Efficient Multi-Path NVLink/PCIe-Aware UCX based Collective Communication for Deep Learning

Yıltan Hassan Temuçin, AmirHossein Sojoodi, Pedram Alizadeh, and Ahmad Afsahi

Parallel Processing Research Lab (PPRL)
Department of Electrical and Computer Engineering
Queen's University, Canada

28th IEEE Hot Interconnects symposium (HotI 2021)
August 18th 2021

Outline

Introduction

- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
  - The popularity of heterogeneous systems with accelerators is continually growing

Introduction

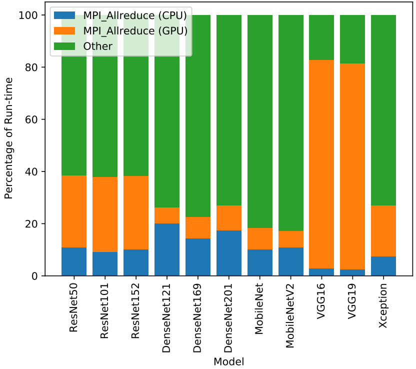
- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
  - The popularity of heterogeneous systems with accelerators is continually growing
- The Message Passing Interface (MPI)
  - Popular parallel programming model in HPC
  - Provides 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
  - The popularity of heterogeneous systems with accelerators is continually growing
- 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

MPI-based Deep Learning

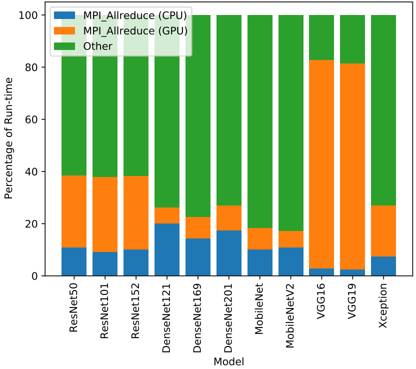
- Distributed Deep Learning Using Horovod is one of the most popular training method.



Impact of MPI_Allreduce on a single AC922 node

MPI-based Deep Learning

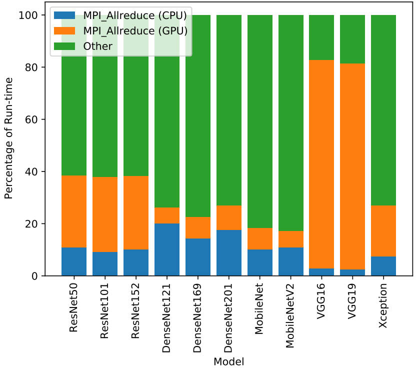
- Distributed Deep Learning Using Horovod is one of the most popular training method.
- Horovod is compatible with multiple Deep Learning frameworks:
  - TensorFlow
  - PyTorch
  - MXNet



Impact of MPI_Allreduce on a single AC922 node

MPI-based Deep Learning

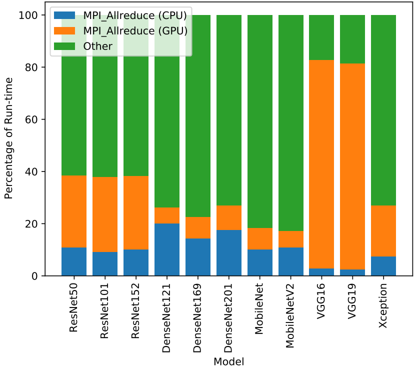
- Distributed Deep Learning Using Horovod is one of the most popular training method.
- Horovod is compatible with multiple Deep Learning frameworks:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`



Impact of MPI_Allreduce on a single AC922 node

MPI-based Deep Learning

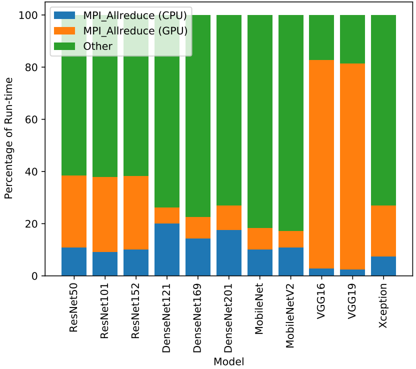
- Distributed Deep Learning Using Horovod is one of the most popular training method.
- Horovod is compatible with multiple Deep Learning frameworks:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`
  - 17-83% of training time was spent in `MPI_Allreduce`



Impact of MPI_Allreduce on a single AC922 node

MPI-based Deep Learning

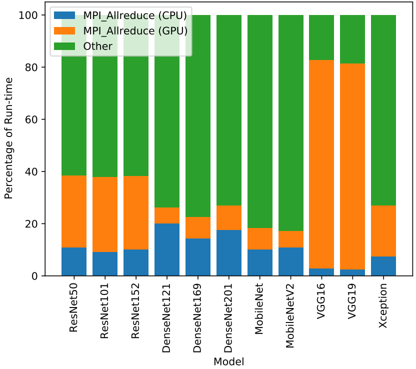
- Distributed Deep Learning Using Horovod is one of the most popular training method.
- Horovod is compatible with multiple Deep Learning frameworks:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`
  - 17-83% of training time was spent in `MPI_Allreduce`
  - Up to 80% of runtime was spent in a GPU based `MPI_Allreduce`.



Impact of MPI_Allreduce on a single AC922 node

MPI-based Deep Learning

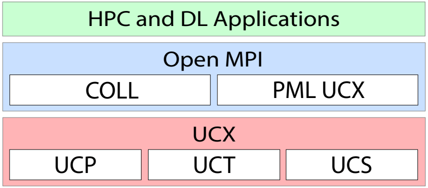
- Distributed Deep Learning Using Horovod is one of the most popular training method.
- Horovod is compatible with multiple Deep Learning frameworks:
  - TensorFlow
  - PyTorch
  - MXNet
- Horovod uses the data-parallel training method using `MPI_Allreduce`
  - 17-83% of training time was spent in `MPI_Allreduce`
  - Up to 80% of runtime was spent in a GPU based `MPI_Allreduce`.
- MPI communication is a major bottleneck in these applications



Impact of MPI_Allreduce on a single AC922 node

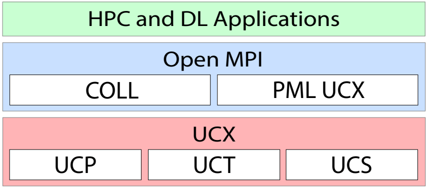
Open MPI + UCX

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



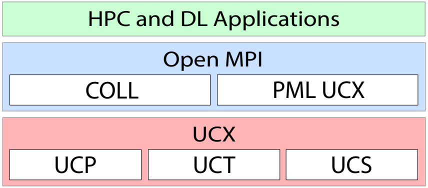
Open MPI + UCX

- **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.

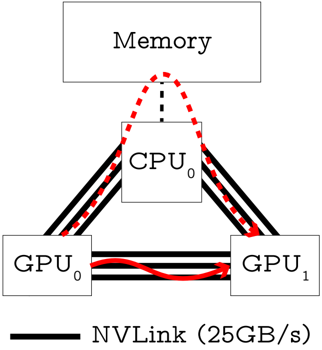


Research Goal

- Improve the performance of Deep Learning applications
- Enhance point-to-point and collective communication for MPI's large GPU messages

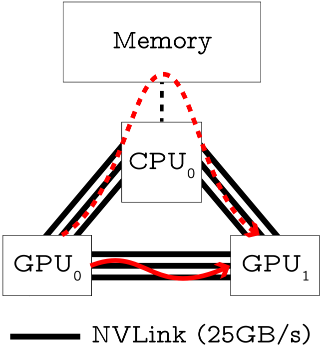
Multi-Path Copy Motivation

- Currently data transfers between GPUs, send the data directly between devices using a zero copy put operation in UCX.
  - (As shown by the solid red line)



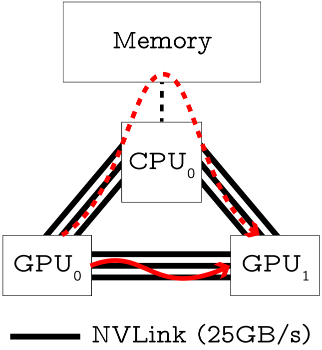
Multi-Path Copy Motivation

- Currently data transfers between GPUs, send the data directly between devices using a zero copy put operation in UCX.
  - (As shown by the solid red line)
- Results in six NVLinks connected to the host to be idle in this transfer.



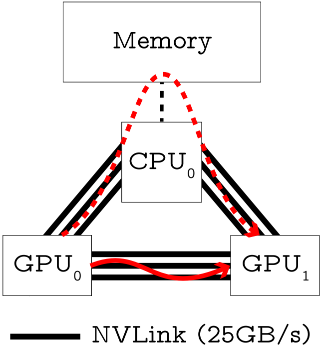
Multi-Path Copy Motivation

- Currently data transfers between GPUs, send the data directly between devices using a zero copy put operation in UCX.
  - (As shown by the solid red line)
- Results in six NVLinks connected to the host to be idle in this transfer.
- A large amount of unused potential bandwidth



Multi-Path Copy Motivation

- Currently data transfers between GPUs, send the data directly between devices using a zero copy put operation in UCX.
  - (As shown by the solid red line)
- Results in six NVLinks connected to the host to be idle in this transfer.
- A large amount of unused potential bandwidth

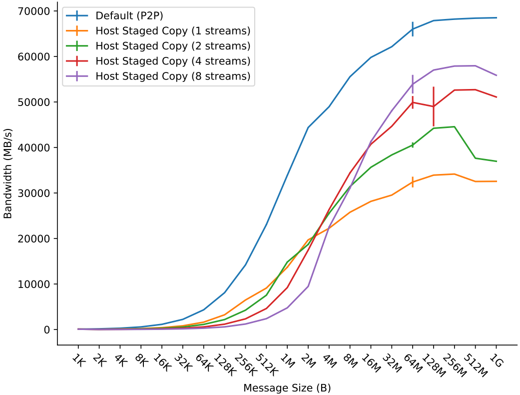


Research Question

Can we design a mechanism to use all communication paths?

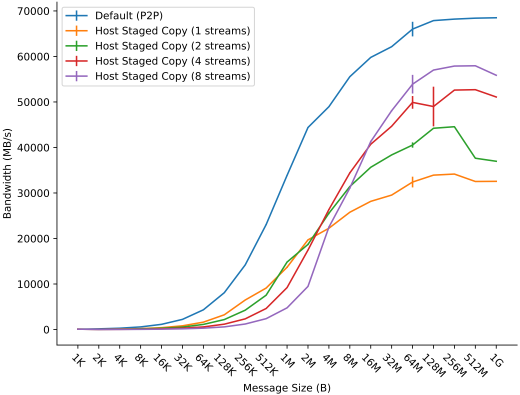
Multi-Path Copy Motivation

- Preliminary investigation showed:



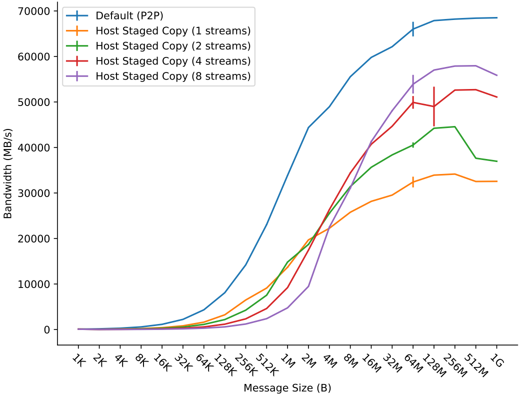
Multi-Path Copy Motivation

- Preliminary investigation showed:
  - Stream count impacts peak bandwidth for the host-path



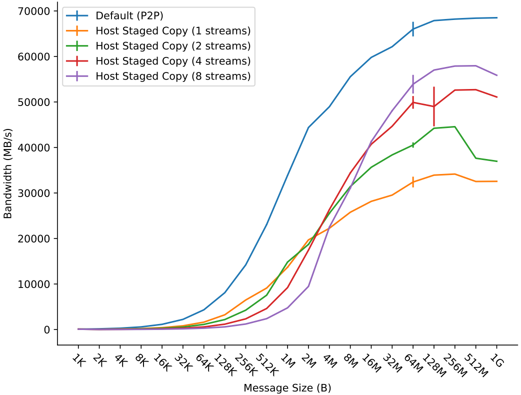
Multi-Path Copy Motivation

- Preliminary investigation showed:
  - Stream count impacts peak bandwidth for the host-path
  - Stream count is dependent on message size



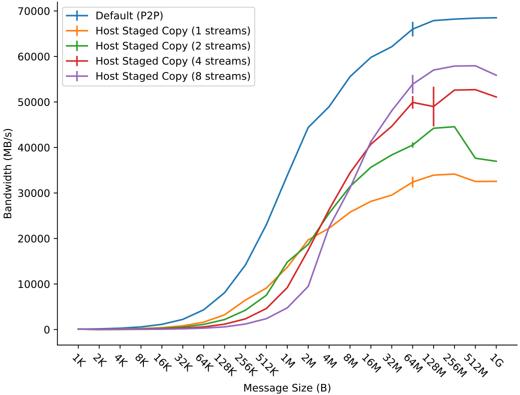
Multi-Path Copy Motivation

- 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



Multi-Path Copy Motivation

- 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
- Good potential for our Multi-Path design



Multi-Path Copy Design

Algorithm: Multi-Path Copy Algorithm

Input: sbuf, host_buf, data_size, host_share, n_host_streams
Output: dbuf

```
host_dsize = data_size * host_share;
host_chunk_dsize = host_dsize / n_host_streams;
d2d_dsize = data_size - host_dsize;
do in parallel
    Copy d2d_dsize bytes from sbuf to dbuf;
    for i ← 0 to n_host_streams by 1 do in parallel
        Copy host_chunk_size bytes from sbuf to host_buf[i];
        Wait for data in host_buf[i];
        Copy host_chunk_size bytes from host_buf[i] to
          dbuf;
    end
end
```

Multi-Path Copy Design

Algorithm: Multi-Path Copy Algorithm

Input: sbuf, host_buf, data_size, host_share, n_host_streams
Output: dbuf

```
host_dsize = data_size * host_share;
host_chunk_dsize = host_dsize / n_host_streams;
d2d_dsize = data_size - host_dsize;
do in parallel
    Copy d2d_dsize bytes from sbuf to dbuf;
    for i ← 0 to n_host_streams by 1 do in parallel
        Copy host_chunk_size bytes from sbuf to host_buf[i];
        Wait for data in host_buf[i];
        Copy host_chunk_size bytes from host_buf[i] to
          dbuf;
    end
end
```

- Tune copy parameters

Multi-Path Copy Design

Algorithm: Multi-Path Copy Algorithm

Input: sbuf, host_buf, data_size, host_share, n_host_streams
Output: dbuf

```
host_dsize = data_size * host_share;
host_chunk_dsize = host_dsize / n_host_streams;
d2d_dsize = data_size - host_dsize;
do in parallel
    Copy d2d_dsize bytes from sbuf to dbuf;
    for i ← 0 to n_host_streams by 1 do in parallel
        Copy host_chunk_size bytes from sbuf to host_buf[i];
        Wait for data in host_buf[i];
        Copy host_chunk_size bytes from host_buf[i] to
          dbuf;
    end
end
```

- Tune copy parameters
- Initiate parallel copy

Multi-Path Copy Design

Algorithm: Multi-Path Copy Algorithm

Input: sbuf, host_buf, data_size, host_share, n_host_streams
Output: dbuf

```
host_dsize = data_size * host_share;
host_chunk_dsize = host_dsize / n_host_streams;
d2d_dsize = data_size - host_dsize;
do in parallel
    Copy d2d_dsize bytes from sbuf to dbuf;
    for i ← 0 to n_host_streams by 1 do in parallel
        Copy host_chunk_size bytes from sbuf to host_buf[i];
        Wait for data in host_buf[i];
        Copy host_chunk_size bytes from host_buf[i] to
          dbuf;
    end
end
```

- Tune copy parameters
- Initiate parallel copy
  - Copy portion directly to destination GPU

Multi-Path Copy Design

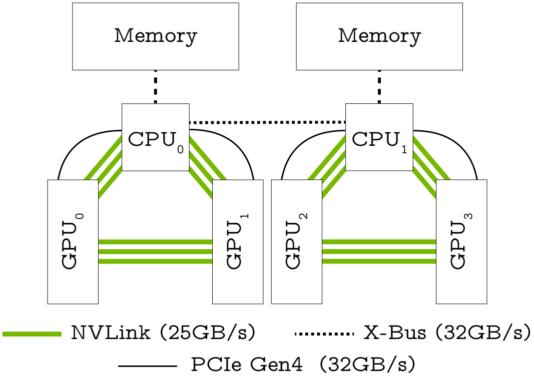
Algorithm: Multi-Path Copy Algorithm

```
Input: sbuf, host_buf, data_size, host_share, n_host_streams
Output: dbuf
host_dsize = data_size * host_share;
host_chunk_dsize = host_dsize / n_host_streams;
d2d_dsize = data_size - host_dsize;
do in parallel
    Copy d2d_dsize bytes from sbuf to dbuf;
    for i ← 0 to n_host_streams by 1 do in parallel
        Copy host_chunk_size bytes from sbuf to host_buf[i];
        Wait for data in host_buf[i];
        Copy host_chunk_size bytes from host_buf[i] to
          dbuf;
    end
end
```

- Tune copy parameters
- Initiate parallel copy
  - Copy portion directly to destination GPU
  - Copy portion via the host

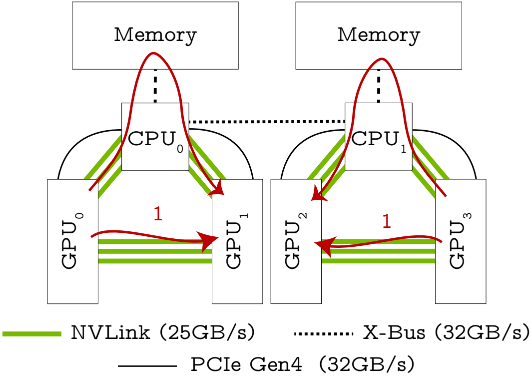
Hierarchical Allreduce with Multi-Path Copy

- The proposed `MPI_Allreduce` algorithm has three steps:



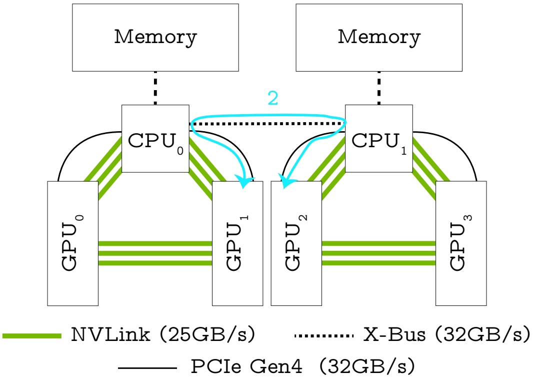
Hierarchical Allreduce with Multi-Path Copy

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



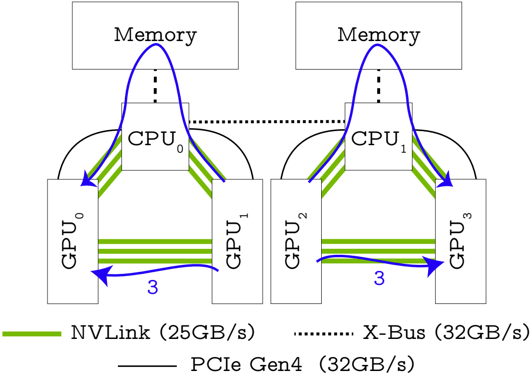
Hierarchical Allreduce with Multi-Path Copy

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



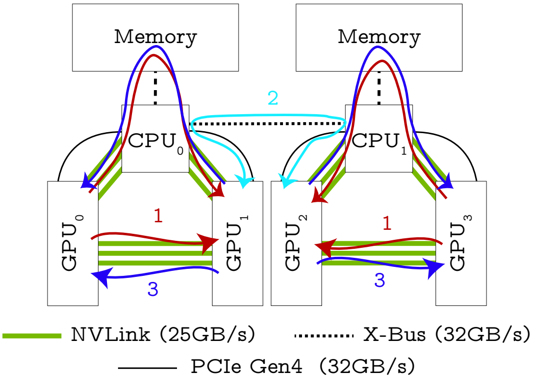
Hierarchical Allreduce with Multi-Path Copy

- 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



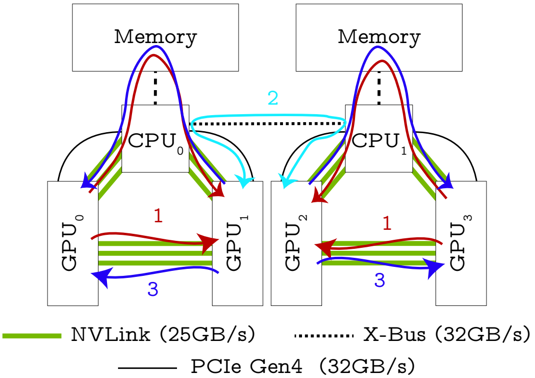
Hierarchical Allreduce with Multi-Path Copy

- 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



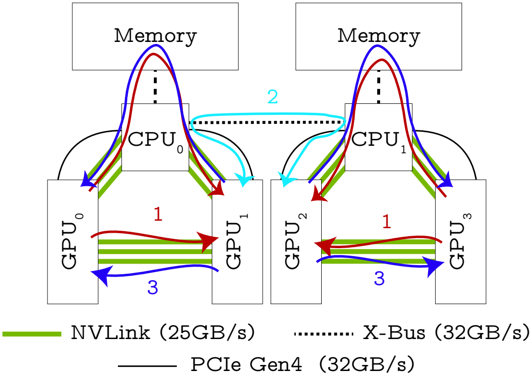
Hierarchical Allreduce with Multi-Path Copy

- 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



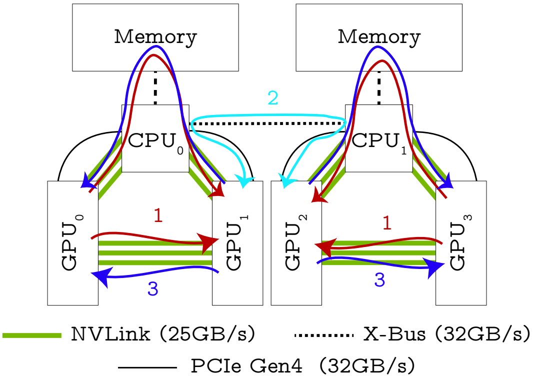
Hierarchical Allreduce with Multi-Path Copy

- 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



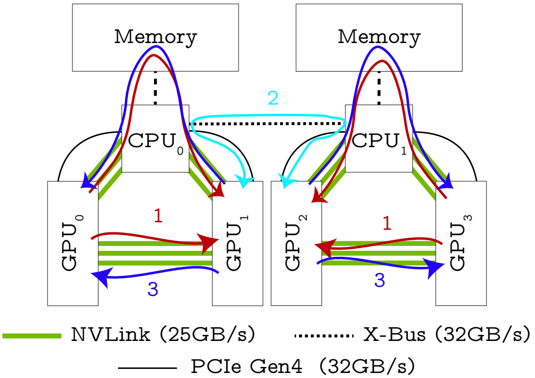
Hierarchical Allreduce with Multi-Path Copy

- 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



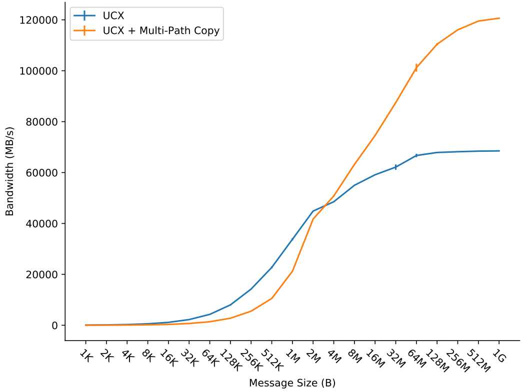
Experimental Setup

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

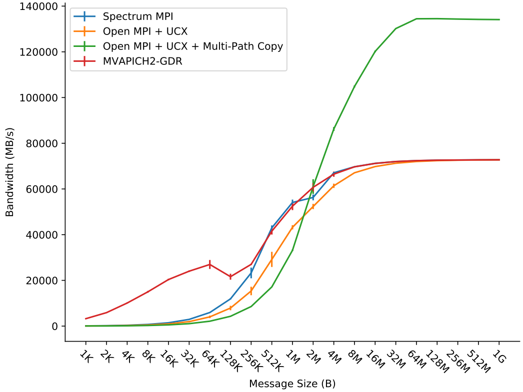
Experimental Setup

- 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

UCX Put and MPI Point-to-Point Results

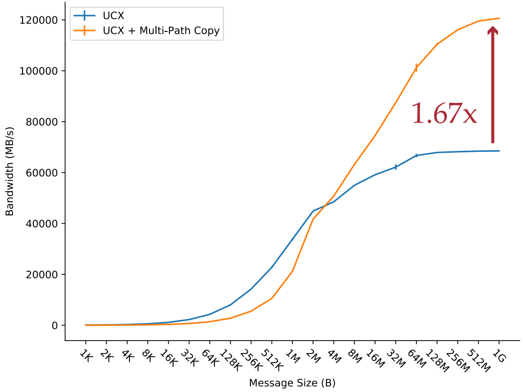


UCX Put Bandwidth

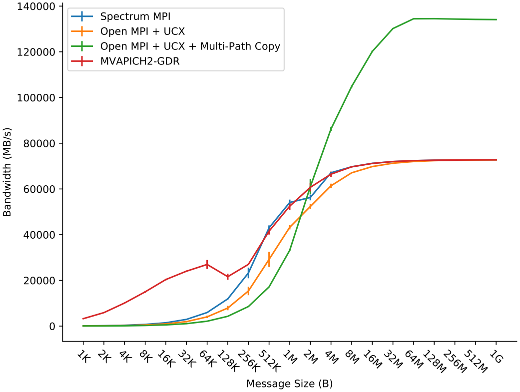


MPI Unidirectional Bandwidth

UCX Put and MPI Point-to-Point Results

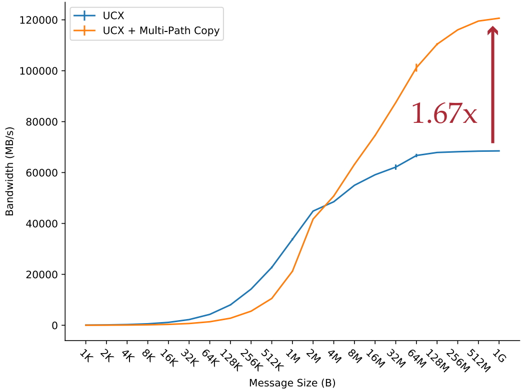


UCX Put Bandwidth

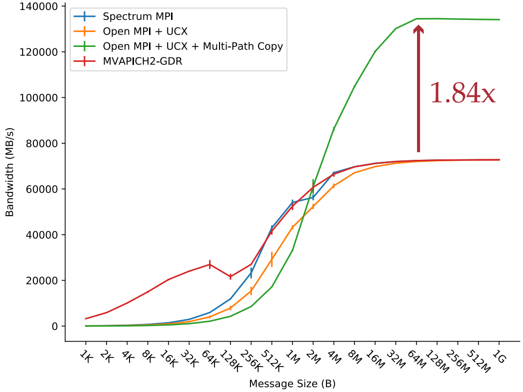


MPI Unidirectional Bandwidth

UCX Put and MPI Point-to-Point Results

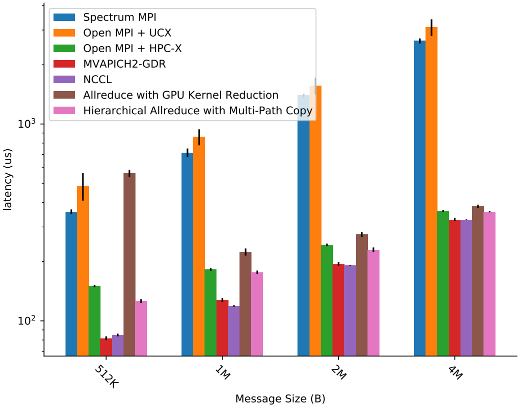


UCX Put Bandwidth



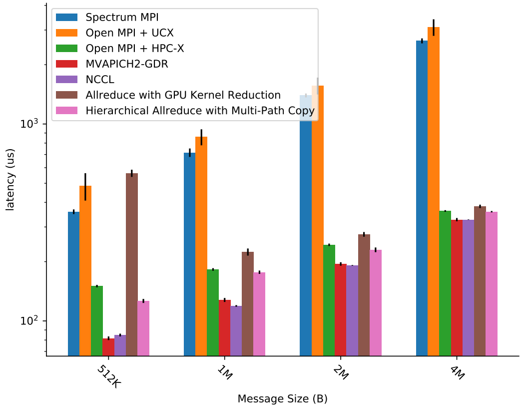
MPI Unidirectional Bandwidth

MPI_Allreduce Results (Medium Message Sizes)



MPI_Allreduce latency on 4 GPUs for medium to large message sizes

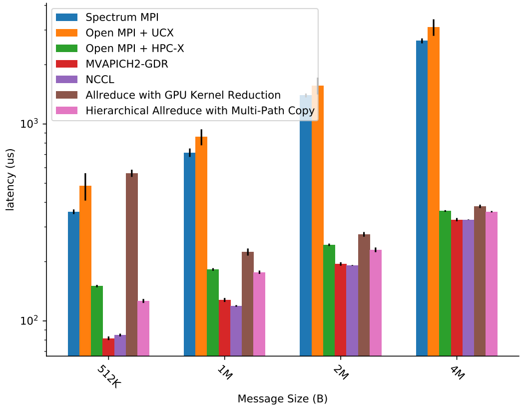
MPI_Allreduce Results (Medium Message Sizes)



MPI_Allreduce latency on 4 GPUs for medium to large message sizes

- 2.75x speedup over Open MPI + UCX at 1MB

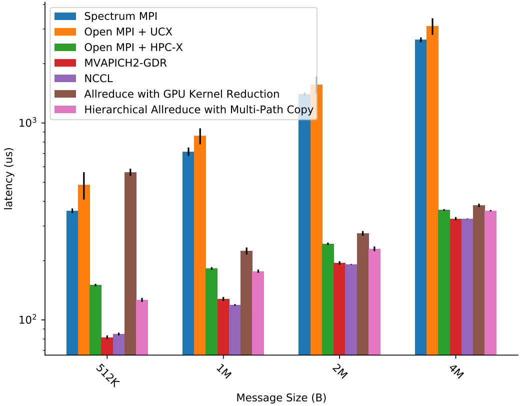
MPI_Allreduce Results (Medium Message Sizes)



MPI_Allreduce latency on 4 GPUs for medium to large message sizes

- 2.75x speedup over Open MPI + UCX at 1MB
- 2.03x speedup over Spectrum MPI at 1MB

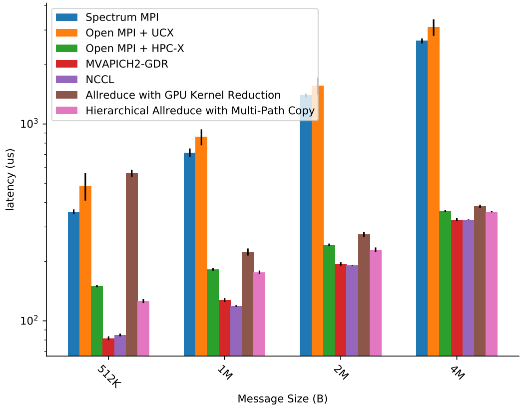
`MPI_Allreduce` Results (Medium Message Sizes)



`MPI_Allreduce` latency on 4 GPUs for medium to large message sizes

- 2.75x speedup over Open MPI + UCX at 1MB
- 2.03x speedup over Spectrum MPI at 1MB
  - We suspect a host-staged implementation of Spectrum MPI

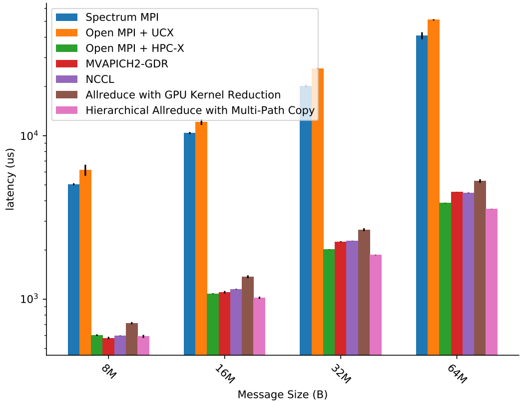
MPI_Allreduce Results (Medium Message Sizes)



`MPI_Allreduce` latency on 4 GPUs for medium to large message sizes

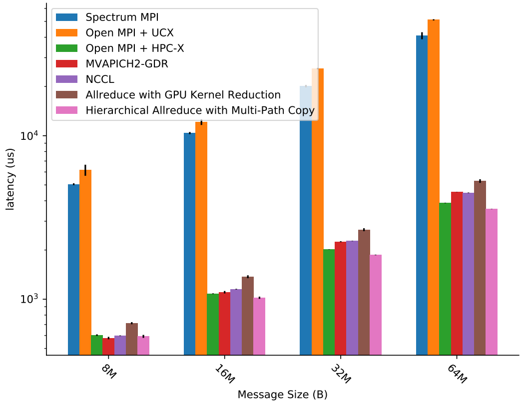
- 2.75x speedup over Open MPI + UCX at 1MB
- 2.03x speedup over Spectrum MPI at 1MB
  - We suspect a host-staged implementation of Spectrum MPI
- Comparable performance to Open MPI + HPC-X

MPI_Allreduce Results (Large Message Sizes)



MPI_Allreduce latency on 4 GPUs for large message sizes

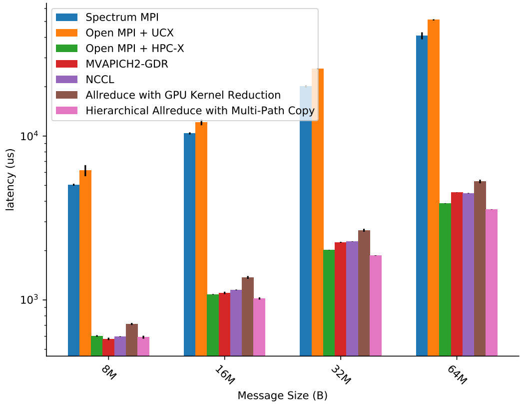
MPI_Allreduce Results (Large Message Sizes)



MPI_Allreduce latency on 4 GPUs for large message sizes

- Comparable performance to optimised libraries at 8MB

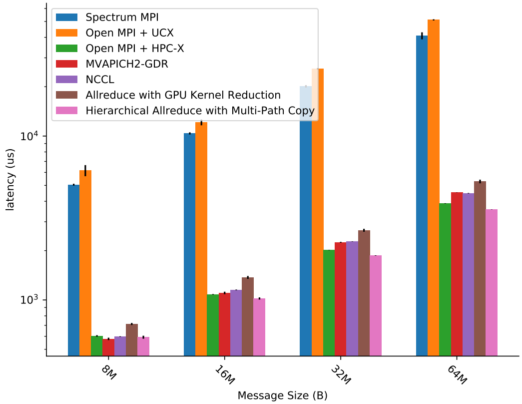
MPI_Allreduce Results (Large Message Sizes)



MPI_Allreduce latency on 4 GPUs for large message sizes

- Comparable performance to optimised libraries at 8MB
- Outperformed all libraries after 16MB

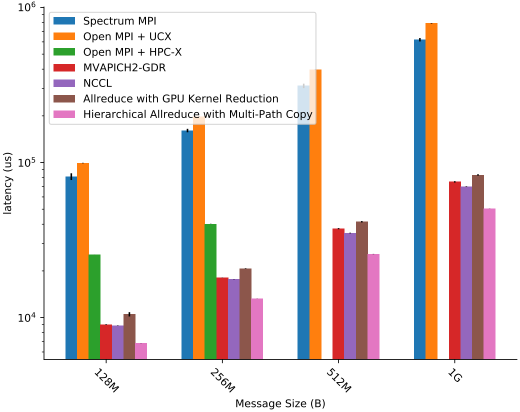
`MPI_Allreduce` Results (Large Message Sizes)



`MPI_Allreduce` latency on 4 GPUs for large message sizes

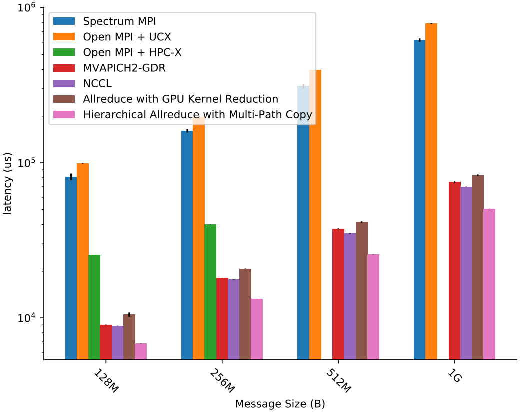
- Comparable performance to optimised libraries at 8MB
- Outperformed all libraries after 16MB
  - Open MPI + HPC-X by 1.09x
  - MVAPICH2-GDR by 1.26x
  - NCCL by 1.25x

MPI_Allreduce Results (Very Large Message Sizes)



MPI_Allreduce latency on 4 GPUs for very large message sizes

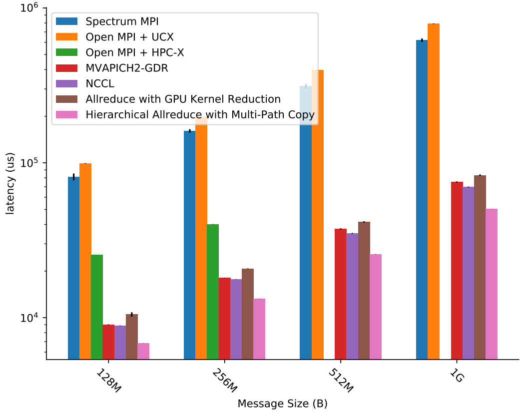
MPI_Allreduce Results (Very Large Message Sizes)



MPI_Allreduce latency on 4 GPUs for very large message sizes

- Much lower latency than Open MPI + HPC-X

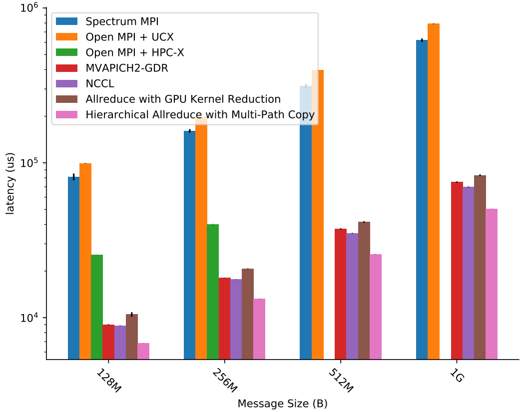
MPI_Allreduce Results (Very Large Message Sizes)



`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 Results (Very Large Message Sizes)

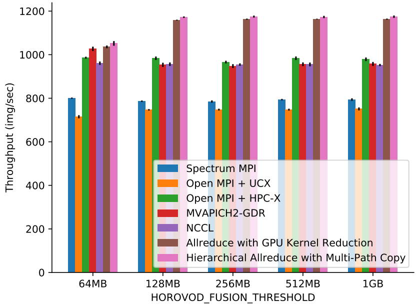


`MPI_Allreduce` latency on 4 GPUs for very large message sizes

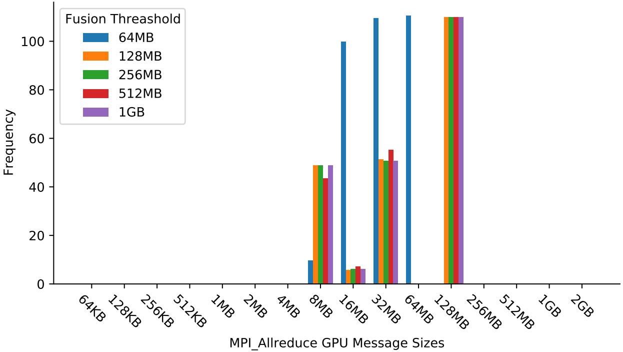
- Much lower latency than Open MPI + HPC-X
- At 1GB we see speedup of:
  - 1.47x over MVAPICH2-GDR
  - 1.38x over NCCL

Application Results (ResNet50)

- ResNet50 up to 1.56x speedup



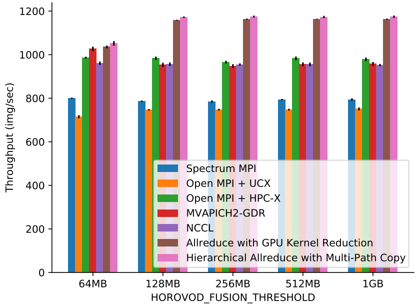
Synthetic Horovod + TensorFlow benchmarks for ResNet50



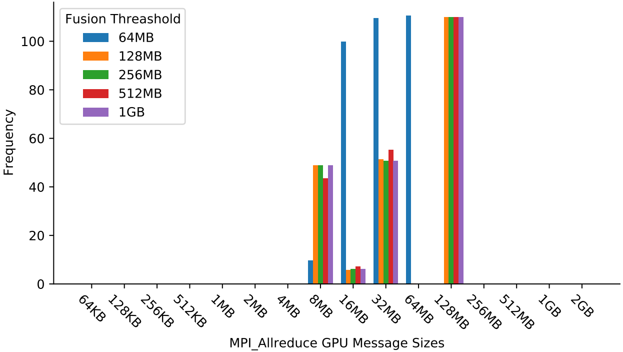
GPU Message Sizes for different `HOROVOD_FUSION_THREASHOLD`

Application Results (ResNet50)

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



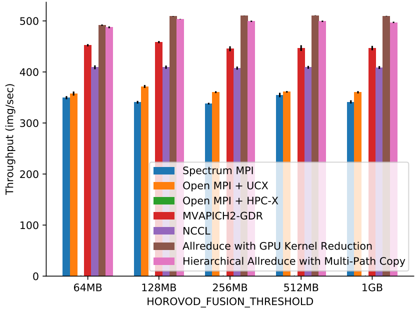
Synthetic Horovod + TensorFlow benchmarks for ResNet50



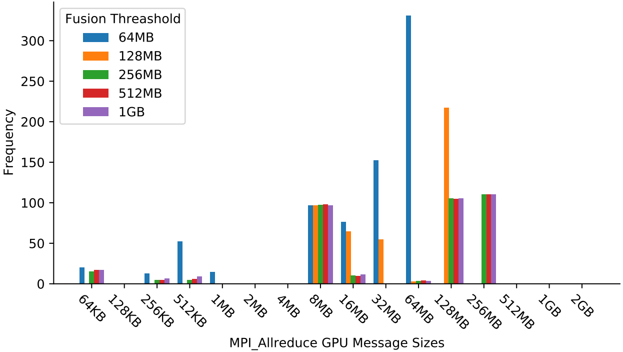
GPU Message Sizes for different `HOROVOD_FUSION_THREASHOLD`

Application Results (ResNet152)

- ResNet152 up to 1.40x speedup



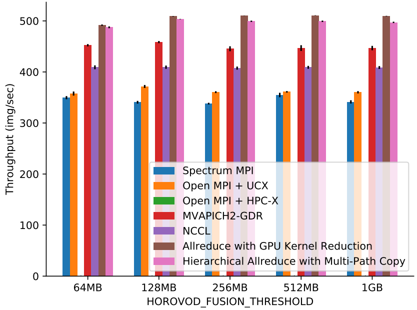
Synthetic Horovod + TensorFlow benchmarks for ResNet152



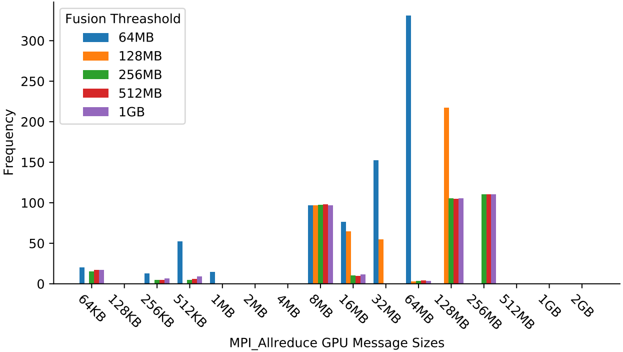
GPU Message Sizes for different `HOROVOD_FUSION_THRESHOLD`

Application Results (ResNet152)

- ResNet152 up to 1.40x speedup
- Modifying fusion threshold increases message sizes to 256MB
  - Also results in smaller messages



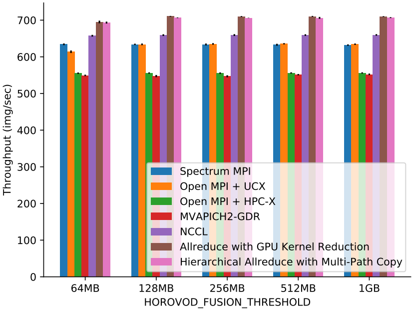
Synthetic Horovod + TensorFlow benchmarks for ResNet152



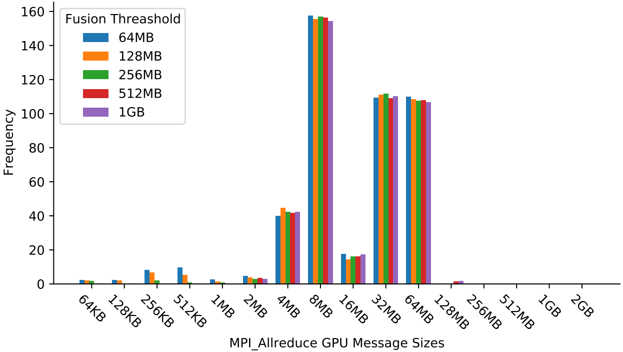
GPU Message Sizes for different `HOROVOD_FUSION_THRESHOLD`

Application Results (DenseNet201)

- DenseNet201 up to 1.26x speedup



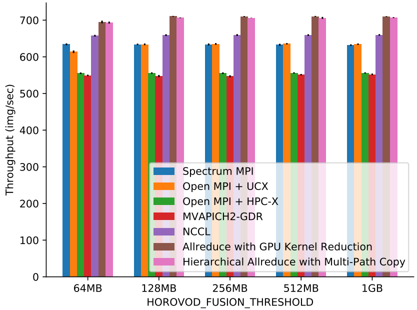
Synthetic Horovod + TensorFlow benchmarks for DenseNet201



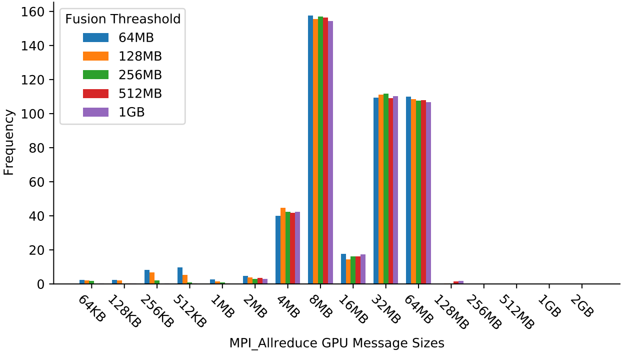
GPU Message Sizes for different `HOROVOD_FUSION_THRESHOLD`

Application Results (DenseNet201)

- DenseNet201 up to 1.26x speedup
- Modifying fusion threshold has minimal impact on message sizes



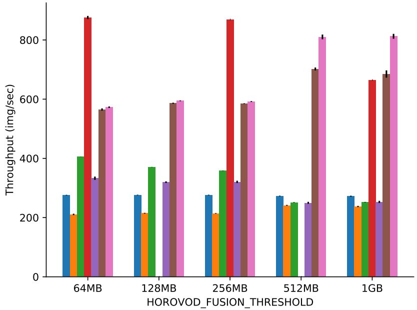
Synthetic Horovod + TensorFlow benchmarks for DenseNet201



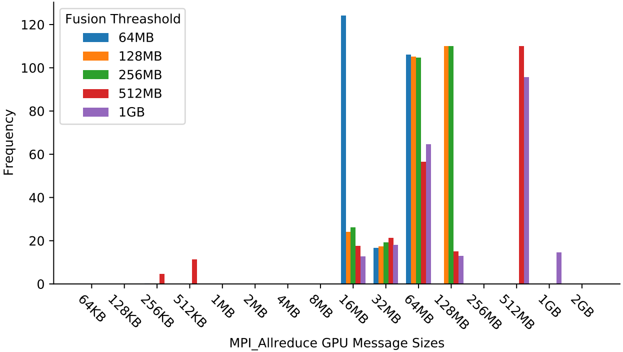
GPU Message Sizes for different `HOROVOD_FUSION_THRESHOLD`

Application Results (VGG16)

- VGG16 up to 3.42x speedup



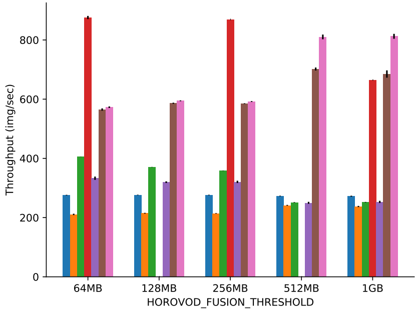
Synthetic Horovod + TensorFlow benchmarks for VGG16



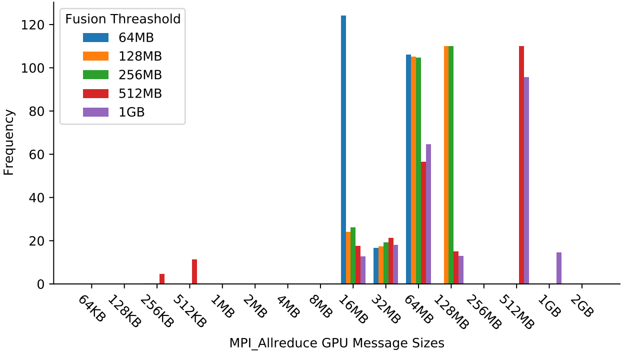
GPU Message Sizes for different `HOROVOD_FUSION_THRESHOLD`

Application Results (VGG16)

- VGG16 up to 3.42x speedup
- Modifying fusion threshold increases message sizes to 1GB



Synthetic Horovod + TensorFlow benchmarks for VGG16



GPU Message Sizes for different `HOROVOD_FUSION_THRESHOLD`

Conclusion

- Deep Learning workloads use `MPI_Allreduce` collective extensively
  - Great importance on large messages

Conclusion

- Deep Learning workloads use `MPI_Allreduce` collective extensively
  - Great importance on large messages
- Intra-node `MPI_Allreduce` communication in MPI is not well optimised

Conclusion

- Deep Learning workloads use `MPI_Allreduce` collective extensively
  - Great importance on large messages
- Intra-node `MPI_Allreduce` communication in MPI is not well optimised
- We propose an intra-socket multi-path point-to-point communication in UCX
  - We now utilised unused bandwidth
  - Directly improves MPI for GPU messages

Conclusion

- Deep Learning workloads use `MPI_Allreduce` collective extensively
  - Great importance on large messages
- Intra-node `MPI_Allreduce` communication in MPI is not well optimised
- We propose an intra-socket multi-path point-to-point communication in UCX
  - We now utilised unused bandwidth
  - Directly improves MPI for GPU messages
- We proposed a new hierarchical `MPI_Allreduce` collective design
  - Is NVLink/PCIe-aware
  - Uses in-GPU reduction
  - Uses our proposed Multi-path copy

Conclusion

- Deep Learning workloads use `MPI_Allreduce` collective extensively
  - Great importance on large messages
- Intra-node `MPI_Allreduce` communication in MPI is not well optimised
- We propose an intra-socket multi-path point-to-point communication in UCX
  - We now utilised unused bandwidth
  - Directly improves MPI for GPU messages
- We proposed a new hierarchical `MPI_Allreduce` collective design
  - Is NVLink/PCIe-aware
  - Uses in-GPU reduction
  - Uses our proposed Multi-path copy
- For Deep Learning Applications we see up to 3.42x speedup with these proposed ideas

Future Work

- We intend to study GPU-based HPC and other Deep Learning workloads

Future Work

- We intend to study GPU-based HPC and other Deep Learning workloads
- Applying the Multi-Path Copy to other hardware topologoies

Future Work

- We intend to study GPU-based HPC and other Deep Learning workloads
- Applying the Multi-Path Copy to other hardware topologoies
- We plan to devise cluster-wide collective algorithms

Future Work

- We intend to study GPU-based HPC and other Deep Learning workloads
- Applying the Multi-Path Copy to other hardware topologoies
- We plan to devise cluster-wide collective algorithms
- We would like to study dynamic tuning approaches

Thank You!

Acknowledgements

Queen's University

compute canada | calcul canada

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

NSERC CRSNG