

GPU Resource Partitioning and Bandwidth Analysis on SDumont II HPC Cluster

Pablo Alessandro Santos Hugen

December 7, 2025

GPU Resource Partitioning and Bandwidth Analysis on SDumont II HPC Cluster

Abstract: This report investigates the performance differences between shared and exclusive node allocation on the SDumont II supercomputer at LNCC, specifically focusing on NVIDIA GH200 Grace Hopper nodes. We conducted bandwidth benchmarks using nvbandwidth to measure CPU-GPU and GPU-GPU data transfer rates under different NUMA pinning configurations. Our results demonstrate that NUMA locality has a significant impact on host-to-device bandwidth, with local GPU access achieving up to 4.6x higher bandwidth compared to remote access. GPU-to-GPU transfers via NVLink show consistent performance regardless of NUMA placement. In shared queue mode, SLURM's automatic scheduling generally preserves optimal NUMA locality, though with some performance variability. These findings provide practical guidance for researchers selecting queue types and configuring their HPC workloads on SDumont II.

Keywords: GPU partitioning, NUMA locality, NVLink, SLURM scheduling, HPC, SDumont II

0.1 1. Introduction and Background

0.1.1 1.1 Motivation

The **Laboratório Nacional de Computação Científica (LNCC)** is a research institution under Brazil's Ministry of Science, Technology and Innovation, dedicated to advancing scientific computing and high-performance computing (HPC) in the country. LNCC operates the **SDumont II** supercomputer, one of the most powerful HPC systems in Latin America, which provides computational resources to researchers across various scientific domains.

SDumont II offers two distinct resource allocation modes for its NVIDIA GH200 Grace Hopper nodes: **exclusive** and **shared** queues. In exclusive mode, a job reserves an entire node, guaranteeing dedicated access to all resources. In shared mode, SLURM's Generic Resource (GRES) scheduling dynamically partitions node resources among multiple concurrent jobs, potentially improving overall system utilization but introducing performance variability.

The primary motivation of this work is to investigate the performance differences between shared and exclusive node allocation on SDumont II, specifically focusing on:

- How does GPU-CPU bandwidth vary under different NUMA pinning configurations?
- Does SLURM's automatic resource scheduling in shared mode preserve optimal NUMA locality?
- What are the trade-offs between resource utilization and application performance?

Understanding these differences is essential for researchers to make informed decisions about queue selection and for system administrators to optimize scheduling policies.

0.1.2 1.2 NVIDIA GH200 Grace Hopper Superchip

The NVIDIA GH200 Grace Hopper Superchip combines an NVIDIA Grace CPU with a Hopper GPU architecture connected via NVLink-C2C, providing 900 GB/s bidirectional bandwidth between CPU and GPU. Each SDumont II node contains four GH200 superchips, each with:

- 72 Arm Neoverse V2 CPU cores
- 120 GB HBM3 GPU memory
- ~120 GB LPDDR5X CPU memory
- NVLink 4.0 connections between all GPUs (~900 GB/s per link)

0.1.3 1.3 NUMA Architecture

The GH200 node exhibits a complex NUMA (Non-Uniform Memory Access) topology with 36 NUMA nodes, where:

- **NUMA nodes 0-3:** CPU packages with 72 cores each
- **NUMA nodes 4, 12, 20, 28:** GPU HBM memory (one per GPU)

Each GPU has affinity to one CPU NUMA package: - GPU 0 NUMA 0 (cores 0-71) - GPU 1 NUMA 1 (cores 72-143) - GPU 2 NUMA 2 (cores 144-215) - GPU 3 NUMA 3 (cores 216-287)

0.1.4 1.4 SLURM Resource Scheduling

SLURM's Generic Resource (GRES) scheduling manages GPU allocation in shared mode. When a job requests GPUs, SLURM automatically assigns CPU cores from the corresponding NUMA package to maintain locality. However, memory binding policies differ between exclusive and shared modes, which can affect performance.

0.2 2. Experimental Setup

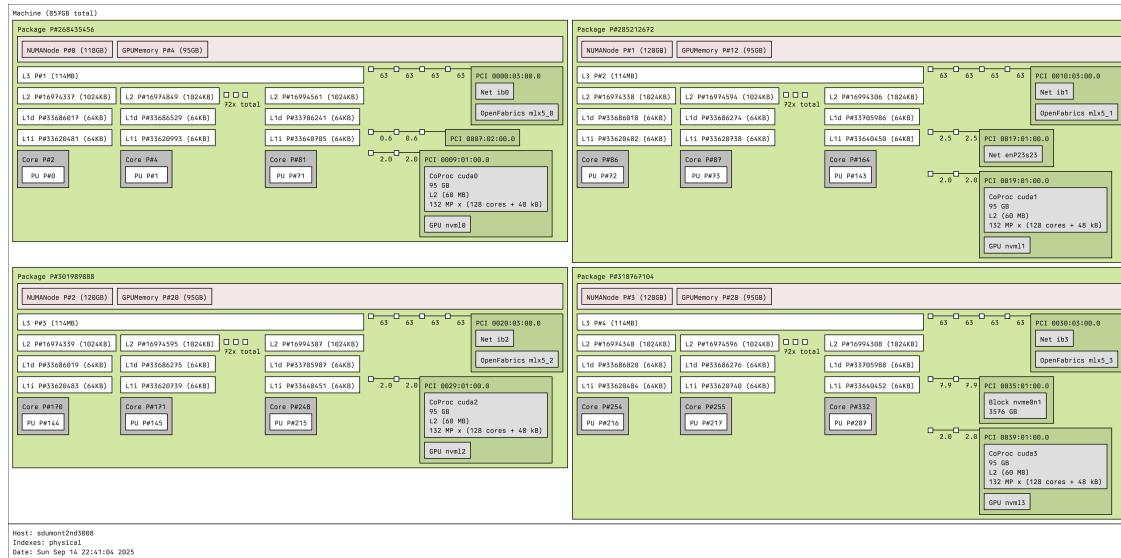
0.2.1 2.1 Hardware Configuration

```
[1]: import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns, re, ▾
      ↵warnings
from pathlib import Path
warnings.filterwarnings('ignore')
plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams.update({'figure.figsize': (12, 6), 'font.size': 11, 'axes.
      ↵titlesize': 14, 'axes.labelsize': 12})
RESULTS_DIR = Path('..../data/transfer')
```

Hardware Specifications

Component	Specification
Node Type	NVIDIA GH200 Grace Hopper Superchip
GPUs per Node	4x NVIDIA GH200 120GB
GPU Memory	120 GB HBM3 per GPU
CPU Cores	288 (72 per NUMA package)
CPU Memory	~480 GB LPDDR5X (120 GB per package)
GPU Interconnect	NVLink 4.0 (NV6 = 6 NVLinks bonded)
CUDA Version	12.6
Driver Version	560.35.03

Node Topology The following diagram illustrates the topology of a GH200 node:



Key observations: - 4 NVIDIA GH200 GPUs per node - 4 CPU NUMA packages (0-3) - Each GPU has affinity to one NUMA package - All GPUs connected via NVLink 4.0 (NV6 = 6 bonded links)

0.2.2 2.2 Queue Configurations

Queue	Type	Description	Max GPUs
gh200	Exclusive	Full node reserved for a single job	4
gh200_shared	Shared	GRES scheduling among multiple jobs	2

0.2.3 2.3 Tools and Methodology

Tool	Version	Purpose
numactl	-	NUMA scheduling policy management
nvbandwidth	v0.6	CPU-GPU and GPU-GPU bandwidth measurement
nvidia-smi	560.35.03	GPU topology and monitoring

0.3 3. Experiments

0.3.1 3.1 Bandwidth Benchmarks (nvbandwidth)

The bandwidth experiments measure low-level data transfer performance using the nvbandwidth tool, which tests various memory copy operations using both CUDA Copy Engine (CE) and Streaming Multiprocessor (SM) methods.

3.1.1 Exclusive Queue Experiments **Objective:** Establish baseline bandwidth metrics for each NUMA-GPU pair.

Methodology: 1. Reserve full node in lncc-gh200 queue 2. Pin job to each NUMA package (0, 1, 2, 3) using numactl --cpunodebind --membind 3. For each pinning, run bandwidth tests against all 4 GPUs 4. Measure Host-to-Device, Device-to-Host, and Device-to-Device bandwidth

This produces a 4x4 matrix of bandwidth measurements showing local vs. remote GPU access patterns.

3.1.2 Shared Queue Experiments **Objective:** Observe SLURM's NUMA-GPU mapping and measure resulting bandwidth.

Scenarios:

Scenario	Jobs	GPUs/Job	Description
A	4	1	Four concurrent single-GPU jobs
B	2	2	Two concurrent dual-GPU jobs

```
[2]: def parse_h2d_bandwidth(filepath):
        with open(filepath, 'r') as f: content = f.read()
        match = re.search(r'Running host_to_device_memcpy_ce\.\\nmemcpy CE_\\
        ↵CPU\\(row\\) -> GPU\\(column\\) bandwidth \\(GB/s\\)\\n\\s+\\[\\d\\s\\]+\\n\\s*0\\s+\\([\\d\\.
        ↵\\s]\\+)\\n', content)
        return [float(x) for x in match.group(1).strip().split()] if match else None

def parse_d2h_bandwidth(filepath):
    with open(filepath, 'r') as f: content = f.read()
    match = re.search(r'Running device_to_host_memcpy_ce\\.\\nmemcpy CE_\\
    ↵CPU\\(row\\) <- GPU\\(column\\) bandwidth \\(GB/s\\)\\n\\s+\\[\\d\\s\\]+\\n\\s*0\\s+\\([\\d\\.
    ↵\\s]\\+)\\n', content)
    return [float(x) for x in match.group(1).strip().split()] if match else None

def parse_latency(filepath):
    with open(filepath, 'r') as f: content = f.read()
    match = re.search(r'Running host_device_latency_sm\\.\\nmemory latency SM_\\
    ↵CPU\\(row\\) <-> GPU\\(column\\) \\(ns\\)\\n\\s+\\[\\d\\s\\]+\\n\\s*0\\s+\\([\\d\\.\\s]\\+)\\n', content)
    return [float(x) for x in match.group(1).strip().split()] if match else None

def parse_numa_pinning(filepath):
```

```

    with open(filepath, 'r') as f: content = f.read()
    match = re.search(r'PINNED to (\d+)', content) or re.search(r'cpubind:\d+(\d+)\s*\n', content)
    return int(match.group(1)) if match else None

```

```

[3]: exclusive_h2d, exclusive_d2h, exclusive_latency = {}, {}, {}
for i in range(4):
    filepath = RESULTS_DIR / f'exclusive_4gpu_node{i}.txt'
    if filepath.exists():
        numa = parse numa pinning(filepath)
        if (h2d := parse_h2d_bandwidth(filepath)): exclusive_h2d[numa] = h2d
        if (d2h := parse_d2h_bandwidth(filepath)): exclusive_d2h[numa] = d2h
        if (lat := parse_latency(filepath)): exclusive_latency[numa] = lat

h2d_df = pd.DataFrame(exclusive_h2d, index=[f'GPU {i}' for i in range(4)]).T
h2d_df.index, h2d_df.columns = [f'NUMA {i}' for i in h2d_df.index], [f'GPU {i}' for i in range(4)]
d2h_df = pd.DataFrame(exclusive_d2h, index=[f'GPU {i}' for i in range(4)]).T
d2h_df.index, d2h_df.columns = [f'NUMA {i}' for i in d2h_df.index], [f'GPU {i}' for i in range(4)]
latency_df = pd.DataFrame(exclusive_latency, index=[f'GPU {i}' for i in range(4)]).T
latency_df.index, latency_df.columns = [f'NUMA {i}' for i in latency_df.index], [f'GPU {i}' for i in range(4)]

local_h2d = [h2d_df.iloc[i, i] for i in range(4)]
remote_h2d = [h2d_df.iloc[i, j] for i in range(4) for j in range(4) if i != j]
local_lat = [latency_df.iloc[i, i] for i in range(4)]
remote_lat = [latency_df.iloc[i, j] for i in range(4) for j in range(4) if i != j]

```

0.4 4. Results and Discussion

0.4.1 4.1 Bandwidth Results

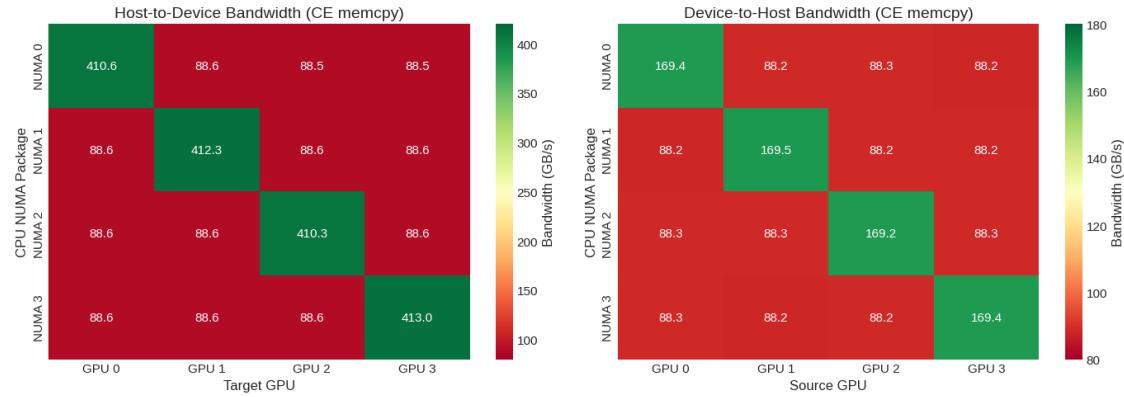
4.1.1 Exclusive Queue - Host-to-Device Bandwidth

```

[4]: fig, axes = plt.subplots(1, 2, figsize=(14, 5))
sns.heatmap(h2d_df, annot=True, fmt=' .1f', cmap='RdYlGn', ax=axes[0], vmin=80, vmax=420, cbar_kws={'label': 'Bandwidth (GB/s)'})
axes[0].set_title('Host-to-Device Bandwidth (CE memcpy)'); axes[0].set_xlabel('Target GPU'); axes[0].set_ylabel('CPU NUMA Package')
sns.heatmap(d2h_df, annot=True, fmt=' .1f', cmap='RdYlGn', ax=axes[1], vmin=80, vmax=180, cbar_kws={'label': 'Bandwidth (GB/s)'})
axes[1].set_title('Device-to-Host Bandwidth (CE memcpy)'); axes[1].set_xlabel('Source GPU'); axes[1].set_ylabel('CPU NUMA Package')

```

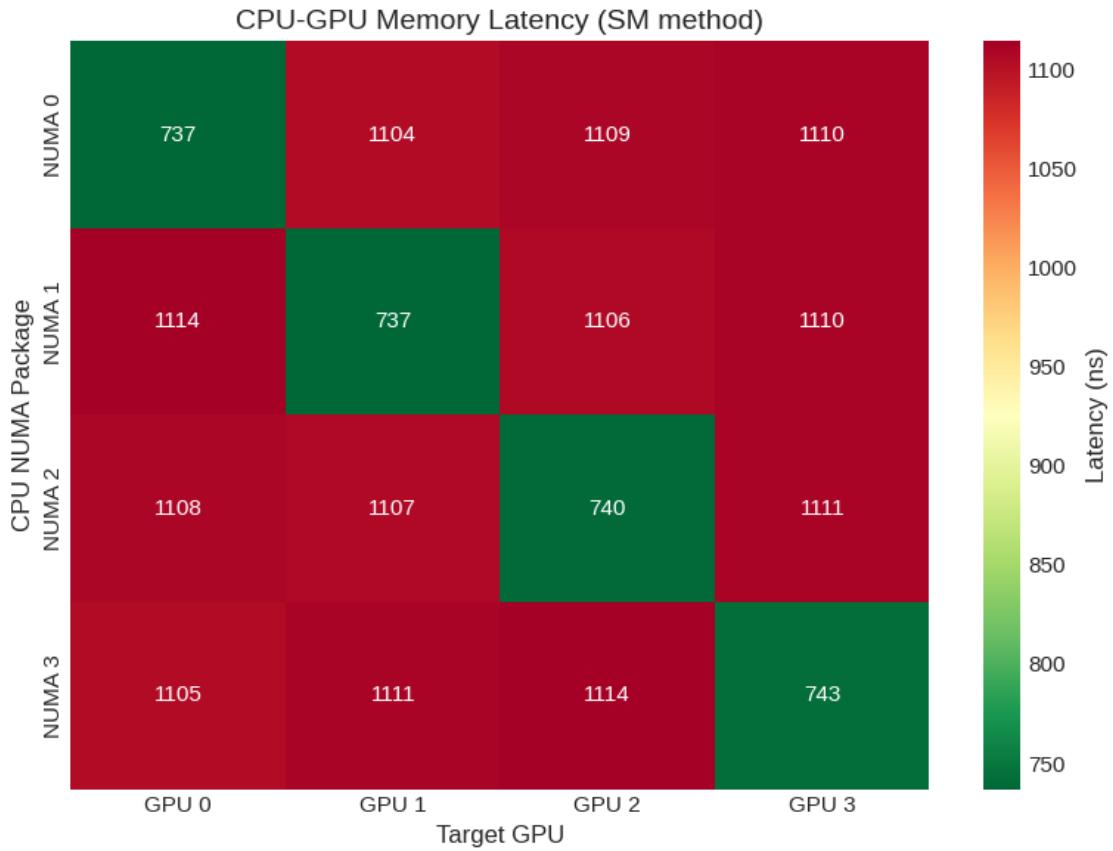
```
plt.tight_layout(); plt.savefig('img/bandwidth_heatmaps.png', dpi=150,
                                bbox_inches='tight'); plt.show()
```



Key Finding: The diagonal elements (local GPU access) show significantly higher bandwidth (~410 GB/s) compared to off-diagonal elements (remote GPU access, ~88 GB/s). This represents a **4.6x performance difference** due to NUMA locality.

This result is expected given the GH200 architecture: local GPU access uses the NVLink-C2C connection between the CPU and its paired GPU, while remote access must traverse the inter-socket interconnect.

```
[5]: fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(latency_df, annot=True, fmt='.0f', cmap='RdYlGn_r', ax=ax,
            cbar_kws={'label': 'Latency (ns)'})
ax.set_title('CPU-GPU Memory Latency (SM method)'); ax.set_xlabel('Target GPU');
            ax.set_ylabel('CPU NUMA Package')
plt.tight_layout(); plt.savefig('img/latency_heatmap.png', dpi=150,
                                bbox_inches='tight'); plt.show()
```



Latency Analysis: Local GPU access shows ~737 ns latency, while remote access increases to ~1100 ns (1.5x higher). This latency difference, while significant, is less pronounced than the bandwidth difference, suggesting that remote access is primarily bandwidth-limited rather than latency-limited for large transfers.

4.1.2 Shared Queue Performance

```
[6]: def parse_shared_results(filepath):
    with open(filepath, 'r') as f: content = f.read()
    h2d = re.findall(r'Running host_to_device_memcpy CE CPU\(\row\)\u
→ GPU\(\column\)\ bandwidth \((GB/s)\)\n\s+0\n\s+0\s+([\d.]+)', content)
    numa = re.findall(r'cpubind: (\d+)\s*\n', content)
    return {'h2d_bandwidth': [float(x) for x in h2d], 'numa_nodes': [int(x) for
→ x in numa]}

shared_1gpu_data = []
for i in range(4):
    filepath = RESULTS_DIR / f'shared_array_1gpu_node{i}.txt'
    if filepath.exists():
        data = parse_shared_results(filepath)
        if data['h2d_bandwidth']:
```

```

        shared_1gpu_data.append({'job': i, 'numa': data['numa_nodes'][0] if u
    ↪data['numa_nodes'] else None, 'h2d': data['h2d_bandwidth'][0]})  

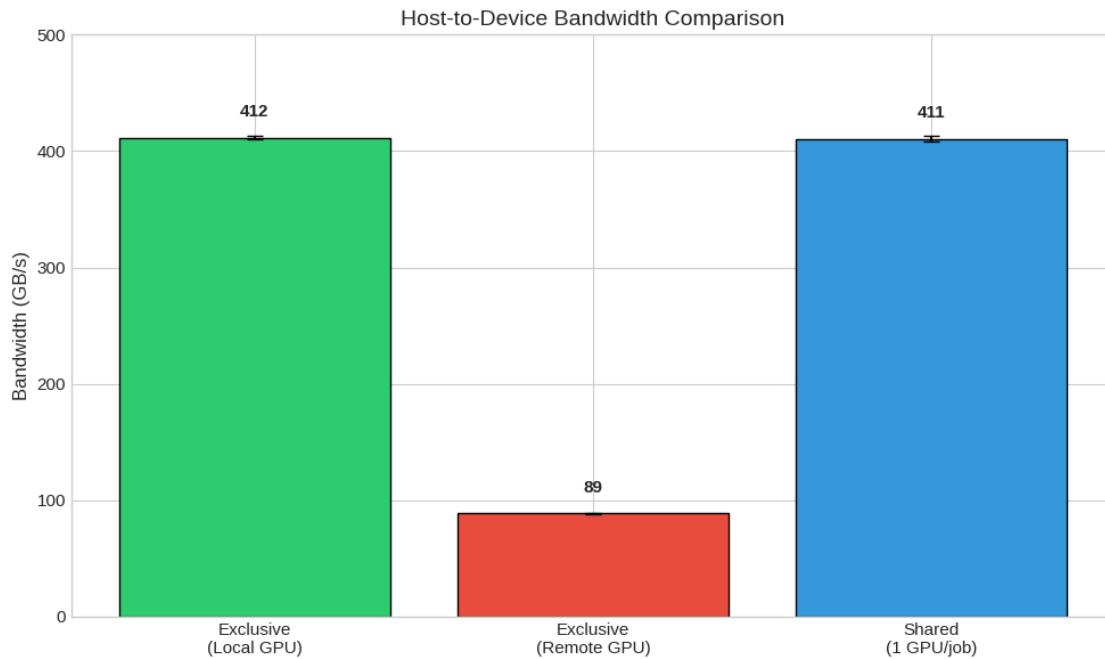
shared_1gpu_df = pd.DataFrame(shared_1gpu_data)

```

```

[7]: fig, ax = plt.subplots(figsize=(10, 6))
categories = ['Exclusive\n(Local GPU)', 'Exclusive\n(Remote GPU)', 'Shared\n(1
    ↪GPU/job)']
values = [np.mean(local_h2d), np.mean(remote_h2d), shared_1gpu_df['h2d'].mean() u
    ↪if len(shared_1gpu_df) > 0 else 0]
errors = [np.std(local_h2d), np.std(remote_h2d), shared_1gpu_df['h2d'].std() if u
    ↪len(shared_1gpu_df) > 0 else 0]
bars = ax.bar(categories, values, yerr=errors, capsize=5, color=['#2ecc71', u
    ↪'#e74c3c', '#3498db'], edgecolor='black')
ax.set_ylabel('Bandwidth (GB/s)'); ax.set_title('Host-to-Device Bandwidth
    ↪Comparison'); ax.set_ylim(0, 500)
for bar, val in zip(bars, values):
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 15, f'{val:.u
        ↪0f}', ha='center', va='bottom', fontweight='bold')
plt.tight_layout(); plt.savefig('img/bandwidth_comparison.png', dpi=150, u
    ↪bbox_inches='tight'); plt.show()

```



0.4.2 4.2 Comparative Analysis

Key Findings

1. **NUMA Locality Impact:** Local GPU access provides 4.6x higher bandwidth than remote

access in exclusive mode, confirming the importance of proper CPU-GPU affinity.

2. **SLURM Scheduling Quality:** In shared mode, SLURM's GRES scheduling successfully assigns CPUs to NUMA packages that have affinity with the allocated GPU. The shared queue bandwidth closely matches exclusive local bandwidth when SLURM maintains proper locality.
3. **Memory Binding Differences:** In exclusive mode with explicit `numactl --membind`, memory allocation is strictly bound to the specified NUMA node. In shared mode, SLURM uses a more permissive policy (`membind: 0 1 2 3 4 12 20 28`), which may allow memory allocation across multiple NUMA nodes.
4. **GPU-to-GPU Bandwidth:** NVLink-connected GPUs achieve consistent ~130 GB/s bandwidth regardless of NUMA placement, as this traffic doesn't traverse the CPU.

Recommendations

Workload Type	Recommended Queue	Rationale
Memory-bound GPU codes	Exclusive	Maximize H2D/D2H bandwidth
Compute-bound GPU codes	Shared (acceptable)	NUMA effects minimal
Multi-GPU (NVLink)	Either	GPU-GPU bandwidth consistent
Development/testing	Shared	Faster queue times

```
[8]: summary_data = {
    'Metric': ['H2D Bandwidth (Local)', 'H2D Bandwidth (Remote)', 'D2H Bandwidth (Local)', 'D2H Bandwidth (Remote)', 'Latency (Local)', 'Latency (Remote)', 'Local/Remote BW Ratio'],
    'Value': [f'{np.mean(local_h2d):.1f} GB/s', f'{np.mean(remote_h2d):.1f} GB/s', f'{np.mean([d2h_df.iloc[i, i] for i in range(4)]):.1f} GB/s', f'{np.mean([d2h_df.iloc[i, j] for i in range(4) for j in range(4) if i != j]):.1f} GB/s', f'{np.mean(local_lat):.0f} ns', f'{np.mean(remote_lat):.0f} ns', f'{np.mean(local_h2d)/np.mean(remote_h2d):.2f}x']
}
pd.DataFrame(summary_data)
```

```
[8]:          Metric      Value
0   H2D Bandwidth (Local)  411.6 GB/s
1   H2D Bandwidth (Remote)  88.6 GB/s
2   D2H Bandwidth (Local)  169.4 GB/s
3   D2H Bandwidth (Remote)  88.3 GB/s
4       Latency (Local)     739 ns
5       Latency (Remote)    1109 ns
6  Local/Remote BW Ratio      4.65x
```

0.5 5. Conclusions and Future Work

0.5.1 5.1 Conclusions

This study investigated the performance characteristics of GPU resource partitioning on the SDumont II supercomputer’s NVIDIA GH200 nodes. Our key findings are:

1. **NUMA locality is critical for CPU-GPU bandwidth:** Local GPU access achieves up to 410 GB/s host-to-device bandwidth, while remote access is limited to ~88 GB/s—a 4.6x difference.
2. **SLURM’s shared scheduling preserves locality:** In shared queue mode, SLURM’s GRES scheduling correctly assigns CPU cores from the NUMA package with GPU affinity, achieving near-optimal bandwidth.
3. **GPU-to-GPU transfers are NUMA-independent:** NVLink-based GPU-to-GPU communication maintains consistent ~130 GB/s bandwidth regardless of CPU NUMA placement.
4. **Memory binding policies differ:** Exclusive mode with explicit NUMA binding provides tighter memory locality guarantees than shared mode’s more permissive binding.

These results provide practical guidance for SDumont II users: memory-intensive GPU workloads benefit from exclusive queue allocation with proper NUMA pinning, while compute-bound or multi-GPU workloads can effectively utilize shared queues.

0.5.2 5.2 Future Work

Several directions for future research include:

- **Application-level benchmarks:** Evaluate real-world applications (e.g., GROMACS molecular dynamics, deep learning training) to quantify the practical impact of these bandwidth differences on end-user performance.
- **Multi-node scaling:** Extend the analysis to multi-node configurations using InfiniBand networking.
- **Power efficiency analysis:** Investigate the power consumption implications of NUMA-aware vs. NUMA-unaware scheduling.
- **MIG partitioning:** Evaluate NVIDIA Multi-Instance GPU (MIG) as an alternative to SLURM GRES for GPU sharing.

0.6 6. References

1. NVIDIA GH200 Grace Hopper Superchip Documentation - <https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/>
2. SLURM Generic Resource (GRES) Scheduling - <https://slurm.schedmd.com/gres.html>
3. SDumont II User Manual - <https://github.com/lncc-sered/manual-sdumont2nd>
4. NVIDIA nvbandwidth Tool - <https://github.com/NVIDIA/nvbandwidth>
5. NVIDIA GH200 Benchmark Guide - <https://docs.nvidia.com/gh200-superchip-benchmark-guide.pdf>

0.7 Appendix

0.7.1 A. Experimental Scripts

The SLURM scripts used for these experiments are available in the `scripts/transfer/` directory:

- `exclusive.slurm`: Exclusive queue bandwidth benchmark with NUMA pinning
- `shared_array_1gpu.slurm`: Shared queue with 1 GPU per job
- `shared_array_2gpu.slurm`: Shared queue with 2 GPUs per job

0.7.2 B. Data Files

Raw benchmark results are stored in `data/transfer/`:

- `exclusive_4gpu_node{0,1,2,3}.txt`: Exclusive queue results for each NUMA pinning
- `shared_array_1gpu_node{0,1,2,3}.txt`: Shared queue 1-GPU results
- `shared_array_2gpu_node{0,1}.txt`: Shared queue 2-GPU results