Intra-Node/Inter-Node Communication - Leonardo CINECA Cluster - Booster Partition

Franco N. Merenda, Katia M. Perchet
Department of Computer Science
University of Salerno
Italy

***Abstract*—In distributed systems, communication often represents a significant portion of the total computational cost. Therefore, understanding the capabilities of the hardware for which a distributed solution is being developed, can greatly enhance both power and compute efficiency.**

As modern applications increasingly leverage GPU computing, it is crucial for developers to be skilled with the underlying hardware and the associated software libraries that maximize the potential of these advanced systems

In this work, we present an overview of the communication capabilities provided by the Leonardo CINECA Cluster - Booster Partition, at both the Intra-Node and Inter-Node levels, by measuring the bandwidth and latency for GPU-to-GPU communication in multi-GPU applications, and CPU-to-GPU communication using CUDA, SYCL, and MPI technologies.

Our results show that leveraging technologies like NVLink for intra-node GPU-to-GPU communication can improve performance by up to 350% over traditional CPU-based communication via PCIe, highlighting the significant advantages of multi-GPU systems. In addition, we compare SYCL and CUDA APIs by examining the performance trade-offs versus programming complexity. For inter-node communication, we compare MPI and CUDA-Aware MPI to highlight their respective advantages.

I. Introduction

High-performance computing (HPC) relies heavily on efficient communication mechanisms, therefore understanding the performance characteristics of different configurations is crucial for optimizing computational workloads.

Motivation. The purpose of this paper is to demonstrate the communication capabilities within a single node (Intra-Node) and between different nodes (Inter-Node) of the *Leonardo Supercomputer - Booster Partition* [1], ranked 7th on the June 2024 TOP500 list [2].

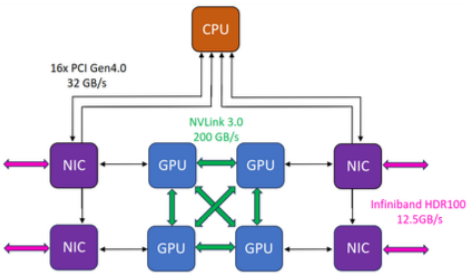

Fig. 1. Leonardo Supercomputer - Booster Partition - Node Communication Detail - Reproduced from [1]

For the intra-node experiments, we used Compute Unified Device Architecture (CUDA) [3] and SYCL [4] application programming interfaces (APIs) to compare bandwidth and latency results while transferring data between CPU-GPU and GPU-GPU. For the inter-node experiments, we used the Message Passing Interface (MPI) Library [5] without CUDA-Aware and CUDA-Aware to compare bandwidth and latency results as well, using the OSU benchmarks (Ohio State University Benchmarks) [6] with host-to-host buffers and device-to-device buffers.

Understanding the trade-offs between technologies and libraries could help optimize the programming model and assist programmers in tuning parallelization to take advantage of the communication capabilities provided by the *Leonardo Booster Partition*.

II. Related work

In this section, we describe related works on the topic of this paper, focusing on Intra-Node and Inter-Node communication technologies available in the context of High-Performance Computing.

A. Trends and Network Solutions for HPC

A. Tekin et al. [7] provides an overview of the state-of-the-art interconnect solutions for HPC and future trends in HPC. It reviews trends in CPUs from different perspectives, including manufacturing, instruction sets, memory, and packaging across various architectures (x86_64, ARM, POWER). It also covers GPUs and FPGAs and their impact on the HPC context. His work provides an overview of communication technologies available between nodes, such as InfiniBand, RoCE, and Next Data Rate, as well as intra-node communication technologies like PCIe, CCIX, and Gen-Z. Finally, it evaluates power efficiency in HPC systems and discusses future trends in this area.

B. Intra-Node Communication

Li et al. [8] provided a comprehensive evaluation of five modern GPU interconnect technologies, including PCIe, NVLink, NV-SLI, NVSwitch, and GPUDirect. Their study, conducted across various platforms such as the NVIDIA DGX-1, DGX-2, and Summit supercomputers, highlighting how interconnect technology significantly affects multi-GPU communication efficiency. The authors identified four key

NUMA effects arising from GPU topology, connectivity, and routing, underscoring the importance of selecting the right GPU combination for maximizing performance within a multi-GPU node.

C. Inter-Node Communication

Wang et al. [9] proposed a GPU-Aware MPI that leverages RDMA (Remote Direct Memory Access) technology to improve communication between GPUs across nodes without staging data through the CPU's host memory. Their design enables direct GPU-to-GPU communication using RDMA-enabled interconnects, which significantly reduces data transfer overheads. By allowing MPI to directly recognize GPU memory buffers, their approach optimizes communication performance, particularly for large-scale applications. This study demonstrated that GPU-Aware MPI reduces latency and enhances bandwidth, offering a substantial improvement over traditional, non-GPU-aware MPI implementations in RDMA-enabled environments.

D. Intra/Inter Node - Recent Work

De Sensi et al. [10] conducted a detailed investigation into GPU-to-GPU communication across modern supercomputers, including Alps [11], Leonardo, and LUMI [12], which feature distinct architectures such as NVIDIA H100, A100, and AMD MI250X GPUs. Their study focused on intra-node and inter-node communication performance across up to 4096 GPUs, offering insights into the potential of high-bandwidth interconnects such as NVLink and InfiniBand.

The authors highlighted several optimization opportunities at both the network and software levels, which are crucial for enhancing the overall efficiency of multi-GPU supercomputing. This work provides important guidance for system architects and developers in improving GPU communication performance.

E. Contrast to Our Work

While prior research has extensively evaluated modern GPU interconnects and their performance in multi-GPU and distributed environments, our contribution focuses on a targeted examination of Intra-Node and Inter-Node communication specifically using SYCL and CUDA. We conduct small-scale experiments on the Leonardo Cineca Cluster - Booster Partition, analyzing data transfer efficiency between CPU, GPU, and across nodes using various APIs, including PCIe, NVLink, MPI, and CUDA-aware MPI. This study aims to highlight the trade-offs between programming complexity and performance degradation in these contexts. Although our experiments are more confined in scope compared to broader analyses, they offer valuable insights specifically relevant to the Leonardo Booster Partition.

III. Background: CPU-GPU/GPU-GPU Communication Technologies and Libraries

This section provides an overview of the fundamental technologies that enable seamless data transfer and communication in high-performance computing systems. We begin by discussing PCI Express (PCIe) [13], the standard for CPU-to-GPU data transfer. We then explore NVLink, which offers high-bandwidth, low-latency communication between GPUs, enhancing multi-GPU configurations.

Next, we provide an overview of MPI and its CUDA-Aware extension, which facilitate efficient inter-node GPU communication by allowing direct GPU-to-GPU data transfers across nodes enabled by Remote Direct Memory Access (RDMA) [14].

Additionally, we highlight the role of CUDA and SYCL frameworks in managing parallel processing and data transfers between CPUs and GPUs.

PCIe. High-speed serial computer expansion bus designed to replace older standards like PCI, PCI-X, and AGP. It provides a robust interconnect solution for connecting peripheral devices to a CPU, offering greater speed and scalability.

It's a point-to-point communication system where devices are directly connected to the CPU through individual lanes, each consisting of two pairs of wires for sending and receiving data. Unlike the older parallel bus architecture, PCIe uses a fully serial interface, which reduces signal degradation over longer distances and increases data throughput.

NVIDIA NVLink. High-speed interconnect designed to improve communication between GPUs and CPUs in HPC environments. It overcomes the limitations of traditional PCIe connections by enabling higher bandwidth, lower latency, and more scalable multi-GPU configurations. NVLink enables GPUs to share memory more efficiently, crucial for data-intensive tasks like AI training, large-scale simulations, and deep learning.

In particular, NVLink 3.0, which is the version that *Leonardo Booster Partition Nodes* uses, significantly boosts performance with full-duplex communication, allowing data to be sent and received simultaneously. Each NVLink 3.0 connection delivers 25 GB/s per direction or 50 GB/s bidirectional bandwidth. In a typical setup using 4 NVLink connections between two GPUs, the total bidirectional bandwidth reaches 200 GB/s (100 GB/s upstream and 100 GB/s downstream).

CUDA. NVIDIA's parallel computing platform and API that enables developers to use GPUs for general-purpose computing. By leveraging the massively parallel architecture of GPUs, CUDA allows significant acceleration of computational tasks compared to traditional CPU-only processing.

CUDA provides a set of libraries, compilers, and development tools that make it easier to offload complex mathematical computations to the GPU, optimizing performance for scientific simulations, machine learning, data analytics, and other high-performance computing (HPC) workloads. It is widely used in HPC environments for its ability to parallelize tasks and efficiently distribute workloads across thousands of GPU cores.

SYCL. Defines abstractions to enable heterogeneous device programming, an important capability in the modern world which has not yet been solved directly in ISO C++. SYCL has evolved with the intent of influencing C++ direction around

heterogeneous compute by creating productized proof points that can be considered in the context of C++ evolution.

A major goal of SYCL is to enable different heterogeneous devices to be used in a single application — for example simultaneous use of CPUs, GPUs, and FPGAs. Although optimized kernel code may differ across the architectures (since SYCL does not guarantee automatic and perfect performance portability across architectures), it provides a consistent language, APIs, and ecosystem in which to write and tune code for accelerator architectures. An application can coherently define variants of code optimized for architectures of interest, and can find and dispatch code to those architectures.

SYCL uses generic programming with templates and generic lambda functions to enable higher-level application software to be cleanly coded with optimized acceleration of kernel code across an extensive range of acceleration backend APIs, such as OpenCL and CUDA.

The popularity of SYCL has given rise to an ecosystem of different implementations, such as:

- Open SYCL [15] (formerly known as hipSYCL)
- neoSYCL [16]
- triSYCL [17]
- Intel® oneAPI tools [18]

MPI. Is a widely adopted parallel programming model designed for distributed-memory systems in HPC. MPI enables processes to communicate with each other by passing messages, making it a critical tool for parallel computing in large-scale clusters. It abstracts the complexity of handling inter-process communication (IPC) across different nodes, offering features like point-to-point communication (e.g., MPI_Send and MPI_Recv) and collective communication (e.g., MPI_Bcast, MPI_Reduce).

RDMA. Is a key technology that enables high-speed data transfer between the memory of different nodes without involving the CPU in the data transfer process. This significantly reduces latency and CPU overhead in distributed systems.

In an HPC context, RDMA allows GPUs (or CPU memory) in different nodes to communicate efficiently by bypassing the host CPU for memory copies. Instead, the NIC (Network Interface Card) directly accesses the memory of another node over a high-speed network (e.g., InfiniBand). This results in low-latency and high-throughput communication, making RDMA a critical component in high-speed GPU-to-GPU communication in distributed clusters.

Unified Communication X (UCX). Is a high-performance, low-latency communication framework designed for HPC applications. It is a unified communication library that provides a single API for multiple types of data transfer technologies, such as RDMA, TCP/IP, shared memory, and GPU Direct RDMA. UCX [19] integrates with both RDMA and CUDA-Aware MPI, making it an important component in high-speed, distributed computing environments.

UCX acts as a transport layer that abstracts and optimizes data communication, enabling applications to use the most efficient protocol based on available hardware. In GPU-to-GPU communication, UCX can utilize RDMA to bypass the CPU and enable direct memory access between GPUs across different nodes. When used with CUDA-Aware MPI, UCX helps facilitate efficient data transfers by automatically selecting the best data path, whether through RDMA, NVLink, or PCIe, depending on the system architecture

CUDA-Aware MPI. Extends the traditional MPI model to work efficiently with GPU buffers by allowing the direct transfer of data between GPUs across nodes without needing to first copy the data into host CPU memory. In traditional MPI implementations, when sending or receiving data from a GPU, the data must first be copied from the GPU device memory to the CPU host memory before it can be sent to another node. Similarly, on the receiving side, data must first arrive at the host memory before being transferred back to the GPU. This intermediate memory copy can introduce significant latency and overhead.

With CUDA-Aware MPI, the MPI implementation is optimized to recognize GPU memory, enabling direct communication between GPU memory spaces using MPI functions like MPI_Send and MPI_Recv. This reduces memory copies and greatly improves communication bandwidth and latency between GPUs in distributed systems.

IV. Experimental Setup

The hardware configuration used for conducting all the experiments described in Experiment Scenarios (Section V), is the Booster Partition Nodes of the Leonardo CINECA Cluster. For intra-node scenarios, a single node was used, while for multi-node scenarios, two nodes from the same partition were employed.

In addition to the hardware configuration, the libraries and drivers used in the experimental setup are also detailed below.

A. Booster Node Specifications

- **Model**: Atos BullSequana X2135 "Da Vinci" single-node GPU Blade
- **Processor**: Intel Xeon Platinum 8358, 32 cores, 2.60GHz - TDP 250W - Intel Ice Lake CPU
- **GPU**: 4 x NVIDIA Ampere GPUs/node, 64GB HBM2e NVLink 3.0 (200GB/s)
- **RAM**: 512 (8x64) GB DDR4 3200 MHz
- **Operating System**: RedHat Enterprise Linux 8.6
- **Storage**: 137.6 PB based on DDN ES7990X and Hard Drive Disks / 5.7 PB based on DDN ES400NVX2 and Solid State Drives (Fast Tier)
- **Network**: DragonFly+ 200 Gbps (NVIDIA Mellanox Infiniband HDR) / 2 x dual port HDR100 per node

B. Libraries and Drivers

- **CUDA**: 12.1
- **Intel® oneAPI tools**: v2024.2.0
- **Codeplay plugin oneAPI for NVIDIA® GPUs [20]**: v2024.2.0
- **NVIDIA Driver**: 530.30.02
- **Open MPI**: 4.1.6 - Compiled with
  - **UCX**: 1.13.0
  - **CUDA**: 12.3

V. Experiment Scenarios

To evaluate the performance and communication capabilities of the Leonardo Booster Partition, we considered two types of communications:

1) **Intra-Node communication**: Communication that happens within the node (Technologies like: PCI-Express, NVLink).
2) **Inter-Node communication**: Communication that occurs between nodes (Technologies like: MPI, CUDA-Aware MPI)

For **Intra-Node communication**, we conducted a series of experiments using different APIs, specifically SYCL and CUDA. The experiments aimed to measure data transfer efficiency between the CPU and GPU, as well as between GPUs. Each experiment included a designated number of warm-up and execution runs, with 20 warm-up runs and 30 execution runs performed to obtain results for the following scenarios:

- **CPU to GPU - PCI Express** - Communication using SYCL and CUDA
- **GPU to GPU - NVLink Unidirectional** - Communication using SYCL and CUDA
- **GPU to GPU - NVLink Bidirectional** - Communication using SYCL and CUDA

For **Inter-Node communication** we conducted a series of communication experiments using MPI and CUDA-Aware MPI. The experiments focused on measuring data transfer efficiency between two GPUs in different nodes by using the OSU Benchmarks v7.3:

- **GPU to GPU - MPI** - Communication between *hosts buffers* with *osu_bw* and *osu_latency* benchmark.
- **GPU to GPU - CUDA-Aware MPI** - Communication between *device (GPUs) buffers* with *osu_bw* and *osu_latency* benchmark.

VI. Results

In this section, we present the performance results of the communication mechanisms explored in this work. The results are divided into two categories: intra-node and inter-node communication, reflecting the different architectures and data transfer patterns within a single node and multiple nodes. All the reported data points of the figures in this section are the median results of multiple times described in Experiment Scenarios Section.

A. Intra-Node Communication

For **Intra-Node communication** we conducted a series of communication experiments [21] using two different APIs, SYCL and CUDA. The experiments focused on measuring data transfer efficiency using different buffer sizes between CPU and GPU, as well as between GPUs.

- **CPU to GPU - PCI Express** - Communication using SYCL and CUDA, Fig. 2.
- **GPU to GPU - NVLink Unidirectional** - Communication using SYCL and CUDA, Fig. 3.
- **GPU to GPU - NVLINK Bidirectional** - Communication using SYCL and CUDA, Fig. 4.

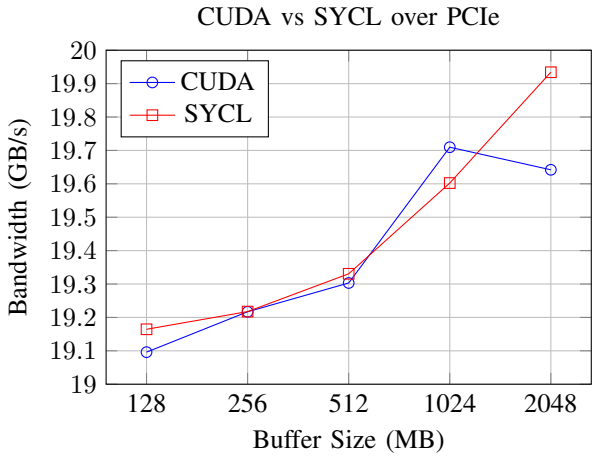

Fig. 2. GPU to GPU over PCI-Express CUDA vs SYCL.

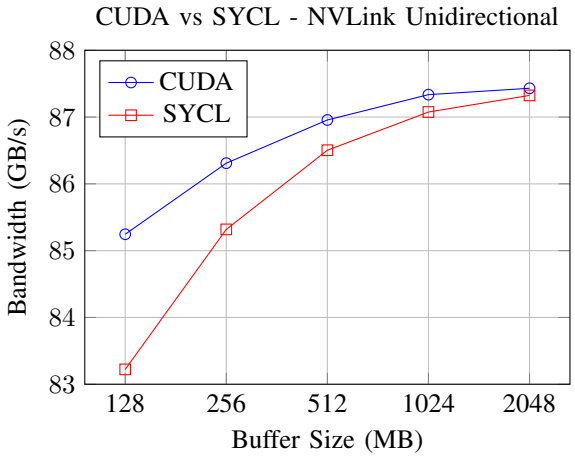

Fig. 3. GPU to GPU over NVLink Unidirectional - CUDA vs SYCL.

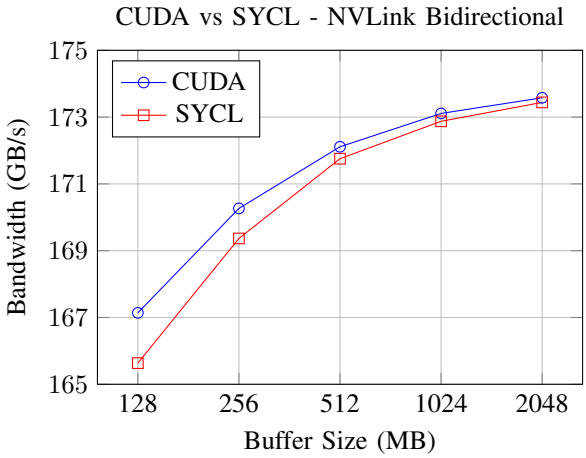

Fig. 4. GPU to GPU over NVLink Bidirectional - CUDA vs SYCL.

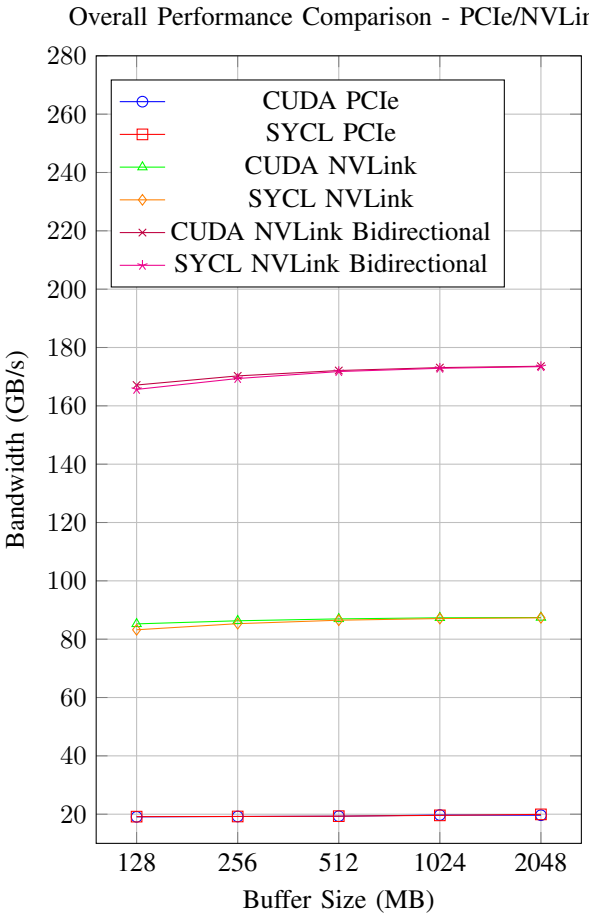

Fig. 5. Comparison of CUDA vs SYCL performance over PCIe, NVLink, and Bidirectional NVLink with different buffer sizes.

From Figs. 2, 3, and 4, we observe that using solutions like NVLink, which bypass CPU-based communication, can increase communication speed by up to 350.46%, depending on the data size. This clearly indicates that multi-GPU systems significantly benefit from NVLink's advantages.

From Fig. 5, we see a comparison of the bandwidth capabilities of NVLink versus PCIe. It is important to note that these results come from peer-to-peer communication. This means that if we perform communication with additional GPUs, the bandwidth will further increase due to NVLink's capabilities.

Regarding programming complexity, CUDA is generally more complex compared to SYCL, which is simpler. The performance differences between CUDA and SYCL range from 0.12% to 2.37% across the different experiments. Thus, by accepting a minor trade-off in performance, programmers can use SYCL without significantly sacrificing performance. Of course, more thorough tests are needed to validate this hypothesis, but this approach could lead to more maintainable code across different types of hardware.

B. Inter-Node Communication

For **Inter-Node communication** we made use of MPI and CUDA-Aware MPI APIs. The experiments focused on measuring data transfer efficiency using different buffer sizes between CPU-to-CPU and GPU-to-GPU between different nodes.

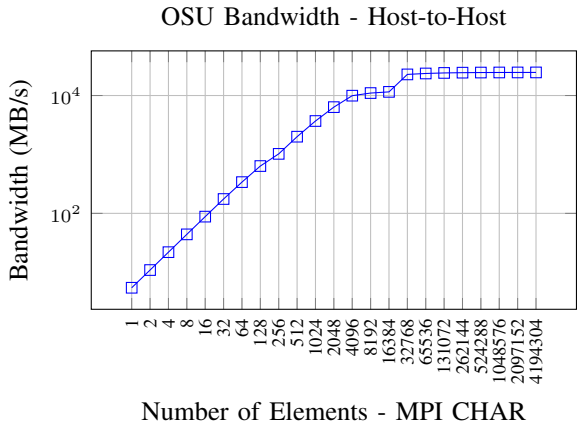

Fig. 6. OSU Bandwidth Benchmark - Host-to-Host

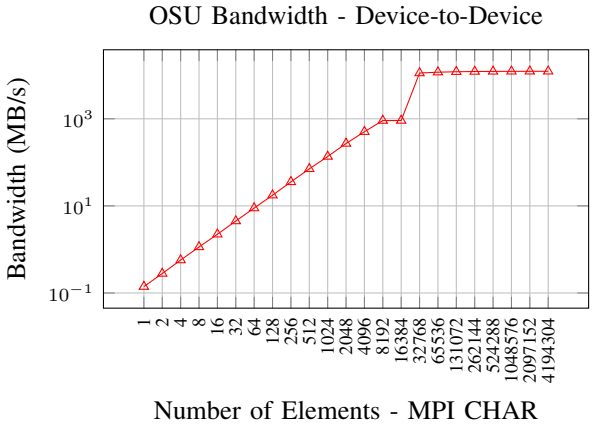

Fig. 7. OSU Bandwidth Benchmark - Device-to-Device

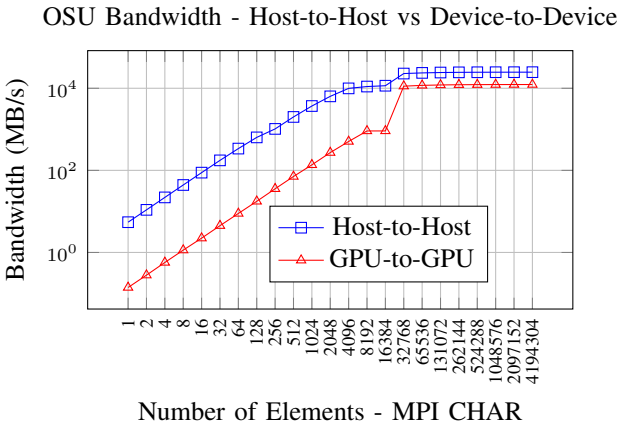

Fig. 8. Host-to-Host vs Device-to-Device

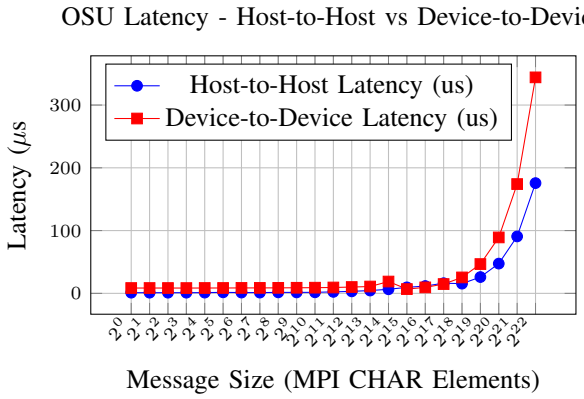

Fig. 9. OSU Latency Benchmark - Host to Host vs Device to Device

For Inter-Node communication, as seen in Fig. 6, host-to-host communication using MPI without CUDA reaches close to the theoretical maximum, especially for larger data sizes, with performance percentages ranging from 0.022% to 98.68%.

In contrast, for device-to-device communication (Fig. 7), the maximum performance observed is approximately 49.34%. The reason device-to-device communication does not reach closer to the theoretical maximum may be that the NVIDIA driver used was not compiled with GDRCopy support for UCX, which is necessary for CUDA-Aware MPI communication.

From the *osu_latency* benchmarks in host-to-host and device-to-device communication from Fig. 9, results reveal a clear distinction between these two scenarios. Host-to-host communication shows consistently low latency, starting at 1.05 µs and gradually increasing with larger message sizes, but still maintaining reasonable latency even at higher data sizes. In contrast, device-to-device communication demonstrates significantly higher initial latencies (around 8.5 µs) and exhibits a more pronounced increase as message sizes grow, reaching over 344 µs for the largest message size tested.

One key observation is that while host-to-host communication can maintain close to optimal latency across varying message sizes, device-to-device communication experiences a more rapid degradation in performance, particularly at larger message sizes. This discrepancy may be attributed to the lack of optimization, such as the absence of GDRCopy support in the tested system for device-to-device transfers.

VII. Conclusions

The experiments demonstrate that using solutions like NVLink, which fully exploit the hardware provided by the Leonardo Booster Partition Node, significantly boosts intra-node communication performance, making it ideal for multi-GPU setups compared to PCIe solutions.

Regarding inter-node communication, we observed that MPI without CUDA achieves near-maximum bandwidth efficiency, especially with large data, while CUDA-Aware MPI falls short at 49.34% due to missing GDRCopy support. By addressing this issue in Leonardo, CUDA-Aware MPI could achieve bandwidth efficiency closer to the theoretical maximum, allowing GPUs to handle communication and freeing up CPU compute power. This could reduce power consumption and shift communication load to the network layer.

Regarding programming complexity and maintainability across different hardware types, while CUDA offers slightly better performance than SYCL, the latter provides an easier programming experience with minimal trade-offs. This makes it a viable option for developers aiming for hardware portability, particularly as domain-specific hardware becomes more prevalent. A unified programming framework like SYCL, which does not sacrifice much performance, could simplify development from a programming perspective.

In summary, we can conclude that by knowing what is the underlying hardware support, which libraries are available and being aware of the communication capabilities, programmers can take advantage of the compute power available, making possible a greater performance and reducing power consumption at the same time.

Bibliography

[1] M. Turisini, G. Amati, and M. Cestari, "Leonardo: A pan-european pre-exascale supercomputer for hpc and ai applications," *Journal of Large Scale Research Facilities*, vol. 8, no. A186, 2024.

[2] TOP500, "Leonardo - top500 june 2024." https://www.top500.org/lists/top500/list/2024/06/, 2024. Ranked 7th on the TOP500 list, CINECA, Italy.

[3] N. Corporation, "CUDA Toolkit Documentation." https://docs.nvidia.com/cuda/. Accessed September 2024.

[4] K. Group, "SYCL Specification." https://www.khronos.org/sycl/. Accessed September 2024.

[5] Message Passing Interface Forum, "MPI Standard." https://www.mpi-forum.org/docs/. Accessed September 2024.

[6] Ohio State University, "OSU Micro-Benchmarks." https://mvapich.cse.ohio-state.edu/benchmarks/, 2024. Accessed: September 2024.

[7] A. Tekin, A. Durak, C. Piechurski, D. Kaliszan, F. A. Sungur, F. Robertsen, and P. Gschwandtner, "State-of-the-Art and Trends for Computing and Interconnect Network Solutions for HPC and AI," 2021.

[8] A. Li, S. Song, J. Li, N. Tallent, K. Barker, J. Chen, and X. Liu, "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, pp. 55–68, July 2019. Manuscript received 11 Mar. 2019; revised 30 May 2019; accepted 27 June 2019. Date of publication 15 July 2019; date of current version 18 Dec. 2019. Corresponding author: Ang Li.

[9] H. Wang, S. Potluri, D. Bureddy, D. Panda, and C. Rosales, "GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1795–1807, August 2013. Manuscript received 8 Mar. 2013; revised 30 July 2013; accepted 3 Aug. 2013.

Date of publication 28 Aug. 2013; date of current version 17 Sept. 2014. Recommended for acceptance by S. Ranka.

[10] Daniele De Sensi and Lorenzo Pichetti and Flavio Vella and Tiziano De Matteis and Zebin Ren and Luigi Fusco and Matteo Turisini and Daniele Cesarini and Kurt Lust and Animesh Trivedi and Duncan Roweth and Filippo Spiga and Salvatore Di Girolamo and Torsten Hoefler, "Exploring GPU-to-GPU Communication: Insights into Supercomputer Interconnects," *arXiv preprint arXiv:2408.14090v1*, 2024. Published in Proceedings of The International Conference for High Performance Computing Networking, Storage, and Analysis (SC '24).

[11] C. S. N. S. Centre, "Alps supercomputer." https://www.cscs.ch/computers/alps, 2024. Accessed: 2024-09-03.

[12] E. J. Undertaking, "Lumi supercomputer." https://www.lumi-supercomputer.eu, 2024. Accessed: 2024-09-03.

[13] A. Verma and P. K. Dahiya, "PCIe BUS: A State-of-the-Art-Review," *IOSR journal of VLSI and Signal Processing*, vol. 07, pp. 24–28, 2017.

[14] N. Corporation, "NVIDIA GPUDirect RDMA Documentation." https://docs.nvidia.com/cuda/gpudirect-rdma/index.html, 2024. Accessed: September 2024.

[15] O. SYCL, "Open sycl: An open source implementation of the sycl standard." https://www.opensycl.org/, 2024. Accessed: September 2024.

[16] neoSYCL, "neosycl: Sycl implementation for intel® cpus." https://github.com/neo-project/neoSYCL, 2024. Accessed: September 2024.

[17] triSYCL, "trisycl: A sycl implementation." https://github.com/triSYCL/triSYCL, 2024. Accessed: September 2024.

[18] I. Corporation, "Intel® oneapi toolkits." https://software.intel.com/content/www/us/en/develop/tools/oneapi.html, 2024. Accessed: September 2024.

[19] OpenUCX Project, "Unified Communication X (UCX)." https://github.com/openucx/ucx/?tab=readme-ov-file, 2024. Accessed: September 2024.

[20] C. Software, "Intel® oneapi toolkits for nvidia." https://developer.codeplay.com/products/oneapi/nvidia/2024.2.1, 2024. Accessed: September 2024.

[21] y3rbiadit0, "HPC Final Project." https://github.com/y3rbiadit0/hpc-final-project, 2024. Accessed: 2024-09-04.