Benchmarking MPI for Deep Learning and HPC Workloads

Yıltan Hassan Temuçin

Parallel Processing Research Laboratory (PPRL)
Computing At Extreme Scale Advanced Research Laboratory (CEASER)
Department of Electrical and Computer Engineering
Queen's University, Canada

Benchmarking in the Data Center: Expanding to the Cloud
Febuary 25th 2023

Outline

Introduction
- Open MPI + UCX

Benchmarking for MPI-Based Deep Learning
- RMA and Collective Communication Design
- Micro-Benchmarks
- Application Results

Benchmarking for MPI-Partitioned Communication
- MPI Partitioned Point-to-Point Communication
- Overhead
- Perceived Bandwidth
- Sweep3D Communication Pattern

Conclusion And Future Work

Introduction

- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
  - There has been a recent popularity of systems with accelerators

Introduction

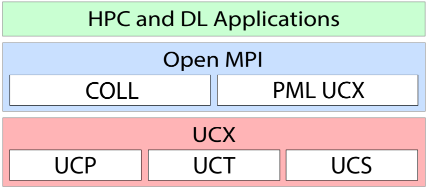
- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
  - There has been a recent popularity of systems with accelerators
- The Message Passing Interface (MPI)
  - Popular parallel programming model in HPC
  - Provides multiple communication APIs
    - Point-to-point
    - Partitioned point-to-point
    - RMA
    - Collective Communication (`MPI_Allreduce`, `MPI_Bcast`, etc.)

Introduction

- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
  - There has been a recent popularity of systems with accelerators
- The Message Passing Interface (MPI)
  - Popular parallel programming model in HPC
  - Provides multiple communication APIs
    - Point-to-point
    - Partitioned point-to-point
    - RMA
    - Collective Communication (`MPI_Allreduce`, `MPI_Bcast`, etc.)
- MPI based Deep Learning on HPC systems
  - As the complexity of DL models grow we move towards using the aggregate power of HPC systems

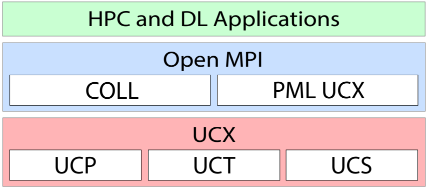
Open MPI + UCX

- **UCX** provides abstract communication primitives to best utilise hardware
  - Point-to-point implemented upon RMA Put/Get operations



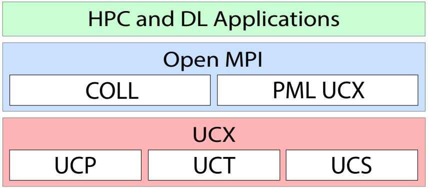
Open MPI + UCX

- **UCX** provides abstract communication primitives to best utilise hardware
  - Point-to-point implemented upon RMA Put/Get operations
- **Open MPI** is an open source MPI implementation
  - Point-to-point communication directly relies on UCX for data transfers
  - Collective communication are internally built with point-to-point primitives



Open MPI + UCX

- **UCX** provides abstract communication primitives to best utilise hardware
  - Point-to-point implemented upon RMA Put/Get operations
- **Open MPI** is an open source MPI implementation
  - Point-to-point communication directly relies on UCX for data transfers
  - Collective communication are internally built with point-to-point primitives



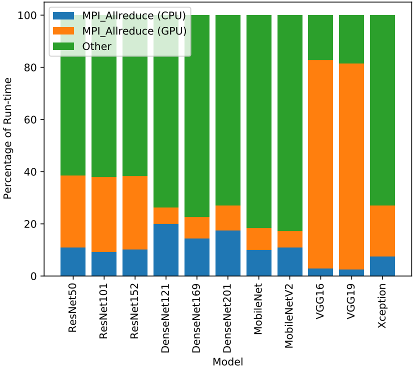
Research Goals

- Improve the performance of GPU MPI communication for Deep Learning
- Obtain a better understanding of the MPI Partitioned Interface

Benchmarking for MPI-Based Deep Learning

MPI-based Deep Learning

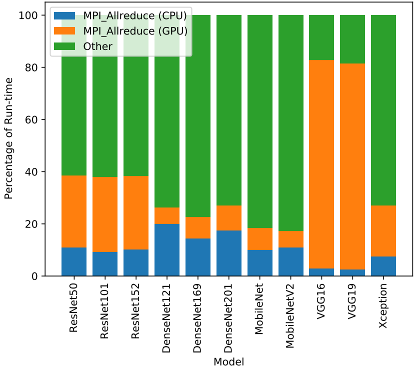
- Distributed Deep Learning using Horovod is possible with models from:



Impact of MPI_Allreduce on a single IBM AC922 node

MPI-based Deep Learning

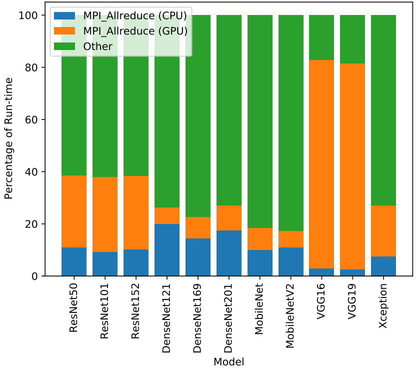
- Distributed Deep Learning using Horovod is possible with models from:
  - TensorFlow
  - PyTorch
  - MXNet



Impact of MPI_Allreduce on a single IBM AC922 node

MPI-based Deep Learning

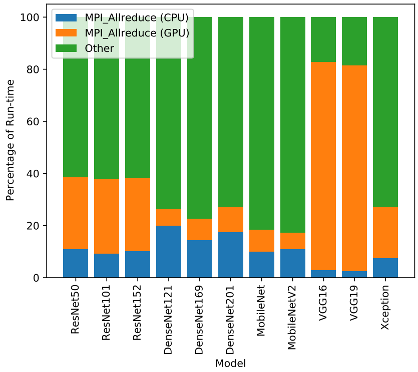
- Distributed Deep Learning using Horovod is possible with models from:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`



Impact of MPI_Allreduce on a single IBM AC922 node

MPI-based Deep Learning

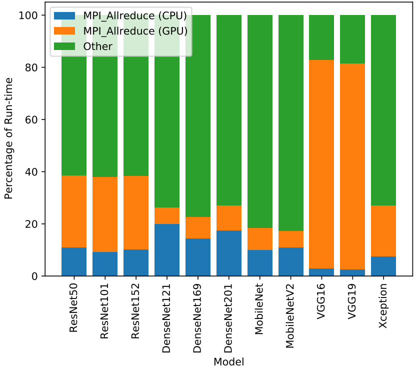
- Distributed Deep Learning using Horovod is possible with models from:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`
  - 17-83% of training time was spent in `MPI_Allreduce`



Impact of MPI_Allreduce on a single IBM AC922 node

MPI-based Deep Learning

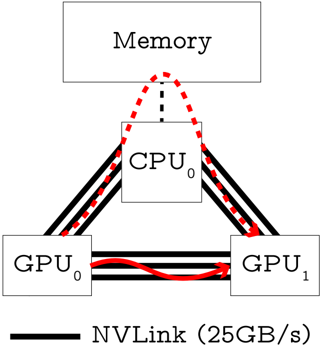
- Distributed Deep Learning using Horovod is possible with models from:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`
  - 17-83% of training time was spent in `MPI_Allreduce`
  - Up to 80% of runtime was spent in a GPU based `MPI_Allreduce`



Impact of MPI_Allreduce on a single IBM AC922 node

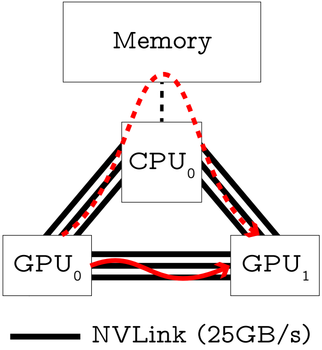
Multi-Path Copy Motivation

- MPI sends data directly from GPU_0 to GPU_1
  - Uses a zero copy put operation in UCX
  - (As shown by the solid red line)



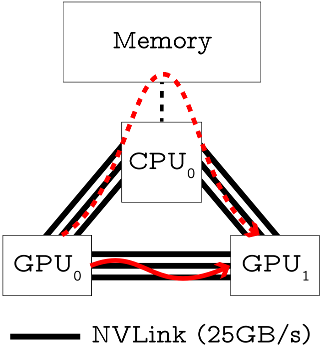
Multi-Path Copy Motivation

- MPI sends data directly from GPU_0 to GPU_1
  - Uses a zero copy put operation in UCX
  - (As shown by the solid red line)
- Six idle NVLinks connected to the host



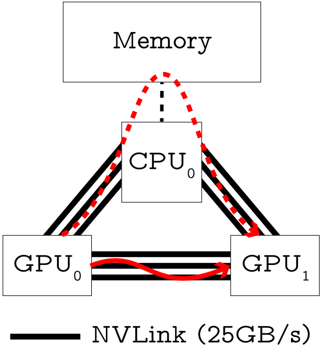
Multi-Path Copy Motivation

- MPI sends data directly from GPU_0 to GPU_1
  - Uses a zero copy put operation in UCX
  - (As shown by the solid red line)
- Six idle NVLinks connected to the host
- A large amount of unused potential bandwidth



Multi-Path Copy Motivation

- MPI sends data directly from GPU_0 to GPU_1
  - Uses a zero copy put operation in UCX
  - (As shown by the solid red line)
- Six idle NVLinks connected to the host
- A large amount of unused potential bandwidth

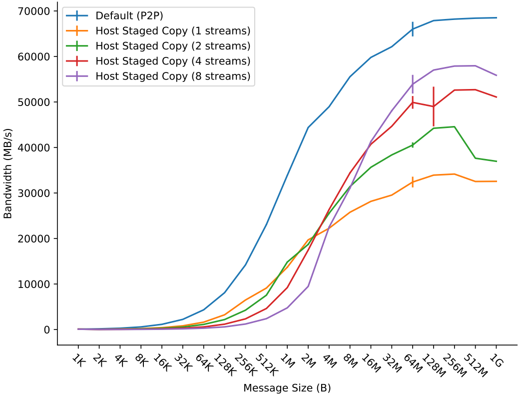


Research Question

Can we design a mechanism to use all communication paths?

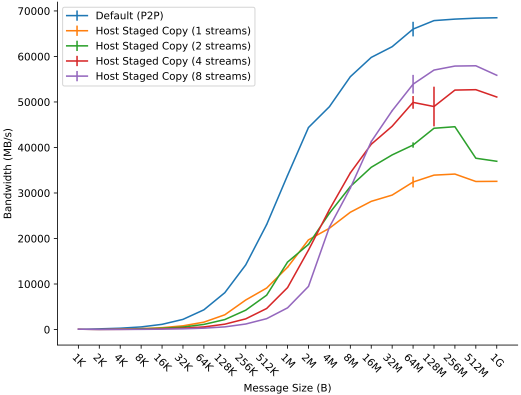
Multi-Path Copy Motivation

- We used the `ucx_perftest` micro-benchmarks to assess the viability of our design idea



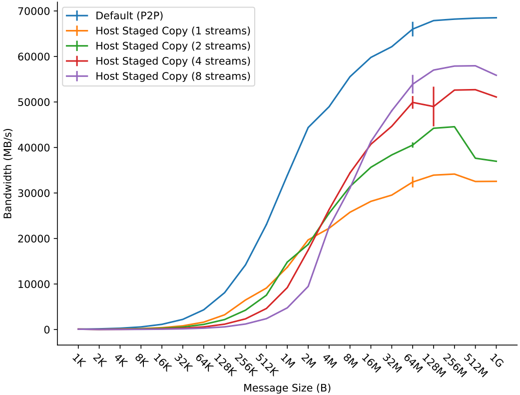
Multi-Path Copy Motivation

- We used the `ucx_perftest` micro-benchmarks to assess the viability of our design idea
- Preliminary investigation showed:



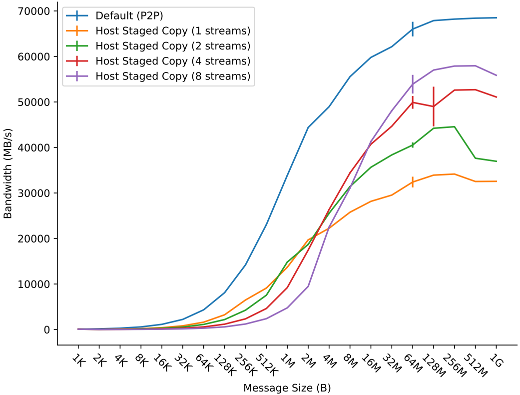
Multi-Path Copy Motivation

- We used the `ucx_perftest` micro-benchmarks to assess the viability of our design idea
- Preliminary investigation showed:
  - Stream count impacts peak bandwidth for the host-path



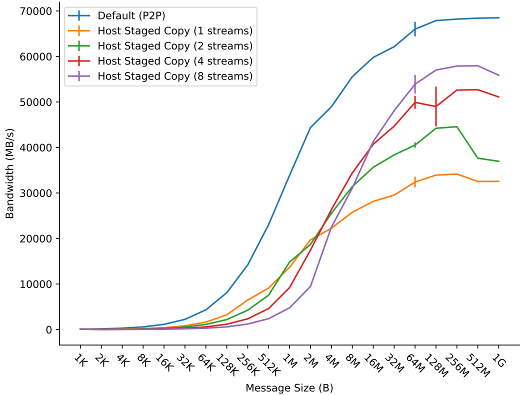
Multi-Path Copy Motivation

- We used the `ucx_perftest` micro-benchmarks to assess the viability of our design idea
- Preliminary investigation showed:
  - Stream count impacts peak bandwidth for the host-path
  - Stream count is dependent on message size



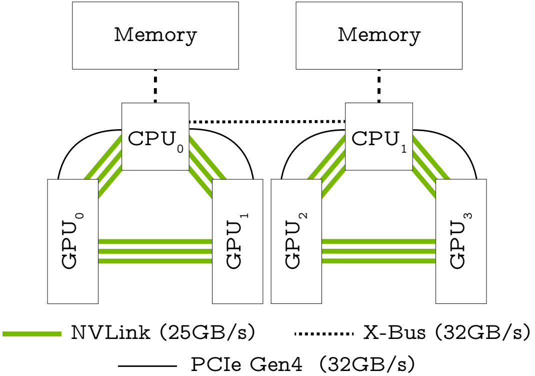
Multi-Path Copy Motivation

- We used the `ucx_perftest` micro-benchmarks to assess the viability of our design idea
- Preliminary investigation showed:
  - Stream count impacts peak bandwidth for the host-path
  - Stream count is dependent on message size
  - Up to 53GB/s of unused bandwidth



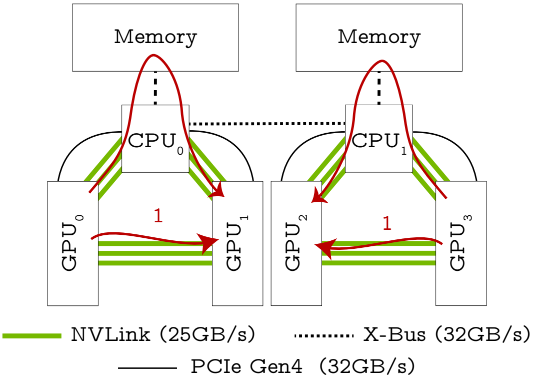
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:



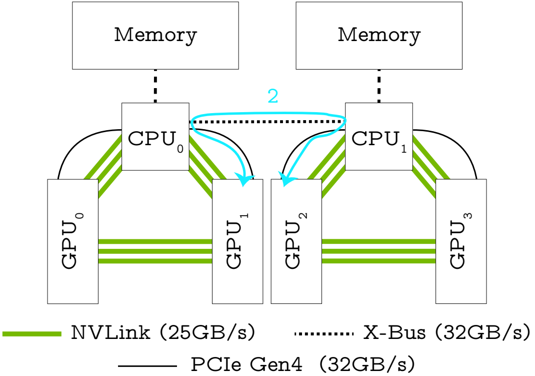
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce



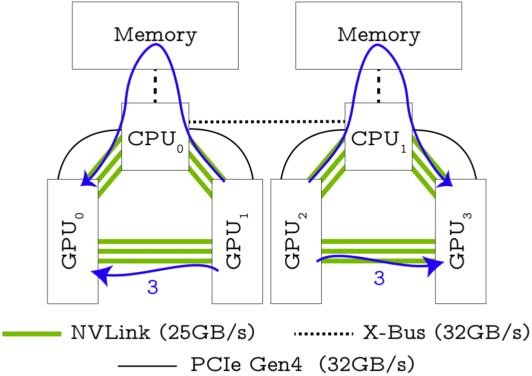
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce



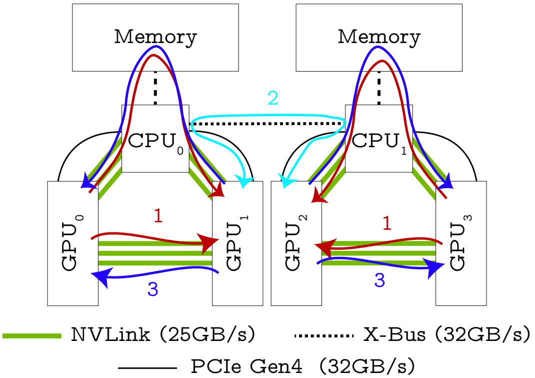
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce
  3. Intra-socket multi-path broadcast



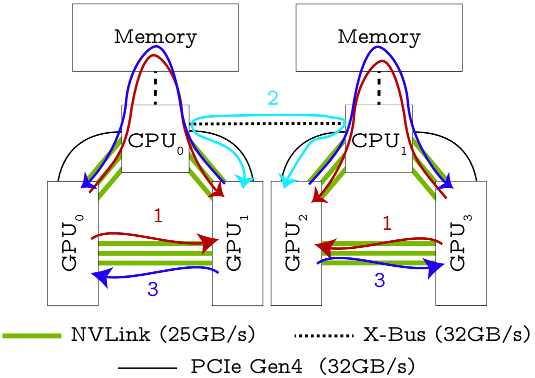
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce
  3. Intra-socket multi-path broadcast
- Design Optimisations



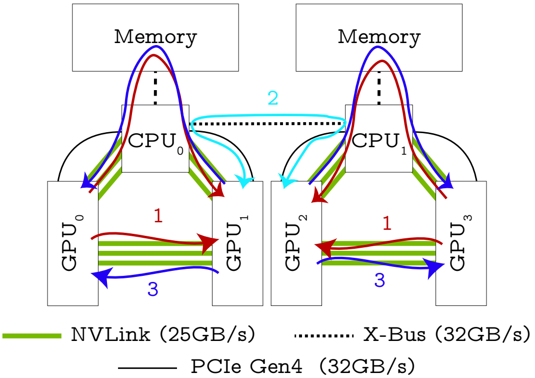
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce
  3. Intra-socket multi-path broadcast
- Design Optimisations
  - Steps 1-3 are pipelined



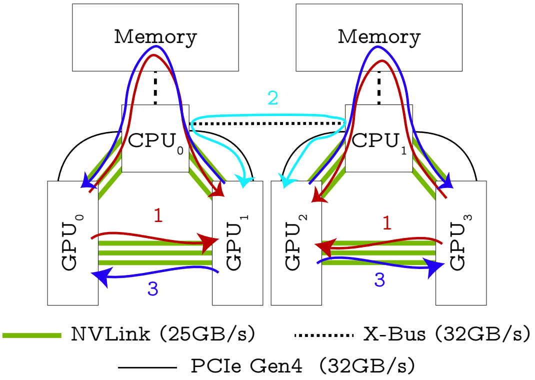
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce
  3. Intra-socket multi-path broadcast
- Design Optimisations
  - Steps 1-3 are pipelined
  - Inter-socket communication dynamically switches between PCIe and NVLink



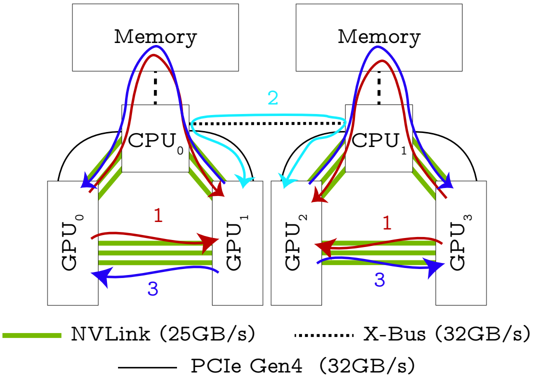
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce
  3. Intra-socket multi-path broadcast
- Design Optimisations
  - Steps 1-3 are pipelined
  - Inter-socket communication dynamically switches between PCIe and NVLink
  - Dynamically send data using Multi-path or Peer-to-Peer copies via the host links



Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:
  1. Intra-socket multi-path reduce
  2. Inter-socket leaders exchange and reduce
  3. Intra-socket multi-path broadcast
- Design Optimisations
  - Steps 1-3 are pipelined
  - Inter-socket communication dynamically switches between PCIe and NVLink
  - Dynamically send data using Multi-path or Peer-to-Peer copies via the host links
    - Minimise intra-socket congestion



Experimental Setup

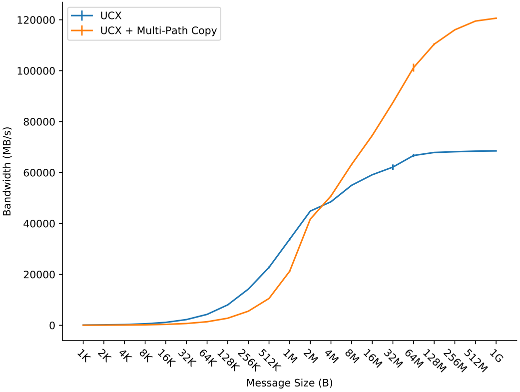
- Hardware:
  - IBM AC922
  - 32 Core, 128 Thread Power9 CPU
  - 256GB RAM
  - Four V100-SMX2-32GB

Experimental Setup

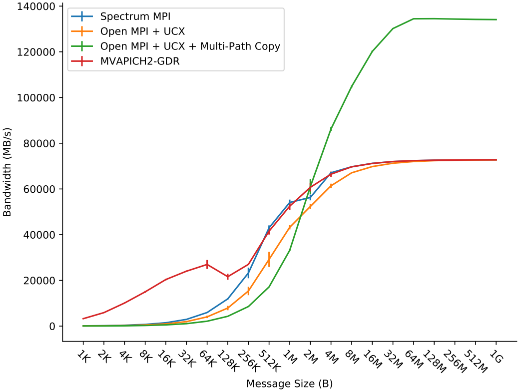
- Hardware:
  - IBM AC922
  - 32 Core, 128 Thread Power9 CPU
  - 256GB RAM
  - Four V100-SMX2-32GB
- Software:
  - Open MPI 4.0.4rc2
  - UCX 1.8.0
  - Open MPI + HPC-X v2.7
  - Spectrum-MPI 10.3.1
  - MVAPICH2-GDR 2.3.5
  - NCCL 2.5.6
  - Horovod 0.20.3
  - TensorFlow 1.15.2

ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

UCX Put and MPI Point-to-Point Results

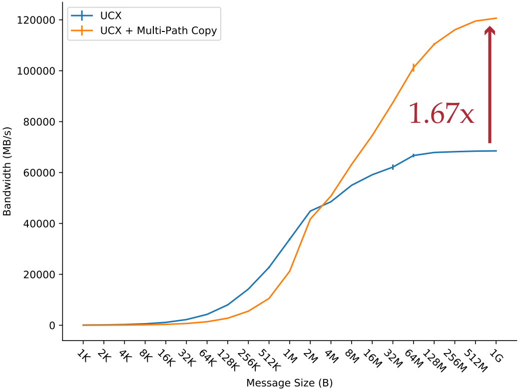


UCX Put Bandwidth

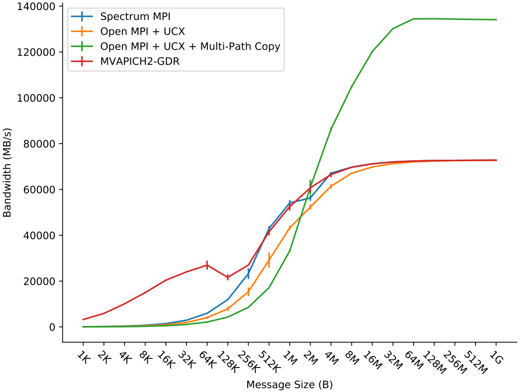


MPI Unidirectional Bandwidth

UCX Put and MPI Point-to-Point Results

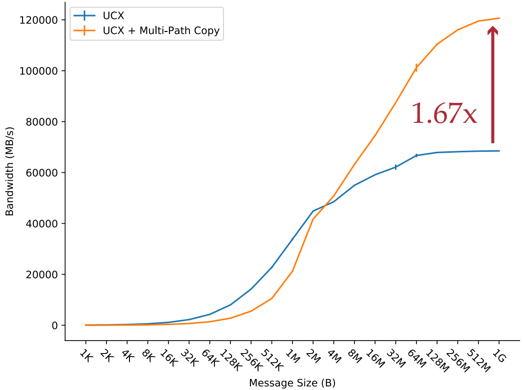


UCX Put Bandwidth

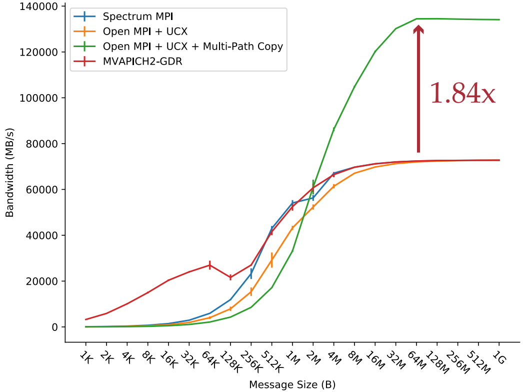


MPI Unidirectional Bandwidth

UCX Put and MPI Point-to-Point Results

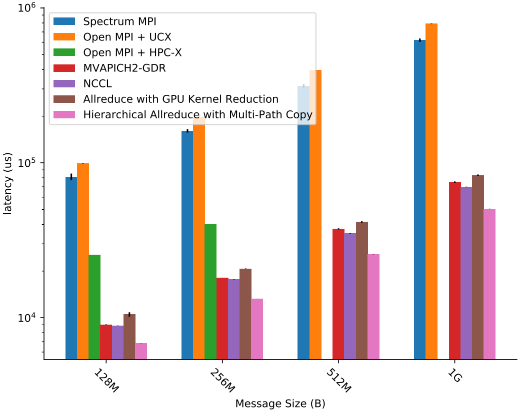


UCX Put Bandwidth



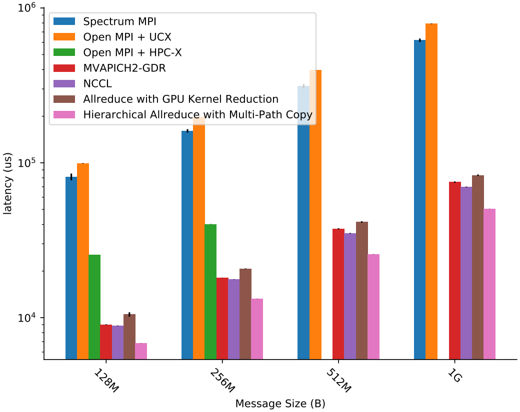
MPI Unidirectional Bandwidth

MPI_Allreduce OSU Microbenchmark Results



MPI_Allreduce latency on 4 GPUs for very large message sizes

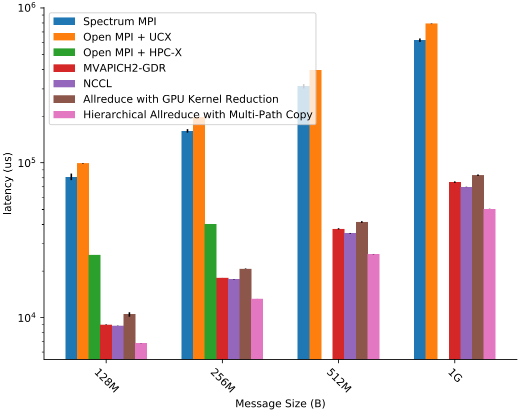
MPI_Allreduce OSU Microbenchmark Results



MPI_Allreduce latency on 4 GPUs for very large message sizes

- Much lower latency than Open MPI + HPC-X

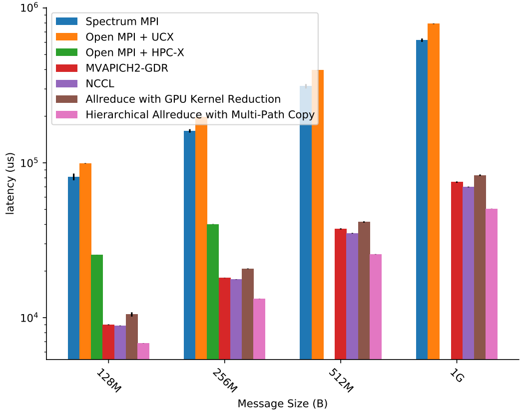
MPI_Allreduce OSU Microbenchmark Results



MPI_Allreduce latency on 4 GPUs for very large message sizes

- Much lower latency than Open MPI + HPC-X
- At 1GB we see speedup of:

`MPI_Allreduce` OSU Microbenchmark Results

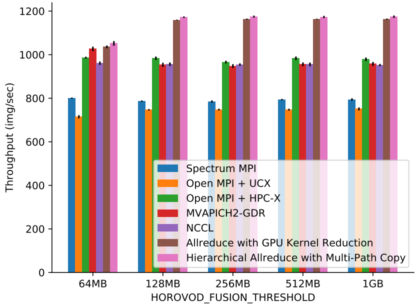


`MPI_Allreduce` latency on 4 GPUs for very large message sizes

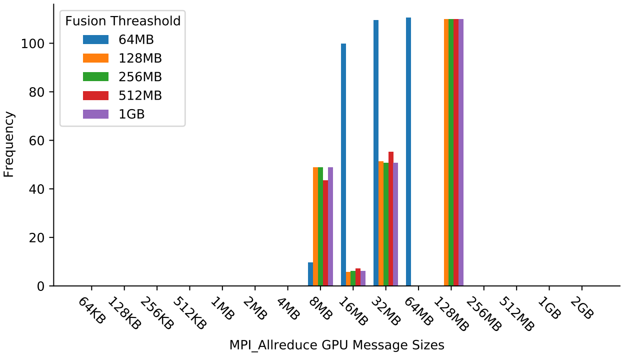
- Much lower latency than Open MPI + HPC-X
- At 1GB we see speedup of:
  - 1.47x over MVAPICH2-GDR
  - 1.38x over NCCL

Application Results

- ResNet50 up to 1.56x speedup



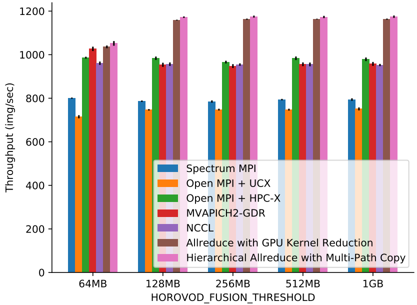
Synthetic Horovod + TensorFlow benchmarks for ResNet50



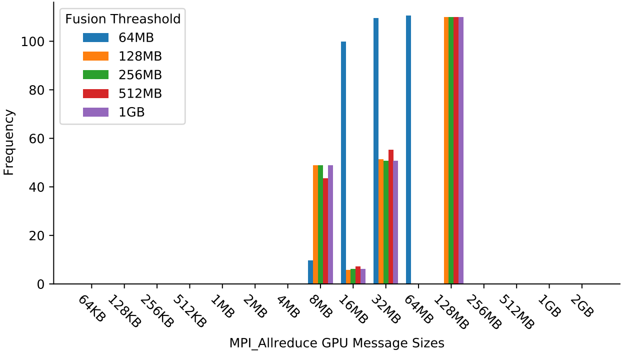
GPU Message Sizes for different `HOROVOD_FUSION_THREASHOLD`

Application Results

- ResNet50 up to 1.56x speedup
- Modifying fusion threshold increases message sizes to 128MB



Synthetic Horovod + TensorFlow benchmarks for ResNet50

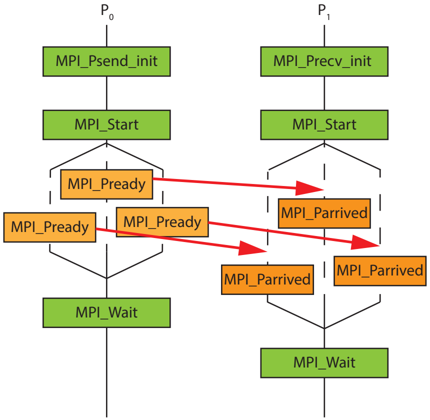


GPU Message Sizes for different `HOROVOD_FUSION_THREASHOLD`

Benchmarking for MPI-Partitioned Communication

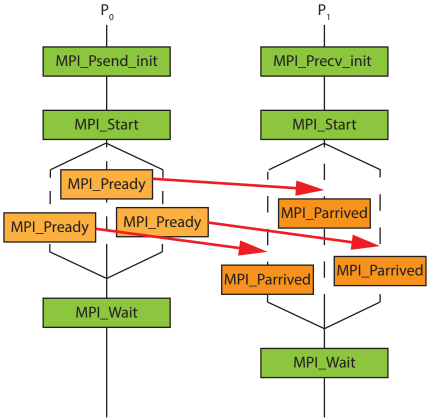
MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init/MPI_Precv_init` is used to initialize communication between processes



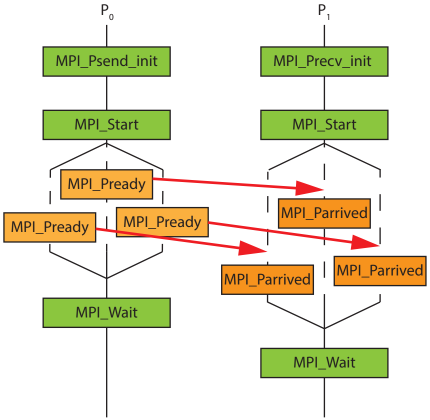
MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init/MPI_Precv_init` is used to initialize communication between processes
  - Message matching occurs here
  - MPI Partitioned does not accept wildcards



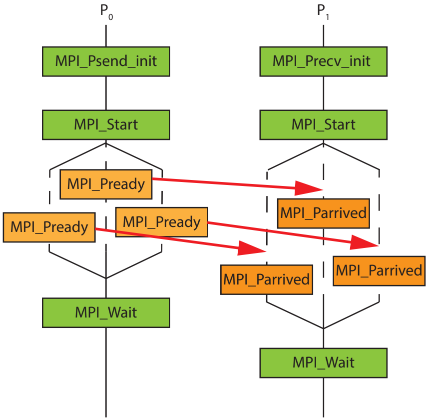
MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init`/`MPI_Precv_init` is used to initialize communication between processes
  - Message matching occurs here
  - MPI Partitioned does not accept wildcards
- `MPI_Start` is called to start communication



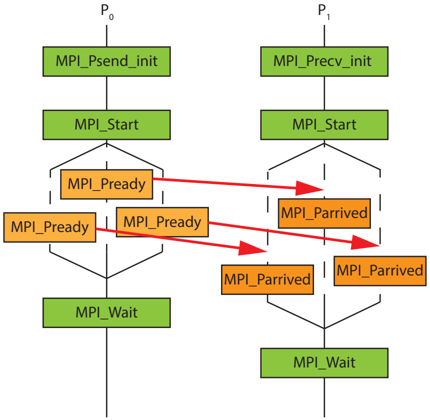
MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init/MPI_Precv_init` is used to initialize communication between processes
  - Message matching occurs here
  - MPI Partitioned does not accept wildcards
- `MPI_Start` is called to start communication
- A parallel for loop is launched



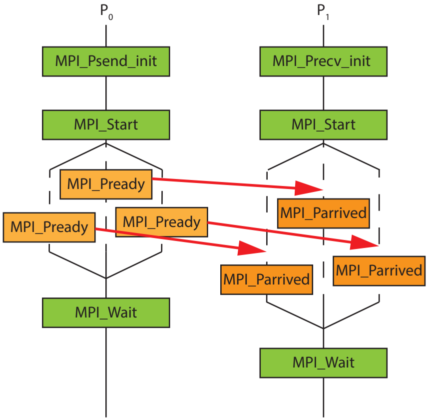
MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init/MPI_Precv_init` is used to initialize communication between processes
  - Message matching occurs here
  - MPI Partitioned does not accept wildcards
- `MPI_Start` is called to start communication
- A parallel for loop is launched
  - Work is Computed
  - Once data is ready, `MPI_Pready` is called
  - Optionally, `MPI_Parrived` to check if incoming data has arrived



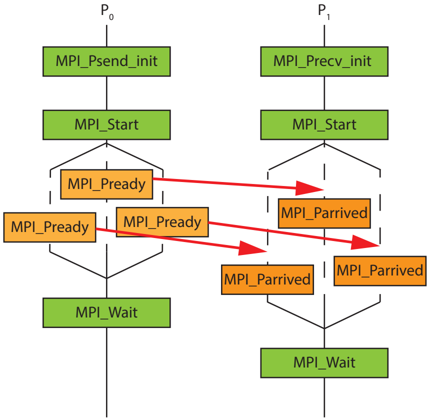
MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init/MPI_Precv_init` is used to initialize communication between processes
  - Message matching occurs here
  - MPI Partitioned does not accept wildcards
- `MPI_Start` is called to start communication
- A parallel for loop is launched
  - Work is Computed
  - Once data is ready, `MPI_Pready` is called
  - Optionally, `MPI_Parrived` to check if incoming data has arrived
- `MPI_Waitall` is called to complete communication



MPI Partitioned Point-to-Point Communication

- `MPI_Psend_init/MPI_Precv_init` is used to initialize communication between processes
  - Message matching occurs here
  - MPI Partitioned does not accept wildcards
- `MPI_Start` is called to start communication
- A parallel for loop is launched
  - Work is Computed
  - Once data is ready, `MPI_Pready` is called
  - Optionally, `MPI_Parrived` to check if incoming data has arrived
- `MPI_Waitall` is called to complete communication
- A good implementation does not have the serialization issues of MPI Point-to-Point



Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)

Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)
- Traditional point-to-point benchmarking techniques do not work for MPI Partitioned

Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)
- Traditional point-to-point benchmarking techniques do not work for MPI Partitioned
- No production application uses MPI Partitioned
  - How can we discover possible candidates for porting?

Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)
- Traditional point-to-point benchmarking techniques do not work for MPI Partitioned
- No production application uses MPI Partitioned
  - How can we discover possible candidates for porting?

Research Questions

Can we design an MPI Partitioned Micro-benchmark to address the following:

Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)
- Traditional point-to-point benchmarking techniques do not work for MPI Partitioned
- No production application uses MPI Partitioned
  - How can we discover possible candidates for porting?

Research Questions

Can we design an MPI Partitioned Micro-benchmark to address the following:

- How can we understand the behaviour and performance of MPI Partitioned?

Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)
- Traditional point-to-point benchmarking techniques do not work for MPI Partitioned
- No production application uses MPI Partitioned
  - How can we discover possible candidates for porting?

Research Questions

Can we design an MPI Partitioned Micro-benchmark to address the following:

- How can we understand the behaviour and performance of MPI Partitioned?
- How could existing applications benefit from this new programming model?

Motivation

- Commonly used benchmarks do not support MPI Partitioned
  - Sandia Micro Benchmarks (SMB)
  - OSU Micro Benchmarks (OSU)
  - Intel MPI Benchmarks (IMB)
- Traditional point-to-point benchmarking techniques do not work for MPI Partitioned
- No production application uses MPI Partitioned
  - How can we discover possible candidates for porting?

Research Questions

Can we design an MPI Partitioned Micro-benchmark to address the following:

- How can we understand the behaviour and performance of MPI Partitioned?
- How could existing applications benefit from this new programming model?
- What are appropriate partition sizes for application developers to use?

Experiment Setup

- Niagara Supercomputer at SciNet[1]
  - 2x 20 Core Intel Skylake at 2.4GHz
  - EDR InfiniBand Network
  - GNU/Linux - CentOS 7.6
  - Open MPI (master branch)
  - UCX v1.11.0
  - MPIPCL

[1]SciNet is funded by: the Canada Foundation for Innovation; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto This research was enabled in part by support provided by the Digital Research Alliance of Canada

Overhead

- What is the cost of using MPI Partitioned?

Overhead

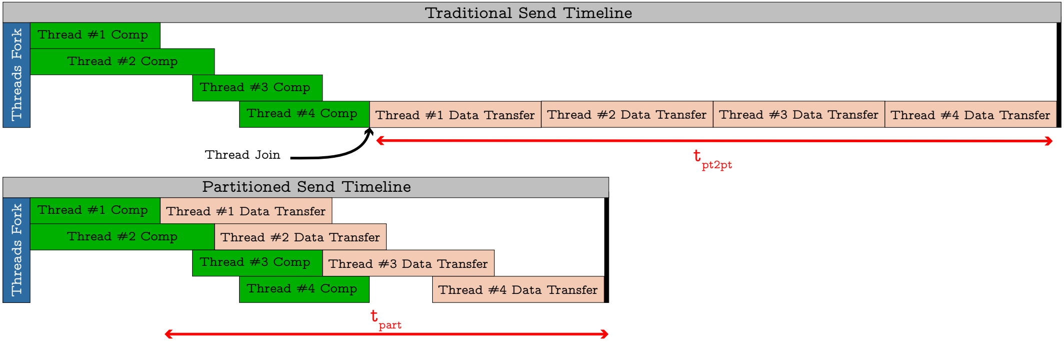
- What is the cost of using MPI Partitioned?
  - We measure each individual data transfer
  - Compare it to MPI Point-to-Point

$$Overhead = \frac{t_{part}}{t_{pt2pt}}$$

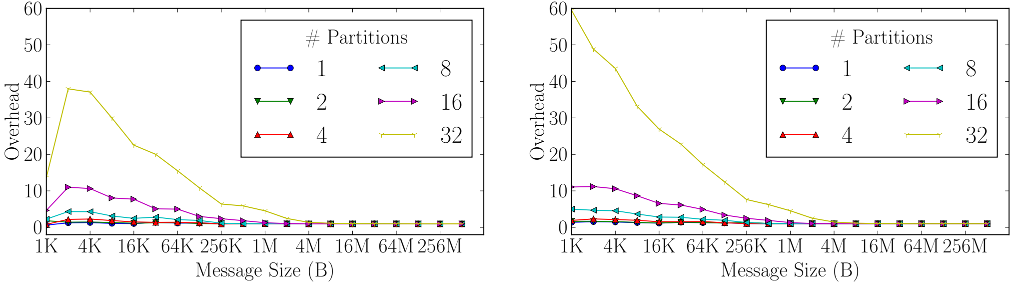
Overhead

- What is the cost of using MPI Partitioned?
  - We measure each individual data transfer
  - Compare it to MPI Point-to-Point

$$Overhead = \frac{t_{part}}{t_{pt2pt}}$$



Overhead Results



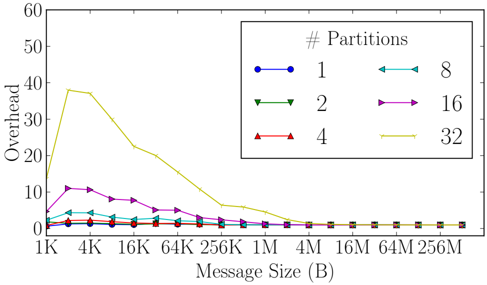
(a) Cold Cache

(b) Hot Cache

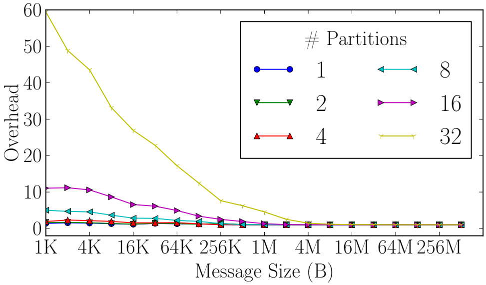
Overhead of Partitioned Point-to-Point Communication Relative to Point-to-Point Communication for 10ms of Compute

Overhead Results

- Partition count correlates with overhead



(a) Cold Cache

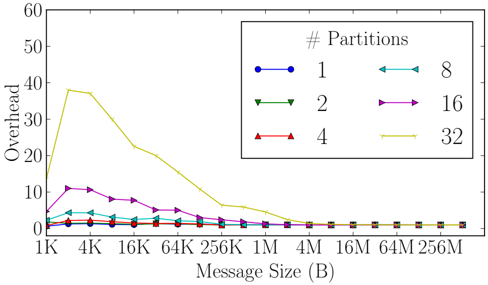


(b) Hot Cache

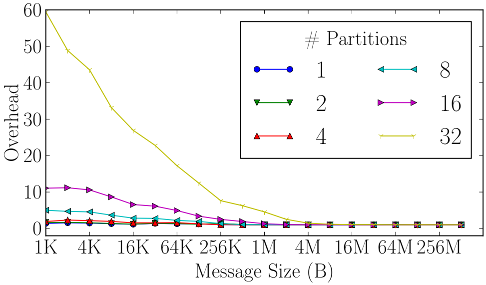
Overhead of Partitioned Point-to-Point Communication Relative to Point-to-Point Communication for 10ms of Compute

Overhead Results

- Partition count correlates with overhead
- Overheads mostly impact small messages



(a) Cold Cache



(b) Hot Cache

Overhead of Partitioned Point-to-Point Communication Relative to Point-to-Point Communication for 10ms of Compute

Perceived Bandwidth

- What would be the required network bandwidth for MPI Point-to-Point to perform the same as MPI Partitioned?

Perceived Bandwidth

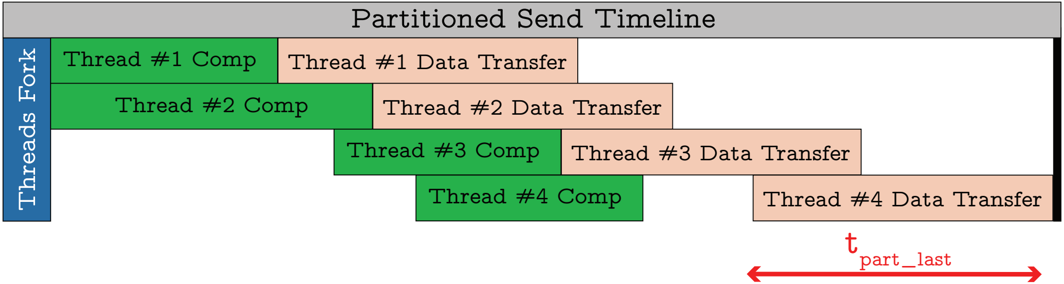
- What would be the required network bandwidth for MPI Point-to-Point to perform the same as MPI Partitioned?

$$Perceived\ Bandwidth = \frac{m}{t_{part\_last}}$$

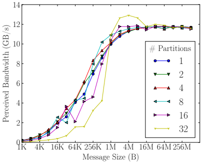
Perceived Bandwidth

- What would be the required network bandwidth for MPI Point-to-Point to perform the same as MPI Partitioned?

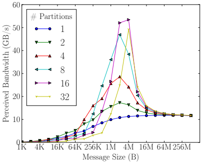
$$Perceived\ Bandwidth = \frac{m}{t_{part\_last}}$$



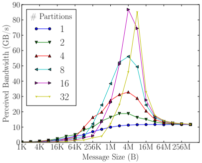
Perceived Bandwidth Results



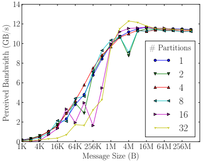
(a) 10ms Comp with 0% Noise



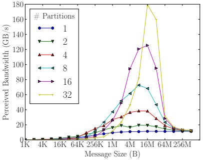
(b) 10ms Comp with 4% Noise



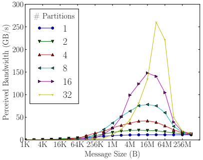
(c) 10ms Comp with 10% Noise



(d) 100ms Comp with 0% Noise



(e) 100ms Comp with 4% Noise

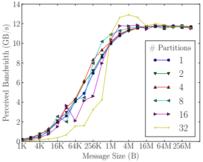


(f) 100ms Comp with 10% Noise

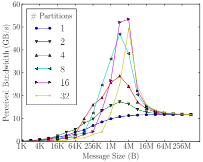
Perceived Bandwidth of MPI Partitioned Point-to-Point Communication with Uniform Noise and a Hot Cache for Different Noise and Compute Amounts

Perceived Bandwidth Results

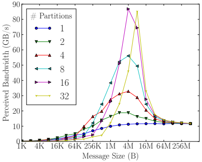
- With 0% noise, we see our traditional bandwidth curve



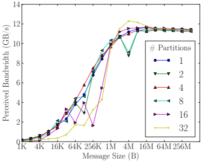
(a) 10ms Comp with 0% Noise



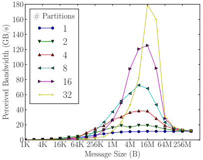
(b) 10ms Comp with 4% Noise



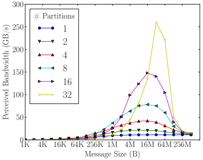
(c) 10ms Comp with 10% Noise



(d) 100ms Comp with 0% Noise



(e) 100ms Comp with 4% Noise

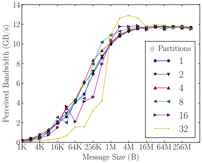


(f) 100ms Comp with 10% Noise

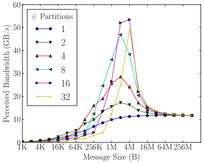
Perceived Bandwidth of MPI Partitioned Point-to-Point Communication with Uniform Noise and a Hot Cache for Different Noise and Compute Amounts

Perceived Bandwidth Results

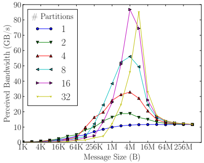
- With 0% noise, we see our traditional bandwidth curve
- Peak bandwidth is obtained for medium sized messages



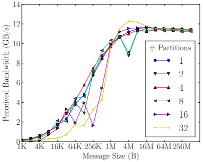
(a) 10ms Comp with 0% Noise



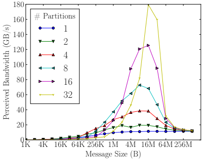
(b) 10ms Comp with 4% Noise



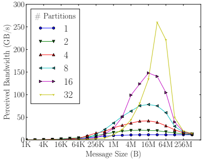
(c) 10ms Comp with 10% Noise



(d) 100ms Comp with 0% Noise



(e) 100ms Comp with 4% Noise

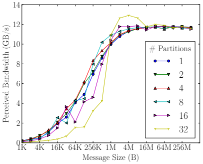


(f) 100ms Comp with 10% Noise

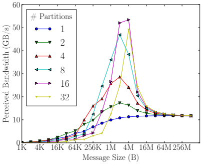
Perceived Bandwidth of MPI Partitioned Point-to-Point Communication with Uniform Noise and a Hot Cache for Different Noise and Compute Amounts

Perceived Bandwidth Results

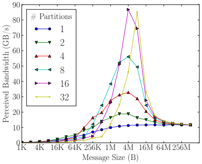
- With 0% noise, we see our traditional bandwidth curve
- Peak bandwidth is obtained for medium sized messages
- Actual network bandwidth is saturated for large messages, thus perceived bandwidth drops



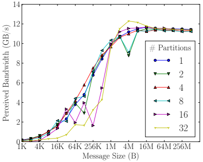
(a) 10ms Comp with 0% Noise



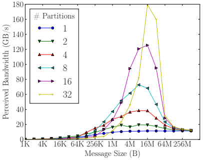
(b) 10ms Comp with 4% Noise



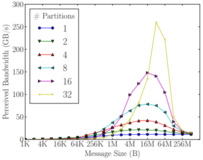
(c) 10ms Comp with 10% Noise



(d) 100ms Comp with 0% Noise



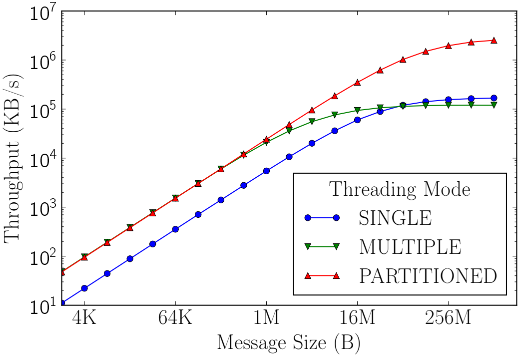
(e) 100ms Comp with 4% Noise



(f) 100ms Comp with 10% Noise

Perceived Bandwidth of MPI Partitioned Point-to-Point Communication with Uniform Noise and a Hot Cache for Different Noise and Compute Amounts

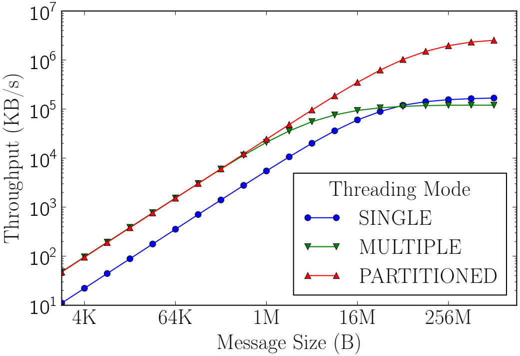
Sweep3D Communication Pattern



Sweep3D communication throughput for 16 partitions, 10ms compute, and 4% Single Noise with a Hot Cache

Sweep3D Communication Pattern

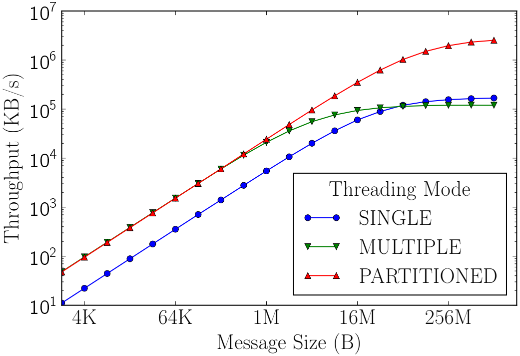
- Sweep3D communication pattern has lots of dependencies



Sweep3D communication throughput for 16 partitions, 10ms compute, and 4% Single Noise with a Hot Cache

Sweep3D Communication Pattern

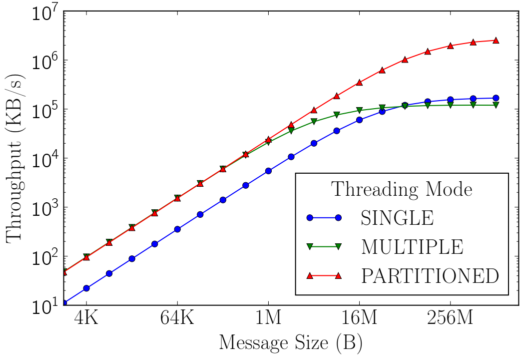
- Sweep3D communication pattern has lots of dependencies
- Generally, multi-threading performs better than single threaded



Sweep3D communication throughput for 16 partitions, 10ms compute, and 4% Single Noise with a Hot Cache

Sweep3D Communication Pattern

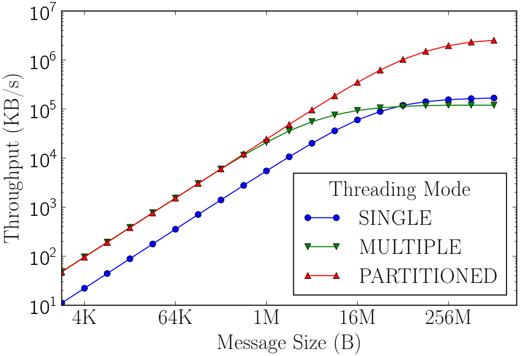
- Sweep3D communication pattern has lots of dependencies
- Generally, multi-threading performs better than single threaded
- The MPI Partition implementation used in this work is built upon MPI Send/Recv



Sweep3D communication throughput for 16 partitions, 10ms compute, and 4% Single Noise with a Hot Cache

Sweep3D Communication Pattern

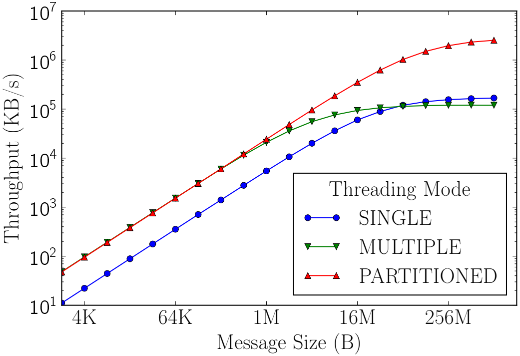
- Sweep3D communication pattern has lots of dependencies
- Generally, multi-threading performs better than single threaded
- The MPI Partition implementation used in this work is built upon MPI Send/Recv
  - Therefore minimal difference for most message sizes



Sweep3D communication throughput for 16 partitions, 10ms compute, and 4% Single Noise with a Hot Cache

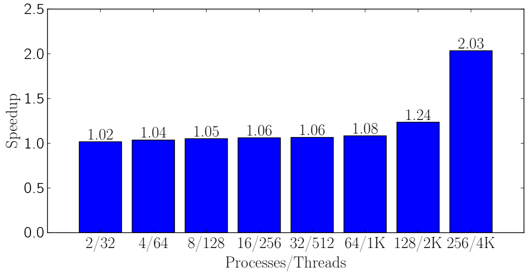
Sweep3D Communication Pattern

- Sweep3D communication pattern has lots of dependencies
- Generally, multi-threading performs better than single threaded
- The MPI Partition implementation used in this work is built upon MPI Send/Recv
  - Therefore minimal difference for most message sizes
- Up to 15.1x higher throughput for large message sizes



Sweep3D communication throughput for 16 partitions, 10ms compute, and 4% Single Noise with a Hot Cache

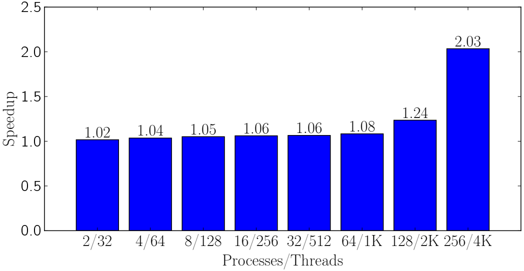
Potential Application Improvements



Expected Speedup From Porting SNAP-C to MPI Partitioned

Potential Application Improvements

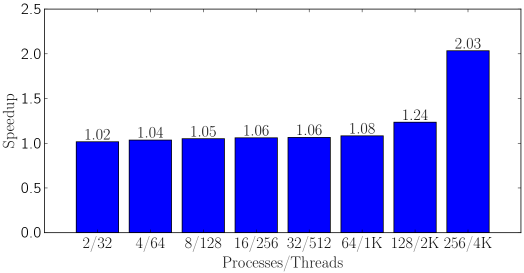
- The Sweep3D communication pattern showed potential for if it were ported to MPI Partitioned



Expected Speedup From Porting SNAP-C to MPI Partitioned

Potential Application Improvements

- The Sweep3D communication pattern showed potential for if it were ported to MPI Partitioned
- SNAP uses a Sweep3D communication
  - We profiled SNAP's communication
  - Projected the potential speedup



Expected Speedup From Porting SNAP-C to MPI Partitioned

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite
  - Showed what communication patterns could benefit from MPI Partitioned

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite
  - Showed what communication patterns could benefit from MPI Partitioned
  - Analyzed MPI Partitioned with a range of different metrics

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite
  - Showed what communication patterns could benefit from MPI Partitioned
  - Analyzed MPI Partitioned with a range of different metrics

Future Work

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite
  - Showed what communication patterns could benefit from MPI Partitioned
  - Analyzed MPI Partitioned with a range of different metrics

Future Work

- Compare across different MPI implementations

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite
  - Showed what communication patterns could benefit from MPI Partitioned
  - Analyzed MPI Partitioned with a range of different metrics

Future Work

- Compare across different MPI implementations
- Porting Applications to MPI Partitioned based upon benchmarking results

Conclusion And Future Work

- Benchmarking for MPI-Based Deep Learning
  - Deep Learning workloads use `MPI_Allreduce` collective with large messages extensively
  - We proposed an intra-socket multi-path point-to-point MPI collective
    - Evaluated with different benchmarks for each layer of the software stack
    - For Deep Learning applications we see up to 3.42x speedup
- Benchmarking for MPI-Partitioned Communication
  - We provide the first MPI Partitioned Micro-Benchmark Suite
  - Showed what communication patterns could benefit from MPI Partitioned
  - Analyzed MPI Partitioned with a range of different metrics

Future Work

- Compare across different MPI implementations
- Porting Applications to MPI Partitioned based upon benchmarking results
- MPI Partitioned Collectives

Thank You!

Acknowledgements

Queen's UNIVERSITY

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

NSERC
CRSNG

Digital Research Alliance of Canada | Alliance de recherche numérique du Canada

Y. H. Temuçin, A. Sojoodi, P. Alizadeh, and A. Afsahi, "Efficient Multi-Path NVLink/PCIe-Aware UCX based Collective Communication for Deep Learning," in *2021 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2021, pp. 25–34.

Y. H. Temuçin, A. H. Sojoodi, P. Alizadeh, B. Kitor, and A. Afsahi, "Accelerating Deep Learning Using Interconnect-Aware UCX Communication for MPI Collectives," *IEEE Micro*, vol. 42, no. 2, pp. 68–76, 2022.

Y. H. Temucin, R. E. Grant, and A. Afsahi, "Micro-Benchmarking MPI Partitioned Point-to-Point Communication," in *Proceedings of the 51st International Conference on Parallel Processing*, ser. ICPP '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3545008.3545088