemm_bench <train|inference> <float|half|int8>`
  - `./bin/rnn_bench <train|inference> <float|half>`

To run the AllReduce benchmarks for DeepBench it is needed to:

- Have a working implementation of MPI, such that running using a command like `mpirun` can be used. This is needed to run the `nccl_mpi_all_reduce` benchmark as shown below.
- Run either of the following commands from the *code*-directory with respectively specified the number of ranks or the number of GPUs for the benchmarks:
  - `mpirun -np <num_ranks> ./bin/nccl_mpi_all_reduce`
  - `./bin/nccl_single_all_reduce <num_gpus>`

```
1  FROM nvidia/cuda:10.0-cudnn7-devel-ubuntu18.04
2
3  WORKDIR /usr/src/deepbench
4
5  # Install dependencies
6  RUN apt-get update && apt-get install -y \
7      openmpi-bin \
8      openssh-client \
9      libopenmpi-dev
10
11 COPY . .
12
13 # Change directory into code
```

```
WORKDIR /usr/src/deepbench/code

# Make the nvidia benchmarks without tensor cores
# Rerun with `USE_TENSOR_CORES=1` to enable them
# Change the ARCH parameter for the given system
RUN make nvidia ARCH=sm_75 USE_TENSOR_CORES=0 MPI_INCLUDE_PATH=/usr/include/mpi
    MPI_PATH=/usr/
```

Listing 19: DeepBench Dockerfile for the Titan RTX system with Turing GPU.

C.3 Tartan

To build the `scale_up_p2p_test` and `scale_up_p2p_packet` benchmarks for Tartan it is needed to:

- **Clone the code** from the authors Git repository.
- Have a working **CUDA** toolkit: this is needed for running the benchmarks used in this report, as the used ones are written using CUDA-code.
- Have a working implementation of **MPI**: this needs to be an implementation that is included in the system environment variable paths so that Tartan finds it.
- Specify the **GPU architecture** in the *shared.mk*-file in the *microbenchmark*-directory. This is the ***ARCH*** variable.
- Specify the **CUDA path** in the *shared.mk*-file in the *microbenchmark*-directory. This is the ***CUDA_DIR*** variable.
- Have a working **NVIDIA Docker**: if the benchmark suite is ran inside a Docker container. Example of Dockerfile is shown below in Listing 20.
- Run the `make clean` and `make` commands for each benchmark application in their respective directory.

To run the `scale_up_p2p_test` and `scale_up_p2p_packet` benchmarks for Tartan it is needed to:

- Run either of the following commands in its respectively directory:
  - `./scale_up_p2p_test`
  - `./scale_up_p2p_packet`

```
FROM nvidia/cuda:10.2-devel-ubuntu18.04

WORKDIR /usr/src/tartan

COPY . .

WORKDIR /usr/src/tartan/microbenchmark

# Make the `scale_up_p2p_test` benchmark
RUN cd scale_up_p2p_test && make clean && make

# Make the `scale_up_p2p_packet` benchmark
```

```
RUN cd scale_up_p2p_packet && make clean && make
```

Listing 20: Tartan Dockerfile for the DGX-2 system.

C.4 Scope

To build the `Comm|Scope` and `NCCL|Scope` benchmarks for Scope it is needed to:

- **Clone the code** from the authors Git repository.
- Have a working **CUDA** toolkit: this is needed for running the benchmarks used in this report, as the used ones are written using CUDA-code.
- Have a working **NCCL**: this is needed as NCCL is used to communicate between multiple nodes in this benchmark suite.
- Have a working implementation of **MPI**: this needs to be an implementation that is included in the system environment variable paths so that Scope finds it.
- **CMake version 3.12** or higher: this is needed to build the benchmarks. The CMake needs to be built with a cURL that has SSL enabled. This is due to this benchmark suite is dependent on Hunter a package manager used for downloading and installing the dependencies.
- The **Git submodules** needs to be updated from their sources.
- Have a working **NVIDIA Docker**: if the benchmark suite is ran inside a Docker container. Example of Dockerfile is shown below in Listing 21.
- Run the `cmake` and `make` commands as on line 23 to 30 in Listing 21.

To run the `Comm|Scope` and `NCCL|Scope` benchmarks for Scope it is needed to:

- Run the command `./scope` from the *build*-directory.
- Specify the **number of GPUs** to use for the benchmark. This is for the NCCL benchmarks and is done by using the ***--ngpu*** parameter.
- Specify the **selected GPUs** to communicate between. This is for the Comm benchmarks and is done by using the `-c=x -c=y` parameters, where `x` and `y` represent the GPU IDs.
- Specify the `--benchmark_filter` parameter for selecting benchmark(s) to run.
- Specify the `--benchmark_out_format=json` and `--benchmark_out=<file_name>.json` parameters to save the benchmark result output to a JSON file.

```
FROM nvidia/cuda:10.2-cudnn7-devel

RUN apt-get update && apt-get install -y --no-install-recommends --no-install-
    suggests \
    curl \
    git \
    && rm -rf /var/lib/apt/lists/*

RUN gcc --version
```

```
10 RUN curl -sSL https://cmake.org/files/v3.12/cmake-3.12.1-Linux-x86_64.tar.gz -o
       cmake.tar.gz \
12     && tar -xf cmake.tar.gz \
       && cp -r cmake-3.12.1-Linux-x86_64/* /usr/. \
14     && rm cmake.tar.gz

16 RUN cmake --version

18 ENV SCOPE_ROOT /opt/scope
   COPY . ${SCOPE_ROOT}
20 WORKDIR ${SCOPE_ROOT}

22 RUN mkdir -p build \
       && cd build \
24     && cmake .. -DCMAKE_BUILD_TYPE=Release \
       -DENABLE_MISC=OFF \
26     -DENABLE_NCCL=ON \
       -DENABLE_CUDNN=OFF \
28     -DENABLE_COMM=ON \
       -DENABLE_EXAMPLE=OFF \
30     -DNVCC_ARCH_FLAGS="3.0 3.2 3.5 3.7 5.0 5.2 5.3 6.0" \
       && make VERBOSE=1

32 ENV PATH ${SCOPE_ROOT}/build:$PATH
```

Listing 21: Scope Dockerfile for the DGX-2 system modified from Scope's GitHub repository [38].

D Benchmark Results

This appendix contains most of the results discussed in this report. GitHub repository with all benchmark results: `https://github.com/ingunnsund/SP19-Benchmark-results`

D.1 SHOC

Results from SHOC benchmark for size `4`. More SHOC results can be seen here: *GitHub - SHOC results*

```
# GPU 0
Running benchmark BusSpeedDownload
    result for bspeed_download:                 11.8479 GB/sec
Running benchmark BusSpeedReadback
    result for bspeed_readback:                 13.1504 GB/sec
Running benchmark MaxFlops
    result for maxspflops:                   16241.7000 GFLOPS
    result for maxdpflops:                   8140.8000 GFLOPS

# GPU 7
Running benchmark BusSpeedDownload
    result for bspeed_download:                 11.8653 GB/sec
Running benchmark BusSpeedReadback
    result for bspeed_readback:                 13.1492 GB/sec
Running benchmark MaxFlops
    result for maxspflops:                   16238.7000 GFLOPS
    result for maxdpflops:                   8146.5200 GFLOPS

# GPU 8
Running benchmark BusSpeedDownload
    result for bspeed_download:                 11.8526 GB/sec
Running benchmark BusSpeedReadback
    result for bspeed_readback:                 13.1490 GB/sec
Running benchmark MaxFlops
    result for maxspflops:                   16244.6000 GFLOPS
    result for maxdpflops:                   8149.0400 GFLOPS

# GPU 15
Running benchmark BusSpeedDownload
    result for bspeed_download:                 11.9008 GB/sec
Running benchmark BusSpeedReadback
    result for bspeed_readback:                 13.1502 GB/sec
Running benchmark MaxFlops
    result for maxspflops:                   16209.5000 GFLOPS
    result for maxdpflops:                   8149.0400 GFLOPS
```

Listing 22: Part of SHOC results for 1 GPU on DGX-2

Average values from Listing 22

BusSpeedDownload:
$(11.8479 + 11.8526 + 11.9008 + 11.8653)/4 = 11.86665$ GB/s
BusSpeedReadback:
$(13.1504 + 13.1490 + 13.1502 + 13.1492)/4 = 13.1497$ GB/s

MaxFlops SP:
$(16241.7000 + 16244.6000 + 16209.5000 + 16238.7000)/4 = 16233.625$ GFLOPS
MaxFlops DP:
$(8140.8000 + 8149.0400 + 8149.0400 + 8146.5200)/4 = 8146.35$ GFLOPS

```
# 1 GPU:
Running benchmark Stencil2D
    result for stencil:                          646.5240 GFLOPS

# 2 GPUs:
## IDs: 0, 1:
Running benchmark QTC
    result for qtc:                                9.4461 s
Running benchmark Stencil2D
    result for stencil:                         1005.6100 GFLOPS
## IDs: 0, 15:
Running benchmark Stencil2D
    result for stencil:                         1035.0200 GFLOPS

# 4 GPUs:
## IDs: 0, 1, 2, 3:
Running benchmark QTC
    result for qtc:                                5.5789 s
Running benchmark Stencil2D
    result for stencil:                          923.6450 GFLOPS
## IDs: 0, 1, 14, 15:
Running benchmark Stencil2D
    result for stencil:                          950.2330 GFLOPS

# 6 GPUs:
## IDs: 0, 1, 2, 3, 4, 5:
Running benchmark QTC
    result for qtc:                                4.1859 s
    result for qtc_kernel:                         3.2554 s
Running benchmark Stencil2D
    result for stencil:                          902.6660 GFLOPS
    result for stencil_dp:                       661.9390 GFLOPS
## IDs: 5, 6, 7, 8, 9, 10:
Running benchmark Stencil2D
    result for stencil:                          904.1280 GFLOPS
    result for stencil_dp:                       661.5450 GFLOPS

# 8 GPUs:
## IDs: 0, 1, 2, 3, 4, 5, 6, 7:
Running benchmark QTC
    result for qtc:                                3.3905 s
Running benchmark Stencil2D
    result for stencil:                         1176.9600 GFLOPS
## IDs: 4, 5, 6, 7, 8, 9, 10, 11:
Running benchmark Stencil2D
    result for stencil:                         1163.8400 GFLOPS

# 10 GPUs:
## IDs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9:
Running benchmark QTC
    result for qtc:                                2.8812 s
Running benchmark Stencil2D
    result for stencil:                         1438.6300 GFLOPS
```

```
54
# 12 GPUs:
56 ## IDs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11:
Running benchmark QTC
58     result for qtc:                                2.5148 s
Running benchmark Stencil2D
60     result for stencil:                         1635.6200 GFLOPS

62 # 14 GPUs:
## IDs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13:
64 Running benchmark QTC
    result for qtc:                                2.2556 s
66 Running benchmark Stencil2D
    result for stencil:                         1938.2200 GFLOPS
68
# 16 GPUs:
70 Running benchmark QTC
    result for qtc:                                1.9353 s
72 Running benchmark Stencil2D
    result for stencil:                         2230.6100 GFLOPS
```

Listing 23: Part of SHOC results for multi-GPU on DGX-2

```
1 # GPU 0
Running benchmark BusSpeedDownload
3     result for bspeed_download:                  70.8237 GB/sec
Running benchmark BusSpeedReadback
5     result for bspeed_readback:                  72.6437 GB/sec
Running benchmark MaxFlops
7     result for maxspflops:                    15637.8000 GFLOPS
    result for maxdpflops:                     7841.0500 GFLOPS
9
# GPU 1
11 Running benchmark BusSpeedDownload
    result for bspeed_download:                  65.8635 GB/sec
13 Running benchmark BusSpeedReadback
    result for bspeed_readback:                  71.7696 GB/sec
15 Running benchmark MaxFlops
    result for maxspflops:                    15530.8000 GFLOPS
17     result for maxdpflops:                     7822.3600 GFLOPS

19 # GPU 2
Running benchmark BusSpeedDownload
21     result for bspeed_download:                  41.6925 GB/sec
Running benchmark BusSpeedReadback
23     result for bspeed_readback:                  35.5537 GB/sec
Running benchmark MaxFlops
25     result for maxspflops:                    15564.3000 GFLOPS
    result for maxdpflops:                     7814.4500 GFLOPS
27
# GPU 3
29 Running benchmark BusSpeedDownload
    result for bspeed_download:                  41.6885 GB/sec
31 Running benchmark BusSpeedReadback
    result for bspeed_readback:                  35.1447 GB/sec
33 Running benchmark MaxFlops
    result for maxspflops:                    15556.7000 GFLOPS
35     result for maxdpflops:                     7840.5800 GFLOPS
```

Listing 24: Part of SHOC results for 1 GPU on Yme (Power AC922, 4 GPUs)

Average values from Listing 24

BusSpeedDownload:
$(70.8237 + 65.8635 + 41.6925 + 41.6885)/4 = 55.01705$ GB/s
BusSpeedReadback:
$(72.6437 + 71.7696 + 35.5537 + 35.1447)/4 = 53.777925$ GB/s
MaxFlops SP:
$(15637.8000 + 15530.8000 + 15564.3000 + 15556.7000)/4 = 15572.4$ GFLOPS
MaxFlops DP:
$(7841.0500 + 7822.3600 + 7814.4500 + 7840.5800)/4 = 7829.61$ GFLOPS

```
# 1 GPU:
Running benchmark Stencil2D
    result for stencil:                          618.6470 GFLOPS

# 2 GPUs:
## IDs: 0, 1:
Running benchmark QTC
    result for qtc:                                9.6263 s
Running benchmark Stencil2D
    result for stencil:                          960.6280 GFLOPS

## IDs: 0, 2:
Running benchmark Stencil2D
    result for stencil:                          973.8870 GFLOPS

# 3 GPUs:
## IDs: 0, 1, 2:
Running benchmark QTC
    result for qtc:                                6.9794 s
Running benchmark Stencil2D
    result for stencil:                          924.1910 GFLOPS

# 4 GPUs:
Running benchmark QTC
    result for qtc:                                5.4775 s
Running benchmark Stencil2D
    result for stencil:                         1119.1400 GFLOPS
```

Listing 25: Part of SHOC results for multi-GPU on Yme (Power AC922, 4 GPUs)

```
# GPU 0
Running benchmark BusSpeedDownload
    result for bspeed_download:                   72.5094 GB/sec
Running benchmark BusSpeedReadback
    result for bspeed_readback:                   72.6444 GB/sec
Running benchmark MaxFlops
    result for maxspflops:                     15590.2000 GFLOPS
    result for maxdpflops:                     7817.5800 GFLOPS

# GPU 1
Running benchmark BusSpeedDownload
    result for bspeed_download:                   41.6871 GB/sec
```

```
Running benchmark BusSpeedReadback
    result for bspeed_readback:                  35.5730 GB/sec
Running benchmark MaxFlops
    result for maxspflops:                     15563.9000 GFLOPS
    result for maxdpflops:                     7821.0800 GFLOPS
```

Listing 26: Part of SHOC results for 1 GPU on Mini Summit (Power AC922, 2 GPUs)

Average values from Listing 26

BusSpeedDownload:
$(72.5094 + 41.6871)/2 = 57.09825$ GB/s
BusSpeedReadback:
$(72.6444 + 35.5730)/2 = 54.1087$ GB/s
MaxFlops SP:
$(15590.2000 + 15563.9000)/2 = 15577.05$ GFLOPS
MaxFlops DP:
$(7817.5800 + 7821.0800)/2 = 7819.33$ GFLOPS

```
# 1 GPU:
Running benchmark Stencil2D
    result for stencil:                       666.7810 GFLOPS

# 2 GPUs:
Running benchmark QTC
    result for qtc:                             8.8826 s
Running benchmark Stencil2D
    result for stencil:                      1041.3000 GFLOPS
```

Listing 27: Part of SHOC results for multi-GPU on Mini Summit (Power AC922, 2 GPUs)

```
# First iteration
Running benchmark MaxFlops
    result for maxspflops:                   18201.7000 GFLOPS
    result for maxdpflops:                     550.2150 GFLOPS

# Second iteration
Running benchmark MaxFlops
    result for maxspflops:                   18201.9000 GFLOPS
    result for maxdpflops:                     550.2180 GFLOPS
```

Listing 28: Part of SHOC results for Titan RTX

Average values from Listing 28

MaxFlops SP:
$(18201.7000 + 18201.9000)/2 = 18201.8$ GFLOPS
MaxFlops DP:
$(550.2150 + 550.2180)/2 = 550.2165$ GFLOPS

```
# First iteration
Running benchmark MaxFlops
    result for maxspflops:                    4934.0400 GFLOPS
    result for maxdpflops:                     157.3030 GFLOPS

```

```
# Second iteration
Running benchmark MaxFlops
    result for maxspflops:                    4927.8000 GFLOPS
    result for maxdpflops:                     157.1560 GFLOPS
```

Listing 29: Part of SHOC results for GTX 980

Average values from Listing 29

MaxFlops SP:
$(4934.0400 + 4927.8000)/2 = 4930.92$ GFLOPS
MaxFlops DP:
$(157.3030 + 157.1560)/2 = 157.2295$ GFLOPS

D.2 Tartan

Results from Tartan benchmark `scale_up_p2p_packet` for size `33554432`. More Tartan results can be seen here: *GitHub - Tartan results*

```
===== 33554432 =====
Uni-PCIe-Bandwidth
   D\D     0      1      2      3
     0 744.36  44.93  30.50  29.89
     1  44.69 741.36  30.71  30.02
     2  29.91  30.05 741.36  25.32
     3  29.98  29.98  24.99 743.04
Uni-NVLink-Bandwidth
   D\D     0      1      2      3
     0 741.04  71.35  41.08  40.78
     1  71.31 741.36  40.61  41.01
     2  40.92  40.76 742.88  71.22
     3  40.67  40.69  71.14 740.99
Bi-PCIe-Bandwidth
   D\D     0      1      2      3
     0 779.03  57.94  44.84  45.46
     1  58.83 781.15  44.37  44.19
     2  44.31  44.89 782.11  23.38
     3  45.04  44.35  23.01 782.20
Bi-NVLink-Bandwidth
   D\D     0      1      2      3
     0 776.52 142.03  39.78  39.14
     1 142.16 782.05  39.09  38.19
     2  39.68  39.06 782.99 141.67
     3  39.17  38.14 142.47 781.12
PCIe-Latency
   D\D     0      1      2      3
     0  89.10 743.06 1092.52 1119.50
     1 735.72  89.03 1086.50 1113.06
     2 1118.30 1118.86  89.08 1334.33
     3 1119.99 1119.22 1349.60  89.04
NVLink-Latency
   D\D     0      1      2      3
     0  89.20 470.10 817.41 821.70
     1 470.21  89.05 825.97 817.01
     2 820.17 822.77  89.13 471.11
     3 824.66 822.47 471.07  89.10
```

Listing 30: Yme (Power AC922, 4 GPUs): Tartan result from `scale_up_p2p_packet`

```
===== 33554432 =====
Uni-PCIe-Bandwidth
   D\D     0      1
     0 753.29  30.36
     1  31.07 755.40
Uni-NVLink-Bandwidth
   D\D     0      1
     0 754.86  41.07
     1  41.08 753.29
Bi-PCIe-Bandwidth
   D\D     0      1
     0 784.86  45.91
     1  46.45 789.89
```

```
Bi-NVLink-Bandwidth
   D\D     0       1
     0 785.45  39.66
     1  39.64 790.54
PCIe-Latency
   D\D     0       1
     0  87.80 1100.70
     1 1082.52  87.72
NVLink-Latency
   D\D     0       1
     0  87.77 814.64
     1 815.02  87.77
```

Listing 31: Mini Summit (Power AC922, 2 GPUs): Tartan result from `scale_up_p2p_packet`

```
===== 33554432 =====
Uni-PCIe-Bandwidth
   D\D     0       1       2       3       4       5       6       7       8       9      10
          11      12      13      14      15
     0 806.04    8.38    9.53    9.53  10.88  10.86  10.84  10.84  10.65  10.70
    10.67  10.65  10.67  10.62  10.69  10.66
     1    8.37 807.09    9.50    9.47  10.88  10.86  10.86  10.89  10.69  10.66
    10.61  10.67  10.59  10.57  10.67  10.62
     2    9.47    9.46 807.09    9.46  10.86  10.86  10.82  10.83  10.65  10.68
    10.57  10.67  10.61  10.58  10.63  10.64
     3    9.57    9.56    9.48 809.09  10.91  10.92  10.91  10.90  10.65  10.60
    10.52  10.65  10.59  10.54  10.62  10.55
     4  10.89  10.90  10.89  10.87 813.10    8.39    9.50    9.51  10.62  10.65
    10.60  10.62  10.63  10.58  10.60  10.62
     5  10.92  10.91  10.92  10.97    8.39 809.09    9.53    9.53  10.67  10.68
    10.60  10.67  10.67  10.61  10.67  10.66
     6  10.96  10.92  10.95  10.89    9.58    9.59 809.09    9.48  10.61  10.62
    10.55  10.56  10.55  10.57  10.58  10.60
     7  10.94  10.91  10.92  10.87    9.55    9.55    9.46 811.09  10.57  10.63
    10.58  10.64  10.56  10.54  10.58  10.56
     8  10.85  10.81  10.82  10.83  10.80  10.78  10.75  10.79 807.09    8.30
    9.06    9.09  10.46  10.45  10.47  10.46
     9  10.84  10.83  10.81  10.82  10.83  10.81  10.79  10.78    8.29 811.09
    9.10    9.11  10.50  10.47  10.49  10.48
    10  10.81  10.84  10.79  10.79  10.81  10.82  10.79  10.80    9.07    9.07
    811.09    9.24  10.46  10.46  10.46  10.43
    11  10.86  10.87  10.84  10.82  10.83  10.83  10.86  10.87    9.08    9.10
    9.22 809.09  10.47  10.48  10.47  10.47
    12  10.82  10.82  10.79  10.80  10.78  10.81  10.76  10.77  10.46  10.46
    10.49  10.45 807.09    8.29    9.07    9.09
    13  10.85  10.86  10.82  10.85  10.85  10.84  10.84  10.83  10.46  10.46
    10.40  10.44    8.30 811.09    9.08    9.07
    14  10.86  10.89  10.84  10.86  10.86  10.88  10.84  10.85  10.42  10.43
    10.45  10.49    9.09    9.09 810.65    9.24
    15  10.89  10.91  10.84  10.86  10.85  10.84  10.83  10.81  10.47  10.46
    10.43  10.45    9.10    9.10    9.25 807.53
Uni-NVLink-Bandwidth
   D\D     0       1       2       3       4       5       6       7       8       9      10
          11      12      13      14      15
     0 808.09 136.81 136.52 136.69 137.03 137.02 136.84 137.09 137.10 136.59
    136.85 135.89 136.36 137.17 137.13 136.30
     1 136.39 809.09 136.63 136.57 136.60 137.26 136.65 136.88 136.44 136.31
    136.40 136.84 136.54 136.21 136.39 136.54
```

```
     2 136.64 136.82 811.09 136.68 136.44 136.80 136.50 136.52 136.88 136.46
    136.58 136.43 136.66 136.78 137.07 136.95
     3 136.45 136.65 136.58 809.09 136.51 136.55 136.55 136.99 136.62 136.60
    136.42 136.80 136.69 136.74 136.91 136.67
     4 136.68 136.59 136.63 136.81 809.09 136.72 136.65 136.57 136.40 136.43
    136.52 136.58 136.33 137.07 136.54 136.85
     5 136.66 136.92 137.10 136.88 137.13 811.09 136.69 136.94 136.54 136.41
    136.54 136.84 136.76 136.95 136.93 136.95
     6 136.90 136.88 136.68 136.86 136.96 136.61 811.09 136.93 136.57 137.15
    136.58 136.92 137.07 136.77 136.79 136.62
     7 136.91 136.95 137.02 136.73 136.85 136.83 136.58 809.09 136.64 136.36
    136.88 136.85 136.76 136.78 136.75 136.90
     8 136.44 136.76 136.12 136.27 136.50 136.60 136.53 136.16 807.09 136.84
    136.31 137.24 137.19 136.43 136.54 136.89
     9 136.54 136.64 136.41 136.48 136.46 136.72 136.33 136.46 136.28 809.09
    136.57 136.36 136.76 136.58 136.86 136.61
    10 136.25 136.52 136.81 136.99 136.42 137.17 136.86 137.15 137.36 136.33
    807.09 137.28 137.29 137.27 137.33 137.35
    11 136.30 136.80 137.17 137.12 137.10 137.07 136.88 137.12 137.04 137.17
    136.93 811.09 137.11 137.21 137.42 136.91
    12 136.21 136.28 137.12 137.02 137.18 137.13 137.15 137.33 137.16 137.31
    137.01 137.41 809.09 136.96 136.79 136.33
    13 136.43 136.55 137.27 137.10 136.36 136.97 136.47 137.17 136.85 137.27
    135.91 136.49 137.13 809.09 136.92 136.89
    14 136.72 136.59 136.65 136.58 136.35 136.40 136.37 136.39 136.69 137.11
    137.38 137.52 136.81 136.32 808.28 136.79
    15 136.36 136.79 137.36 136.30 136.53 136.94 137.16 137.32 137.23 137.29
    137.06 137.41 137.09 137.06 137.26 809.77
Bi-PCIe-Bandwidth
   D\D     0      1      2      3      4      5      6      7      8      9     10
        11     12     13     14     15
     0 840.61   8.77  10.27  10.27  16.64  16.65  17.24  17.35  16.63  16.60
    17.02  17.03  16.63  16.62  17.03  17.04
     1   8.77 845.63  10.24  10.26  16.67  16.67  17.13  17.35  16.60  16.58
    17.07  17.07  16.60  16.62  17.06  17.10
     2  10.30  10.27 846.72  10.30  17.34  17.31  18.47  18.47  17.31  17.24
    18.03  18.12  17.28  17.29  18.05  18.19
     3  10.26  10.28  10.30 845.63  17.38  17.34  18.60  18.65  17.27  17.32
    18.15  18.18  17.28  17.26  18.14  18.11
     4  16.64  16.69  17.21  17.18 845.63   8.77  10.26  10.27  16.58  16.61
    17.04  17.06  16.63  16.63  17.12  17.10
     5  16.67  16.71  17.28  17.29   8.77 845.63  10.29  10.30  16.58  16.56
    17.09  17.00  16.64  16.61  17.08  17.02
     6  17.31  17.36  18.51  18.44  10.26  10.28 844.54  10.29  17.26  17.27
    17.99  18.07  17.25  17.27  18.17  18.06
     7  17.17  17.34  18.64  18.65  10.29  10.28  10.28 845.63  17.32  17.23
    17.97  18.22  17.33  17.29  18.10  18.21
     8  16.64  16.61  17.09  17.11  16.61  16.64  17.08  17.07 845.63   8.75
    10.24  10.25  16.54  16.54  16.91  16.86
     9  16.58  16.64  17.09  17.03  16.61  16.62  17.03  17.01   8.75 844.54
    10.25  10.27  16.57  16.53  16.90  16.93
    10  17.31  17.30  18.12  18.16  17.35  17.34  17.98  18.17  10.24  10.24
    845.63  10.27  16.96  16.93  17.28  17.33
    11  17.31  17.27  18.09  18.26  17.28  17.35  18.13  18.25  10.24  10.24
    10.22 845.63  16.93  16.94  17.25  17.26
    12  16.60  16.61  17.03  16.98  16.63  16.63  17.06  17.04  16.55  16.53
    16.95  16.87 844.54   8.75  10.25  10.26
    13  16.58  16.62  17.07  17.06  16.64  16.62  17.05  17.06  16.56  16.54
    16.96  16.96   8.76 844.54  10.27  10.29
```

```
    14  17.32  17.32  18.15  18.24  17.33  17.31  18.22  18.09  16.90  16.91
   17.20  17.22  10.26  10.22 844.40  10.27
    15  17.33  17.27  18.13  18.23  17.31  17.39  18.23  18.13  16.92  16.94
   17.28  17.30  10.25  10.25  10.27 844.60
Bi-NVLink-Bandwidth
   D\D     0      1      2      3      4      5      6      7      8      9     10
         11     12     13     14     15
     0 841.25 252.43 255.44 254.21 255.20 258.75 254.34 257.48 254.64 253.78
   255.91 254.90 254.92 253.46 256.62 258.91
     1 254.02 842.37 254.81 257.41 254.61 258.42 255.20 256.00 253.43 251.87
   257.61 252.84 255.00 255.00 256.60 256.80
     2 252.64 255.20 844.54 256.40 256.00 258.42 254.61 258.63 253.43 255.40
   257.61 254.81 256.00 254.41 258.42 258.22
     3 255.00 254.21 257.41 843.45 255.80 257.81 256.00 258.63 254.61 254.21
   256.40 254.02 253.43 254.81 259.45 259.86
     4 256.20 253.62 256.80 256.80 842.37 258.42 254.61 257.21 254.81 255.40
   258.22 254.61 253.62 255.40 258.22 257.21
     5 253.82 254.21 256.60 257.41 257.41 842.37 255.80 256.40 253.43 253.82
   259.86 256.20 256.80 255.20 259.45 259.45
     6 255.20 253.43 256.80 256.40 256.20 256.20 843.45 255.40 255.60 256.20
   257.81 256.00 254.41 255.00 256.20 258.22
     7 252.06 252.64 257.21 257.61 256.20 258.22 253.23 842.37 255.40 255.40
   256.20 256.00 256.40 254.21 256.60 259.45
     8 254.02 252.26 255.80 256.00 253.82 256.60 254.21 257.41 843.45 253.62
   256.40 255.00 251.87 251.48 253.43 255.40
     9 254.02 252.45 253.82 256.40 255.80 256.40 253.82 256.00 253.03 842.37
   255.80 254.41 253.03 253.03 255.40 256.60
    10 254.81 254.02 257.00 257.41 255.80 258.63 254.21 257.81 255.60 255.60
   844.54 256.40 255.20 254.02 259.45 258.22
    11 253.62 254.81 255.20 256.60 257.81 258.02 253.23 256.40 253.03 255.80
   257.21 843.45 252.26 254.02 257.81 256.80
    12 253.82 252.06 256.20 255.80 254.02 258.02 252.84 256.40 255.00 254.81
   256.40 254.81 843.45 252.84 252.26 255.40
    13 255.40 255.00 257.00 256.20 255.00 257.61 254.81 255.60 255.00 254.61
   258.02 254.81 251.48 843.45 258.63 257.41
    14 253.38 254.33 259.05 258.16 255.76 256.89 253.54 254.42 253.33 256.08
   259.79 255.40 253.70 256.02 842.67 259.37
    15 253.87 253.71 256.02 256.46 257.19 256.97 254.42 256.74 255.54 256.23
   256.09 255.87 255.84 255.06 259.05 842.26
PCIe-Latency
   D\D     0      1      2      3      4      5      6      7      8      9     10
         11     12     13     14     15
     0  81.33 3930.25 3476.11 3478.56 3052.04 3053.20 3055.98 3056.75 3132.36
   3133.17 3132.56 3133.77 3131.74 3132.39 3132.47 3131.79
     1 3948.65  81.34 3485.45 3484.45 3067.27 3065.14 3069.05 3069.41 3136.22
   3135.56 3138.60 3137.08 3136.84 3135.42 3136.95 3134.57
     2 3484.32 3486.02  81.41 3488.37 3056.51 3057.64 3060.09 3062.63 3137.98
   3137.29 3135.43 3132.78 3135.41 3131.40 3133.42 3131.45
     3 3477.46 3482.69 3486.46  81.35 3054.28 3055.11 3059.20 3057.97 3134.81
   3135.10 3133.26 3134.98 3137.82 3134.47 3136.42 3132.55
     4 3057.82 3063.39 3057.19 3057.15  81.35 3940.70 3488.05 3487.39 3136.56
   3136.41 3136.05 3136.16 3136.27 3133.75 3135.20 3132.95
     5 3058.32 3065.21 3055.74 3055.70 3945.80  81.31 3476.57 3472.66 3137.56
   3134.83 3134.37 3135.91 3135.44 3135.12 3136.16 3135.37
     6 3060.39 3066.12 3061.97 3063.60 3484.47 3481.82  81.39 3479.11 3138.39
   3141.16 3135.85 3141.35 3135.29 3138.48 3141.50 3137.30
     7 3060.04 3060.16 3062.15 3060.98 3492.21 3489.03 3493.40  81.35 3146.95
   3137.23 3144.30 3134.52 3145.07 3135.79 3137.91 3134.21
     8 3080.22 3078.35 3079.17 3079.91 3081.09 3079.94 3083.25 3081.61  81.35
```

```
    4024.21 3661.89 3659.70 3199.22 3199.73 3198.16 3199.14
 85   9 3079.07 3077.76 3080.76 3081.41 3081.32 3080.96 3082.05 3084.00 4023.44
    81.33 3662.16 3657.66 3195.52 3196.07 3194.56 3195.22
     10 3081.33 3078.81 3080.57 3081.15 3082.62 3081.34 3078.68 3082.31 3665.89
    3662.30  81.36 3625.09 3187.69 3187.42 3186.09 3188.23
 87   11 3078.64 3076.83 3081.35 3080.34 3079.28 3078.98 3077.98 3079.54 3663.82
    3659.86 3629.93  81.31 3189.53 3188.68 3189.83 3190.66
     12 3080.57 3078.62 3082.21 3082.69 3081.60 3081.74 3081.35 3082.20 3193.07
    3193.87 3193.13 3192.79  81.41 4021.30 3665.71 3657.15
 89   13 3080.24 3077.16 3077.94 3077.65 3079.76 3078.27 3077.08 3078.01 3197.12
    3194.24 3194.41 3194.43 4021.11  81.36 3661.40 3655.29
     14 3078.14 3078.66 3078.20 3081.44 3079.49 3080.17 3076.41 3077.66 3188.24
    3187.79 3188.30 3191.59 3661.03 3665.45  81.43 3626.81
 91   15 3074.77 3073.94 3077.46 3075.29 3078.99 3078.11 3076.67 3077.99 3188.07
    3187.85 3186.12 3186.24 3660.21 3663.90 3627.05  81.38
NVLink-Latency
 93   D\D     0      1      2      3      4      5      6      7      8      9     10
          11     12     13     14     15
      0  81.34 246.53 246.64 246.54 246.48 246.44 246.67 246.23 246.27 246.70
    246.75 246.26 246.24 246.38 246.27 246.40
 95   1 246.41  81.33 246.53 246.50 246.49 246.40 246.54 246.34 246.26 246.70
    246.74 246.30 246.45 246.37 246.27 246.50
      2 246.48 246.47  81.44 246.52 246.46 246.26 246.61 246.34 246.22 246.68
    246.59 246.38 246.38 246.35 246.16 246.40
 97   3 246.37 246.48 246.49  81.34 246.45 246.32 246.46 246.24 246.25 246.71
    246.57 246.16 246.28 246.43 246.26 246.42
      4 246.42 246.45 246.46 246.43  81.35 246.36 246.34 246.14 246.24 246.66
    246.61 246.29 246.37 246.46 246.28 246.42
 99   5 246.50 246.49 246.39 246.44 246.33  81.34 246.51 246.22 246.24 246.60
    246.58 246.23 246.48 246.55 246.37 246.46
      6 246.39 246.48 246.42 246.50 246.30 246.43  81.42 246.08 246.22 246.60
    246.55 246.33 246.40 246.45 246.36 246.45
101   7 246.35 246.66 246.46 246.41 246.35 246.48 246.56  81.31 246.32 246.62
    246.66 246.16 246.37 246.50 246.20 246.46
      8 247.26 247.33 247.25 247.27 247.12 246.79 246.95 246.79  81.48 246.97
    246.88 246.48 246.57 246.54 246.37 246.61
103   9 247.13 247.34 247.20 247.17 247.10 247.04 247.36 246.86 246.59  81.49
    246.83 246.63 246.63 246.63 246.37 246.48
     10 247.10 247.34 247.21 247.18 247.04 246.75 246.86 246.69 246.47 246.81
    81.45 246.39 246.50 246.50 246.26 246.41
105   11 247.08 247.26 247.18 247.00 247.00 246.88 247.04 246.74 246.57 246.64
    246.59  81.33 246.38 246.44 246.26 246.57
     12 247.11 247.40 247.27 247.22 247.14 247.04 246.99 246.59 246.52 246.83
    246.89 246.48  81.40 246.61 246.42 246.51
107   13 247.35 247.23 247.22 247.29 247.01 246.91 246.90 246.57 246.62 246.82
    246.72 246.28 246.49  81.35 246.39 246.56
     14 247.24 247.21 247.25 247.18 247.18 247.13 247.26 246.62 246.64 246.87
    246.86 246.54 246.78 246.73  81.36 246.44
109   15 247.13 247.27 247.31 246.84 246.80 246.78 247.04 246.45 246.47 246.79
    246.67 246.12 246.43 246.60 246.34  81.34
```

Listing 32: DGX-2: Tartan result from `scale_up_p2p_packet`

D.3 Scope

D.3.1 Scope|COMM

More Scope|COMM results can be seen here: *GitHub - Scope|COMM results*


```
{
  "name": "Comm_Memcpy_GPUToHost/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.9656620820363358e+07,
  "cpu_time": 7.9808650222222209e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3479630606242205e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_HostToGPU/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.8470226791169912e+07,
  "cpu_time": 7.8625670000001773e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3683429600089111e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_GPUToWC/log2(N):30/manual_time",
  "iterations": 47,
  "real_time": 1.4764326783095269e+07,
  "cpu_time": 1.4809836510638202e+07,
  "time_unit": "ns",
  "bytes_per_second": 7.2725417133777054e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_WCToGPU/log2(N):30/manual_time",
  "iterations": 47,
  "real_time": 1.4787259234234374e+07,
  "cpu_time": 1.4807360595744727e+07,
  "time_unit": "ns",
  "bytes_per_second": 7.2612632739551361e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
```


Listing 33: Memcpy between GPU 0 and CPU on Yme


```
{
  "name": "Comm_Memcpy_GPUToHost/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.9924696849452123e+07,
  "cpu_time": 8.0076797111111373e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3434418475462261e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
```

```
{
  "name": "Comm_Memcpy_HostToGPU/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.8523172272576228e+07,
  "cpu_time": 7.8676283777778313e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3674203332905825e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_GPUToWC/log2(N):30/manual_time",
  "iterations": 47,
  "real_time": 1.4759395350801183e+07,
  "cpu_time": 1.4805322085106324e+07,
  "time_unit": "ns",
  "bytes_per_second": 7.2749716264068649e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_WCToGPU/log2(N):30/manual_time",
  "iterations": 47,
  "real_time": 1.4788778300615067e+07,
  "cpu_time": 1.4809492595744247e+07,
  "time_unit": "ns",
  "bytes_per_second": 7.2605174151224030e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
```


Listing 34: Memcpy between GPU 1 and CPU on Yme


```
{
  "name": "Comm_Memcpy_GPUToHost/log2(N):30/manual_time",
  "iterations": 6,
  "real_time": 8.6491568634907410e+07,
  "cpu_time": 8.6713417333333120e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.2414410340184826e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 2.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_HostToGPU/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.8842226001951426e+07,
  "cpu_time": 7.8998858444445357e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3618867432452044e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 2.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_GPUToWC/log2(N):30/manual_time",
  "iterations": 23,
  "real_time": 3.0239759417979613e+07,
  "cpu_time": 3.0314348260869499e+07,
  "time_unit": "ns",
```

```
  "bytes_per_second": 3.5507617939631714e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 2.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_WCToGPU/log2(N):30/manual_time",
  "iterations": 27,
  "real_time": 2.5919982197659988e+07,
  "cpu_time": 2.5942769185184665e+07,
  "time_unit": "ns",
  "bytes_per_second": 4.1425253142995430e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 2.0000000000000000e+00
},
```


Listing 35: Memcpy between GPU 2 and CPU on Yme


```
{
  "name": "Comm_Memcpy_GPUToHost/log2(N):30/manual_time",
  "iterations": 6,
  "real_time": 8.6330885688463852e+07,
  "cpu_time": 8.6553008333332568e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.2437516601818914e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 3.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_HostToGPU/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.8932084971004069e+07,
  "cpu_time": 7.9086391333330795e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3603363250754648e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 3.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_GPUToWC/log2(N):30/manual_time",
  "iterations": 23,
  "real_time": 3.0520908372557681e+07,
  "cpu_time": 3.0596830000000041e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.5180532993750458e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 3.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_WCToGPU/log2(N):30/manual_time",
  "iterations": 27,
  "real_time": 2.5906639242613759e+07,
  "cpu_time": 2.5928271259258676e+07,
  "time_unit": "ns",
  "bytes_per_second": 4.1446588804687759e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 3.0000000000000000e+00
},
```


Listing 36: Memcpy between GPU 3 and CPU on Yme

```
{
  "name": "Comm_Memcpy_GPUToHost/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.8821813894642726e+07,
  "cpu_time": 7.8974236888888702e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3622394245268427e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_HostToGPU/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 7.7445870472325221e+07,
  "cpu_time": 7.7595123777778476e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.3864416752649126e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_GPUToWC/log2(N):30/manual_time",
  "iterations": 47,
  "real_time": 1.4767099290769150e+07,
  "cpu_time": 1.4811986212765941e+07,
  "time_unit": "ns",
  "bytes_per_second": 7.2711763011656006e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_WCToGPU/log2(N):30/manual_time",
  "iterations": 47,
  "real_time": 1.4784960789566344e+07,
  "cpu_time": 1.4803410936169857e+07,
  "time_unit": "ns",
  "bytes_per_second": 7.2623920975004074e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 0.0000000000000000e+00
},
```


Listing 37: Memcpy between GPU 0 and CPU on Mini Summit


```
{
  "name": "Comm_Memcpy_GPUToHost/log2(N):30/manual_time",
  "iterations": 6,
  "real_time": 8.6413566023111343e+07,
  "cpu_time": 8.6635182000000790e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.2425616409729317e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_HostToGPU/log2(N):30/manual_time",
  "iterations": 9,
  "real_time": 8.1054421762625381e+07,
  "cpu_time": 8.1210262444443524e+07,
```

```
  "time_unit": "ns",
  "bytes_per_second": 1.3247171476277288e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_GPUToWC/log2(N):30/manual_time",
  "iterations": 23,
  "real_time": 3.0247684406197589e+07,
  "cpu_time": 3.0323988086956393e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.5498314832324684e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
{
  "name": "Comm_Memcpy_WCToGPU/log2(N):30/manual_time",
  "iterations": 27,
  "real_time": 2.5971899360970214e+07,
  "cpu_time": 2.5993310962962613e+07,
  "time_unit": "ns",
  "bytes_per_second": 4.1342445120266670e+10,
  "bytes": 1.0737418240000000e+09,
  "cuda_id": 1.0000000000000000e+00
},
```


Listing 38: Memcpy between GPU 1 and CPU on Mini Summit

D.3.2 Scope|NCCL

More Scope|NCCL results can be seen here: *GitHub - Scope|NCCL results*


```
# 1 GPU
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 69,
  "real_time": 1.0048626071732977e+07,
  "cpu_time": 1.0060178434782607e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.0685459050172652e+11,
  "avg": 1.0041631698608398e+01,
  "bytes": 1.0737418240000000e+09
}
# 2 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 22,
  "real_time": 3.1565079465508461e+07,
  "cpu_time": 3.1575743045457538e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.4016762896899738e+10,
  "avg": 3.2160000409930944e-03,
  "bytes": 1.0737418240000000e+09
}
# 3 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 20,
```

```
  "real_time": 3.5115980729460716e+07,
  "cpu_time": 3.5119151150000505e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.0577013704167439e+10,
  "avg": 3.1359998974949121e-03,
  "bytes": 1.0737418240000000e+09
}
# 4 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 20,
  "real_time": 3.5187609679996967e+07,
  "cpu_time": 3.5197830700001016e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.0514770220677647e+10,
  "avg": 3.2880001235753298e-03,
  "bytes": 1.0737418240000000e+09
}
# 5 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 20,
  "real_time": 3.5256524756550789e+07,
  "cpu_time": 3.5267769399999335e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.0455123737074936e+10,
  "avg": 3.7183999083936214e-03,
  "bytes": 1.0737418240000000e+09
}
# 6 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 20,
  "real_time": 3.5355033352971077e+07,
  "cpu_time": 3.5435224249997079e+07,
  "time_unit": "ns",
  "bytes_per_second": 3.0370267601791630e+10,
  "avg": 3.2800000626593828e-03,
  "bytes": 1.0737418240000000e+09
}
```

Listing 39: Scope|NCCL: DGX-2, size 30

```
# 1 GPU
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 64,
  "real_time": 1.1004969986970536e+07,
  "cpu_time": 1.1020289062499966e+07,
  "time_unit": "ns",
  "bytes_per_second": 9.7568809844213043e+10,
  "avg": 1.0997920036315918e+01,
  "bytes": 1.0737418240000000e+09
}
# 2 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 11,
  "real_time": 6.2659577212550424e+07,
```

```
  "cpu_time": 6.2673325090908736e+07,
  "time_unit": "ns",
  "bytes_per_second": 1.7136116644351288e+10,
  "avg": 4.1119996458292007e-03,
  "bytes": 1.0737418240000000e+09
}
# 3 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 5,
  "real_time": 1.3210091590881348e+08,
  "cpu_time": 1.3211960559999910e+08,
  "time_unit": "ns",
  "bytes_per_second": 8.1281936360015984e+09,
  "avg": 4.1066664271056652e-03,
  "bytes": 1.0737418240000000e+09
}
# 4 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 5,
  "real_time": 1.3051654994487762e+08,
  "cpu_time": 1.3053456880000027e+08,
  "time_unit": "ns",
  "bytes_per_second": 8.2268633706107330e+09,
  "avg": 4.1359998285770416e-03,
  "bytes": 1.0737418240000000e+09
}
```

Listing 40: Scope|NCCL: Yme, size 30

```
# 1 GPU
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 64,
  "real_time": 1.0911102959653363e+07,
  "cpu_time": 1.0924489062499544e+07,
  "time_unit": "ns",
  "bytes_per_second": 9.8408183661215485e+10,
  "avg": 1.0904959678649902e+01,
  "bytes": 1.0737418240000000e+09
}
# 2 GPUs
{
  "name": "NCCL/ops/broadcast/log2(N):30/manual_time",
  "iterations": 5,
  "real_time": 1.2810552120208740e+08,
  "cpu_time": 1.2812559800000116e+08,
  "time_unit": "ns",
  "bytes_per_second": 8.3816982587828074e+09,
  "avg": 4.1439998894929886e-03,
  "bytes": 1.0737418240000000e+09
}
```

Listing 41: Scope|NCCL: Mini Summit, size 30

D.4 DeepBench

```
                  Running training benchmark
                         Times
--------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float                42775
   7680    5481    2560       0       1         float                15526
```

Listing 42: GEMM Training (float): DGX-2 with TC

```
                  Running training benchmark
                         Times
--------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float               126962
   7680    5481    2560       0       1         float                15567
```

Listing 43: GEMM Training (float): DGX-2 without TC

```
                  Running training benchmark
                         Times
--------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float                40241
   7680    5481    2560       0       1         float                16923
```

Listing 44: GEMM Training (float): Yme with TC

```
                  Running training benchmark
                         Times
--------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float               152053
   7680    5481    2560       0       1         float                17107
```

Listing 45: GEMM Training (float): Yme without TC

```
                  Running training benchmark
                         Times
--------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float                37499
   7680    5481    2560       0       1         float                16465
```

Listing 46: GEMM Training (float): Mini Summit with TC

```
                  Running training benchmark
                         Times
--------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float               151428
   7680    5481    2560       0       1         float                16661
```

Listing 47: GEMM Training (float): Mini Summit without TC

```
                    Running training benchmark
                           Times
-------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float               58032
   7680    5481    2560       0       1         float               14454
```

Listing 48: GEMM Training (float): Titan RTX with TC

```
                    Running training benchmark
                           Times
-------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float              134574
   7680    5481    2560       0       1         float               14511
```

Listing 49: GEMM Training (float): Titan RTX without TC

```
                    Running training benchmark
                           Times
-------------------------------------------------------------------------------
    m       n       k       a_t     b_t       precision         time (usec)
   7680   48000    2560       0       0         float              471997
   7680    5481    2560       0       1         float               53839
```

Listing 50: GEMM Training (float): GTX 980

```
 NCCL MPI AllReduce
 Num Ranks: 2
-------------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
-------------------------------------------------------------------------------
         100000          400000           0.0384472          0.0385512
        3097600        12390400            0.169455           0.169464
        4194304        16777216            0.214412           0.214421
        6553600        26214400            0.308329           0.308338
       16777217        67108868             2.70963            2.70974
       38360000       153440000             1.68379            1.68395
       64500000       258000000             2.77863            2.77874

 NCCL MPI AllReduce
 Num Ranks: 3
-------------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
-------------------------------------------------------------------------------
         100000          400000           0.0373482          0.0374285
        3097600        12390400             0.20466            0.20471
        4194304        16777216            0.258751           0.258769
        6553600        26214400            0.376342           0.376359
       16777217        67108868             2.80293            2.80327
       38360000       153440000             1.98109            1.98125
       64500000       258000000             3.24947            3.24967

 NCCL MPI AllReduce
 Num Ranks: 4
-------------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
-------------------------------------------------------------------------------
```

```
         100000         400000           0.0414398           0.0417442
        3097600       12390400            0.227037            0.227077
        4194304       16777216            0.286671            0.286699
        6553600       26214400            0.414103            0.414155
       16777217       67108868             2.84221             2.84256
       38360000      153440000             2.10368             2.10396
       64500000      258000000             3.49185             3.49215

 NCCL MPI AllReduce
 Num Ranks: 8
---------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
---------------------------------------------------------------------------
         100000         400000           0.0518246           0.0522136
        3097600       12390400            0.279964            0.280061
        4194304       16777216            0.351505            0.351608
        6553600       26214400            0.498288             0.49841
       16777217       67108868             2.99622             2.99712
       38360000      153440000             2.30725             2.30818
       64500000      258000000             3.91183             3.91286

 NCCL MPI AllReduce
 Num Ranks: 12
---------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
---------------------------------------------------------------------------
         100000         400000           0.0570434            0.057639
        3097600       12390400            0.326875            0.327046
        4194304       16777216            0.401075             0.40122
        6553600       26214400            0.555655            0.555805
       16777217       67108868             3.05683             3.05844
       38360000      153440000             2.47745             2.47889
       64500000      258000000             3.97833             3.97976

 NCCL MPI AllReduce
 Num Ranks: 16
---------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
---------------------------------------------------------------------------
         100000         400000           0.0709125           0.0717716
        3097600       12390400            0.366346            0.366564
        4194304       16777216            0.435344            0.435558
        6553600       26214400            0.600155             0.60035
       16777217       67108868             3.10012             3.10208
       38360000      153440000             2.60025             2.60229
       64500000      258000000               4.0425             4.04476
```

Listing 51: NCCL MPI AllReduce: DGX-2

```
 NCCL MPI AllReduce
 Num Ranks: 2
---------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
---------------------------------------------------------------------------
         100000         400000           0.0493164           0.0493305
        3097600       12390400            0.260277            0.260281
        4194304       16777216            0.334999            0.335002
        6553600       26214400            0.510145            0.510148
       16777217       67108868              3.3676             3.36766
```

```
       38360000      153440000             2.71301             2.71303
       64500000      258000000             4.42318             4.42321

 NCCL MPI AllReduce
 Num Ranks: 3
-------------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
-------------------------------------------------------------------------------
         100000         400000            0.0798487           0.0798962
        3097600       12390400             0.777842            0.777848
        4194304       16777216              1.01134             1.01135
        6553600       26214400              1.52334             1.52335
       16777217       67108868                6.797              6.79718
       38360000      153440000              8.51575             8.51579
       64500000      258000000              14.3225             14.3225

 NCCL MPI AllReduce
 Num Ranks: 4
-------------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
-------------------------------------------------------------------------------
         100000         400000            0.0852101            0.085241
        3097600       12390400             0.876076            0.876083
        4194304       16777216              1.11588             1.11589
        6553600       26214400                1.626             1.62601
       16777217       67108868               6.9926              6.99267
       38360000      153440000               9.2942              9.29426
       64500000      258000000              15.5052             15.5052
```

Listing 52: NCCL MPI AllReduce: Yme

```
 NCCL MPI AllReduce
 Num Ranks: 2
-------------------------------------------------------------------------------
    # of floats    bytes transferred    Avg Time (msec)    Max Time (msec)
-------------------------------------------------------------------------------
         100000         400000            0.0612649           0.0612709
        3097600       12390400              0.66023            0.660237
        4194304       16777216             0.883019             0.88303
        6553600       26214400              1.34992             1.34992
       16777217       67108868              6.37216             6.37219
       38360000      153440000              7.55603             7.55608
       64500000      258000000               12.4679             12.4679
```

Listing 53: NCCL MPI AllReduce: Mini Summit