Micro-Benchmarking MPI Partitioned Point-to-Point Communication

Yıltan Hassan Temuçin[1], Ryan E. Grant[2], and Ahmad Afsahi[1]

[1]The Parallel Processing Research Laboratory (PPRL)
Department of Electrical and Computer Engineering
Queen's University, Canada

[2]Computing at Extreme Scale Advanced Research Laboratory (CAESAR)
Department of Electrical and Computer Engineering
Queen's University, Canada

51st International Conference on Parallel Processing (ICPP '22),
August 29-September 1, 2022

Outline

Introduction

Multi-Threaded MPI Point-to-Point Communication

MPI Partitioned Point-to-Point Communication

Motivation

Micro-Benchmark Design And Results

Overhead

Perceived Bandwidth

Application Availability

Early Bird Communication

Sweep3D Communication Pattern

Halo3D Communication Pattern

Potential Application Improvements

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

Introduction

- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
- The Message Passing Interface (MPI)

Introduction

- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
- The Message Passing Interface (MPI)
  - Popular parallel programming model in HPC

Introduction

- HPC is used to solve large complex problems in many domains
  - Communication is one of the main bottlenecks in applications
- 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 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 Threading Modes:
  - `MPI_THREAD_SINGLE`
  - `MPI_THREAD_FUNNELLED`
  - `MPI_THREAD_SERIALIZED`
  - `MPI_THREAD_MULTIPLE`

Multi-Threaded MPI Point-to-Point Communication

- MPI_THREAD_SERIALIZED

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
}

MPI_Isend(...);
MPI_Irecv(...);

MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- MPI_THREAD_SERIALIZED
  - All work is executed in the parallel region
  - Wait for the all the threads to join
  - Transfer Data

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
}

MPI_Isend(...);
MPI_Irecv(...);

MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- `MPI_THREAD_SERIALIZED`
  - All work is executed in the parallel region
  - Wait for the all the threads to join
  - Transfer Data
- Some threads may finish early and may become idle as they wait

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
}

MPI_Isend(...);
MPI_Irecv(...);

MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- `MPI_THREAD_SERIALIZED`
  - All work is executed in the parallel region
  - Wait for the all the threads to join
  - Transfer Data
- Some threads may finish early and may become idle as they wait
- Network bandwidth becomes under utilized.

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
}

MPI_Isend(...);
MPI_Irecv(...);

MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- `MPI_THREAD_MULTIPLE`

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
      /* Do Work */
      MPI_Isend(...);
      MPI_Irecv(...);
}
MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- `MPI_THREAD_MULTIPLE`
  - All work is executed in the parallel region
  - As each thread completes its work, MPI calls are made

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
        /* Do Work */
        MPI_Isend(...);
        MPI_Irecv(...);
}
MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- MPI_THREAD_MULTIPLE
  - All work is executed in the parallel region
  - As each thread completes its work, MPI calls are made
- Although it may appear that data is transferred as soon as data is ready, communication may be serialized.

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Isend(...);
    MPI_Irecv(...);
}
MPI_Waitall(..);
```

Multi-Threaded MPI Point-to-Point Communication

- MPI_THREAD_MULTIPLE
  - All work is executed in the parallel region
  - As each thread completes its work, MPI calls are made
- Although it may appear that data is transferred as soon as data is ready, communication may be serialized.
- Multi-threaded communication usually has issues with MPI's message matching queues.

```
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Isend(...);
    MPI_Irecv(...);
}
MPI_Waitall(..);
```

MPI Partitioned Point-to-Point Communication

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

```
MPI_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

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_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

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_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

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_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

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_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

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

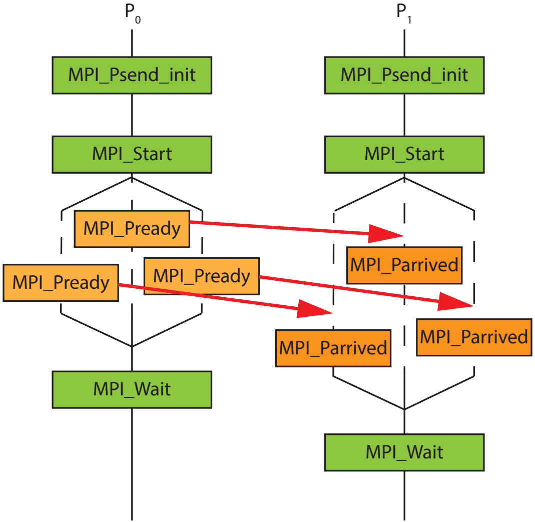
```
MPI_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

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.

```
MPI_Psend_init(...);
MPI_Precv_init(...);
MPI_Start(...);
#pragma omp parallel for
for(int i=1; i<100; ++i)
{
    /* Do Work */
    MPI_Pready(...);
    int flag = 0;
    while (!flag)
    {
        /* Do Work */
        MPI_Parrived(...);
    }
    /* Do Work */
}
MPI_Waitall(..);
```

MPI Partitioned Point-to-Point Communication



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?

Micro-Benchmark Design

- Point-to-Point Metrics
  - Overhead
  - Perceived Bandwidth
  - Early Bird Communication
  - Availability

Micro-Benchmark Design

- Point-to-Point Metrics
  - Overhead
  - Perceived Bandwidth
  - Early Bird Communication
  - Availability
- Communication Patterns
  - Halo3D
  - Sweep3D

Micro-Benchmark Design

- Point-to-Point Metrics
  - Overhead
  - Perceived Bandwidth
  - Early Bird Communication
  - Availability
- Communication Patterns
  - Halo3D
  - Sweep3D
- Noise Models
  - Single Thread Delay
  - Uniform Distribution
  - Gaussian Distribution

Micro-Benchmark Design

- Point-to-Point Metrics
  - Overhead
  - Perceived Bandwidth
  - Early Bird Communication
  - Availability
- Communication Patterns
  - Halo3D
  - Sweep3D
- Noise Models
  - Single Thread Delay
  - Uniform Distribution
  - Gaussian Distribution
- Hot vs. Cold Cache

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

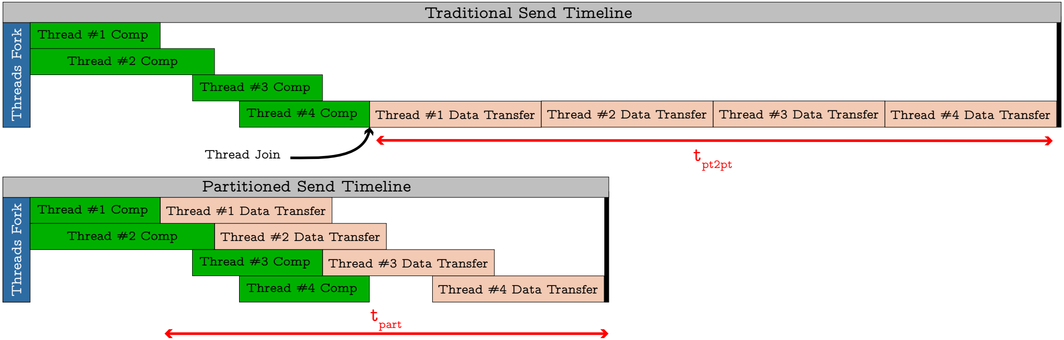
- What is the cost of using MPI Partitioned?
  - We measure each individual data transfer
  - Compare it to MPI Point-to-Point

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

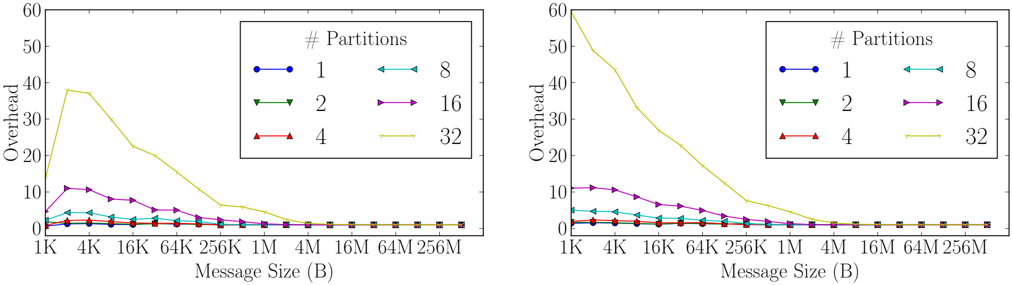
Overhead

- 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



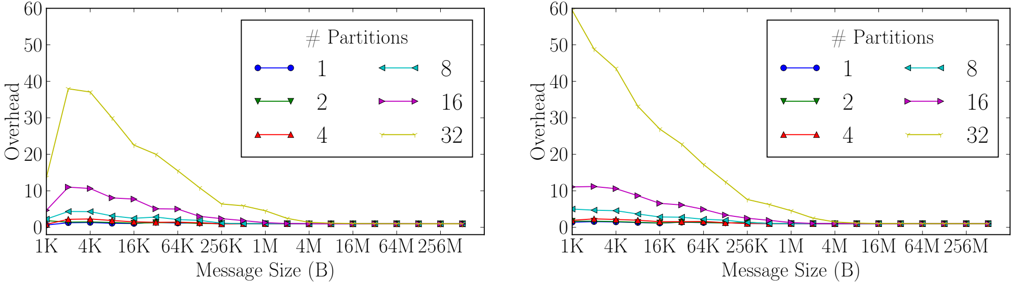
(a) Cold Cache

(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.



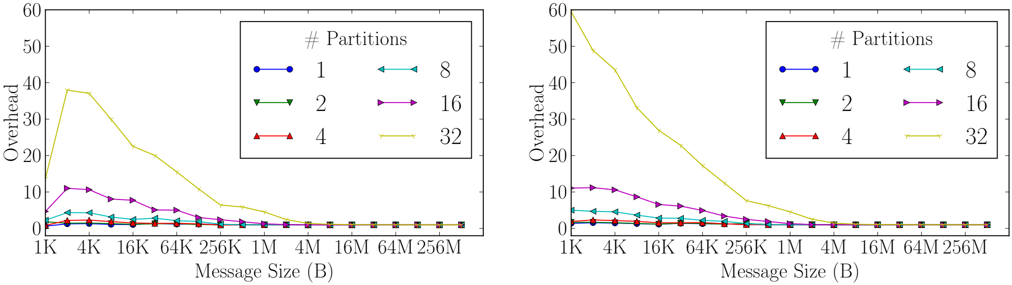
(a) Cold Cache

(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

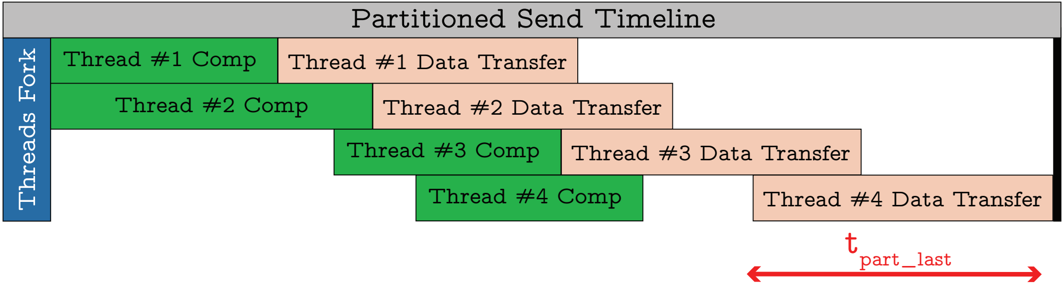
- What would be the required network bandwidth for MPI Point-to-Point to perform the same as MPI Partitioned?

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

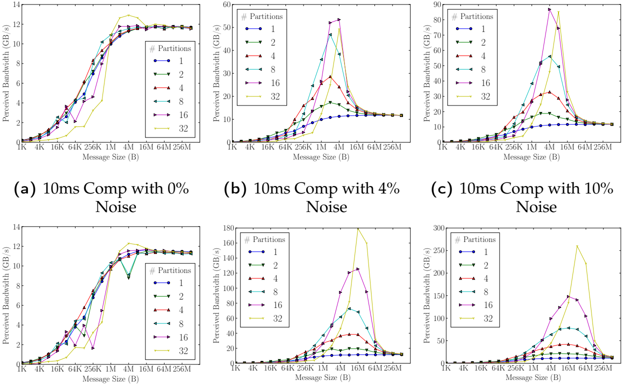
Perceived Bandwidth

- 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

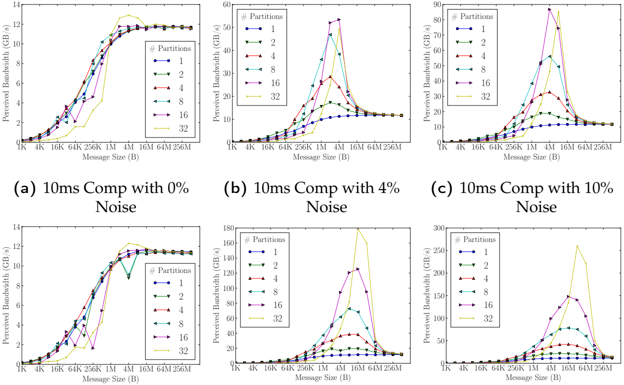
(e) 100ms Comp with 4% Noise

(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

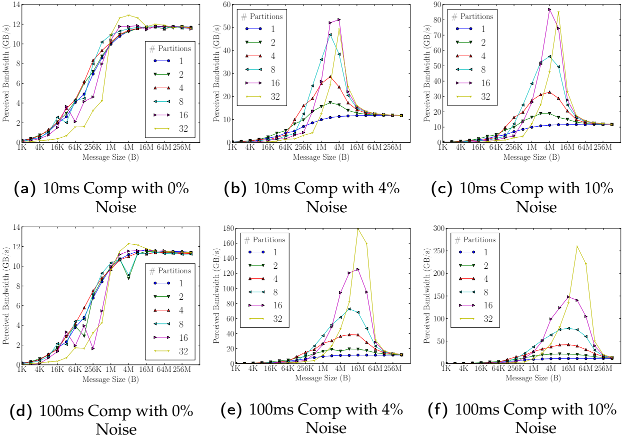
(e) 100ms Comp with 4% Noise

(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

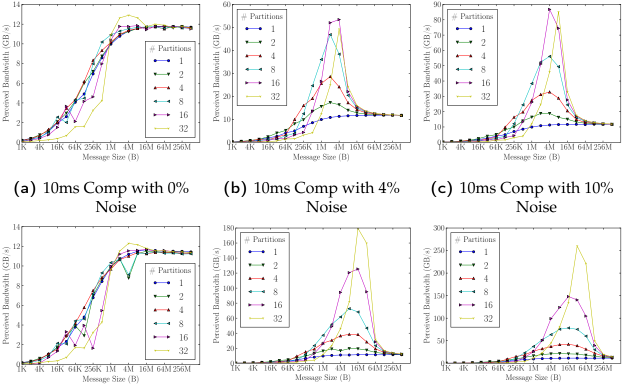
(e) 100ms Comp with 4% Noise

(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

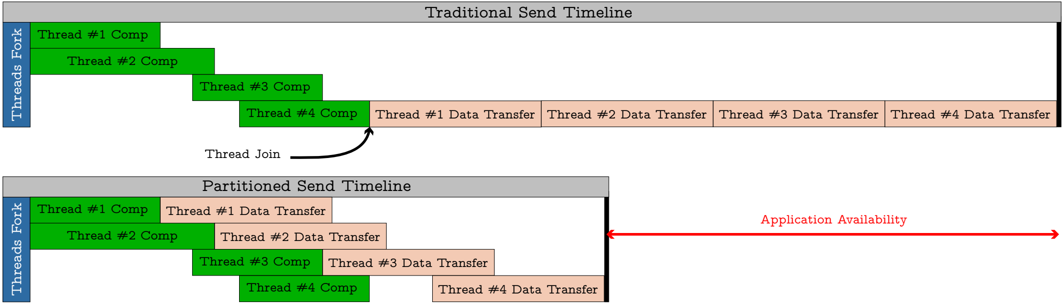
Perceived Bandwidth of MPI Partitioned Point-to-Point Communication with Uniform Noise and a Hot Cache for Different Noise and Compute Amounts

Application Availability

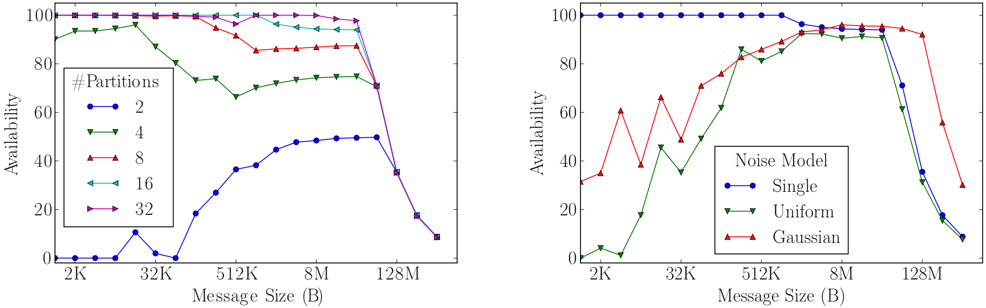
- If we switched to MPI Partitioned, how much extra work could the application do?

Application Availability

- If we switched to MPI Partitioned, how much extra work could the application do?



Application Availability



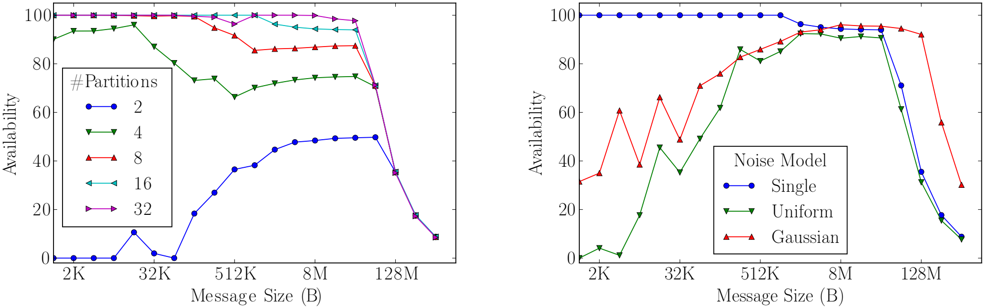
(a) Partition Counts (Single)

(b) Noise Distributions (16P)

Application Availability with 100ms Compute and 4% Noise

Application Availability

- Increasing partitions improve application availability for applications with a single thread delay.



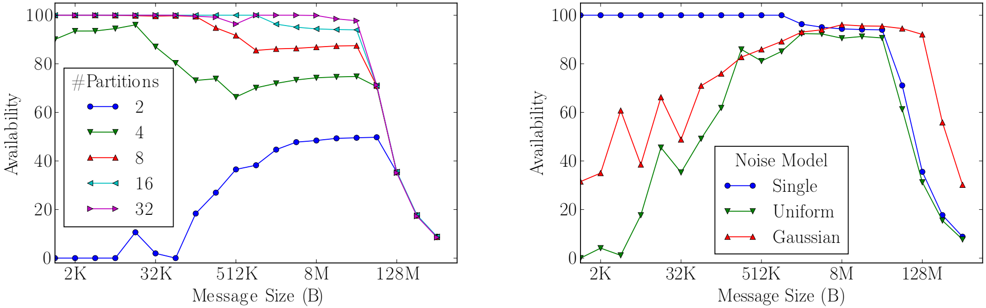
(a) Partition Counts (Single)

(b) Noise Distributions (16P)

Application Availability with 100ms Compute and 4% Noise

Application Availability

- Increasing partitions improve application availability for applications with a single thread delay.
- Application noise will change how much can be gained from using MPI Partitioned.



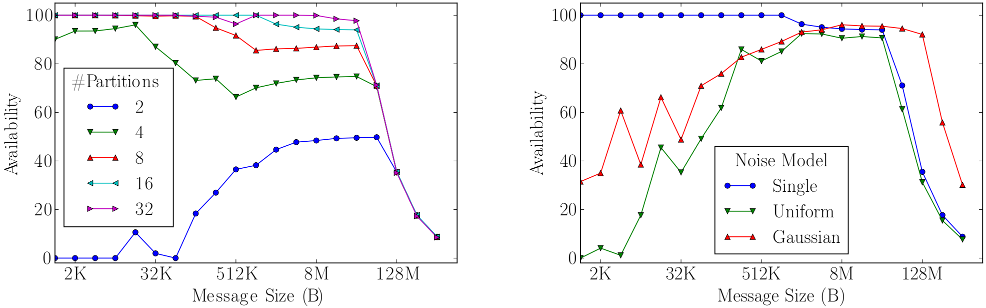
(a) Partition Counts (Single)

(b) Noise Distributions (16P)

Application Availability with 100ms Compute and 4% Noise

Application Availability

- Increasing partitions improve application availability for applications with a single thread delay.
- Application noise will change how much can be gained from using MPI Partitioned.
- Less beneficial for very large messages.



(a) Partition Counts (Single)

(b) Noise Distributions (16P)

Application Availability with 100ms Compute and 4% Noise

Early Bird Communication

- How much communication occurs before the thread join?

Early Bird Communication

- How much communication occurs before the thread join?
  - How much do we overlap communication?

Early Bird Communication

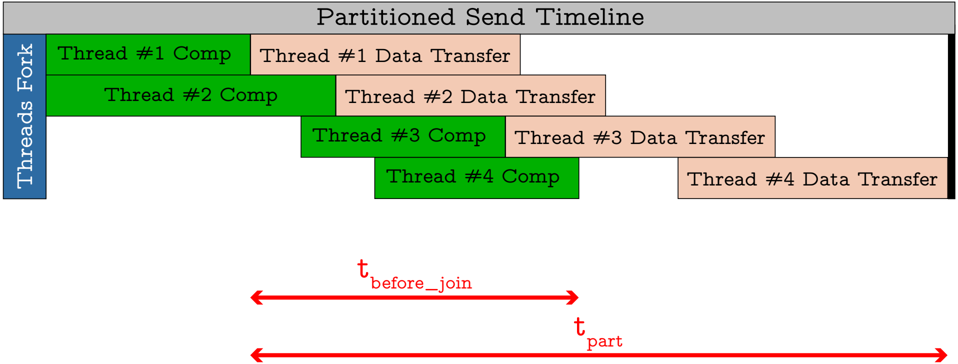
- How much communication occurs before the thread join?
  - How much do we overlap communication?

$$\%\ Early\ Bird = \frac{t_{before\_join}}{t_{part}}$$

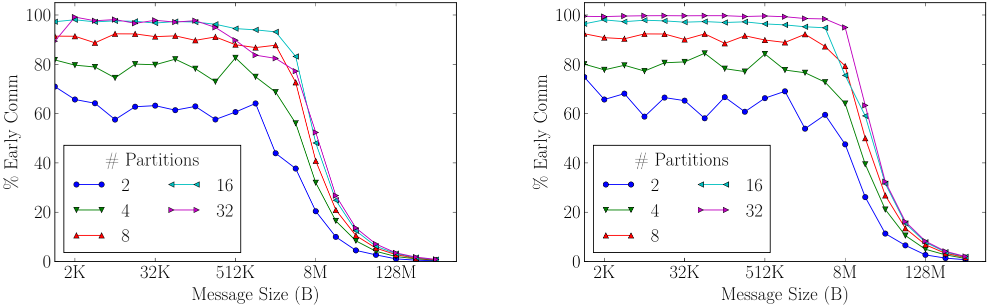
Early Bird Communication

- How much communication occurs before the thread join?
  - How much do we overlap communication?

$$\%\ Early\ Bird = \frac{t_{before\_join}}{t_{part}}$$



Early Bird Communication

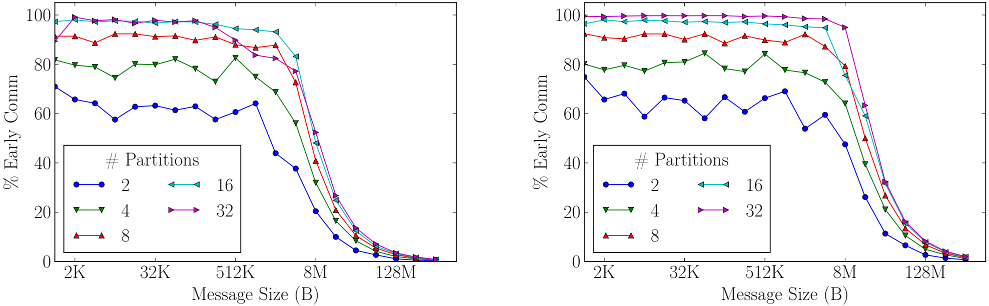


(a) 4% Noise

(b) 10% Noise

Early Bird Communication with 10ms Compute and Uniform Noise

Early Bird Communication



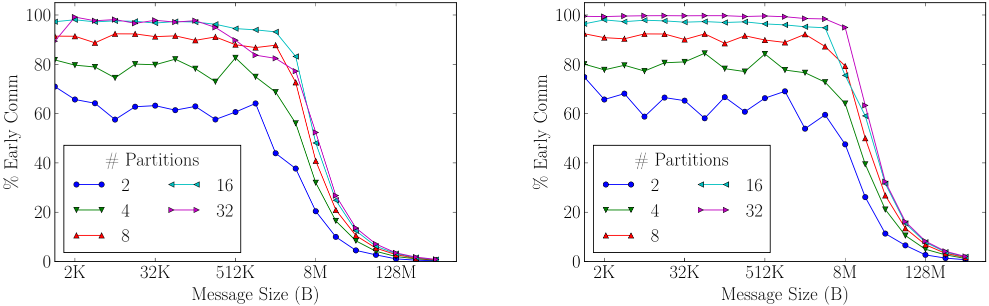
(a) 4% Noise

(b) 10% Noise

Early Bird Communication with 10ms Compute and Uniform Noise

Early Bird Communication

- Better overlap with more partitions for small message sizes.



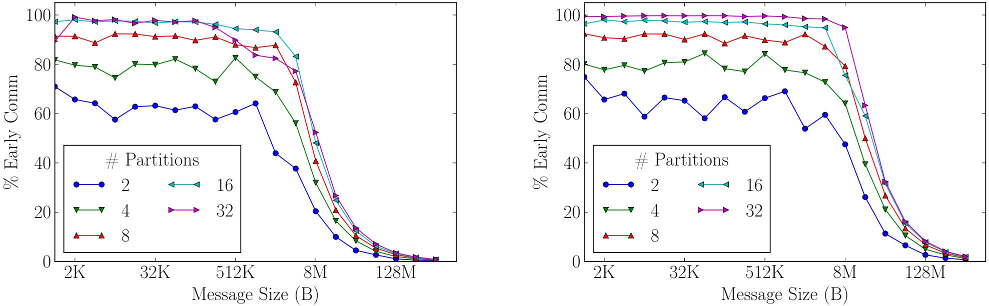
(a) 4% Noise

(b) 10% Noise

Early Bird Communication with 10ms Compute and Uniform Noise

Early Bird Communication

- Better overlap with more partitions for small message sizes.
- More allows for more overlap for large messages.

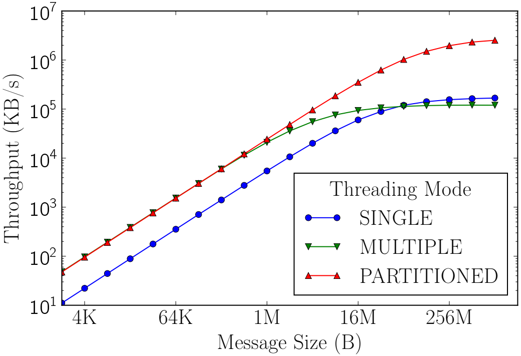


(a) 4% Noise

(b) 10% Noise

Early Bird Communication with 10ms Compute and Uniform Noise

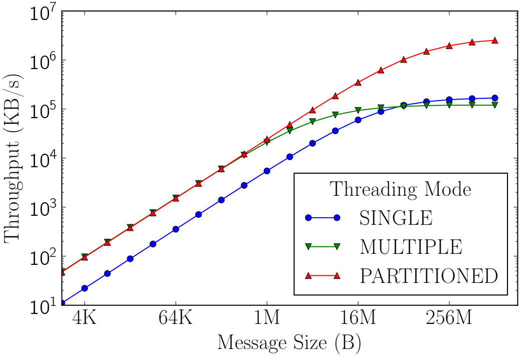
Sweep3D Communication Pattern



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

Sweep3D Communication Pattern

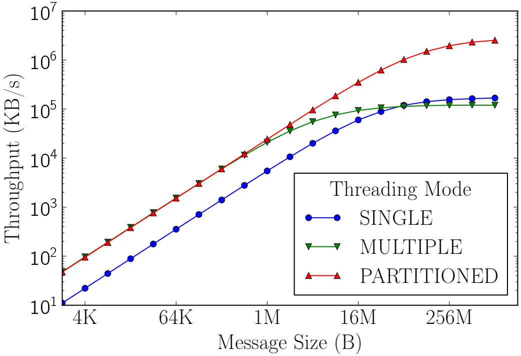
- Sweep3D communication pattern has lots of dependencies.



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

Sweep3D Communication Pattern

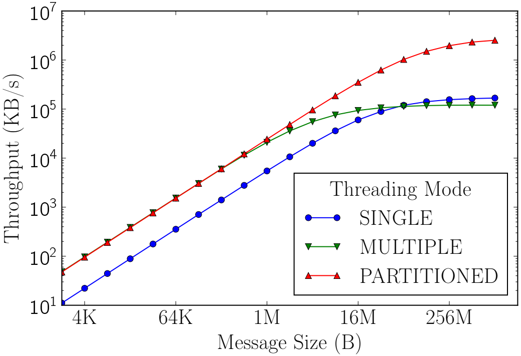
- Sweep3D communication pattern has lots of dependencies.
- Generally, multi-threading performs better than single threaded.



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

Sweep3D Communication Pattern

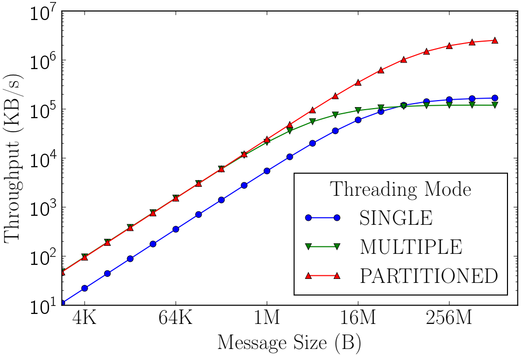
- Sweep3D communication pattern has lots of dependencies.
- Generally, multi-threading performs better than single threaded.
- The MPI Partition implementation in this paper is built upon MPI Send/Recv.



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

Sweep3D Communication Pattern

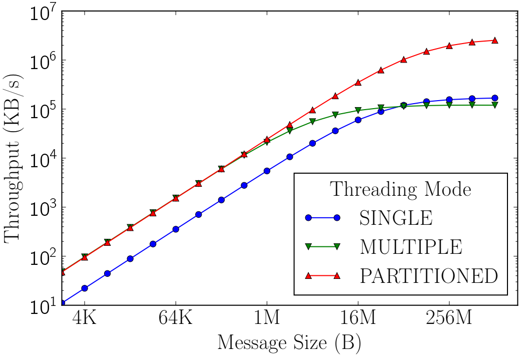
- Sweep3D communication pattern has lots of dependencies.
- Generally, multi-threading performs better than single threaded.
- The MPI Partition implementation in this paper is built upon MPI Send/Recv.
  - Therefore minimal difference for most message sizes.



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

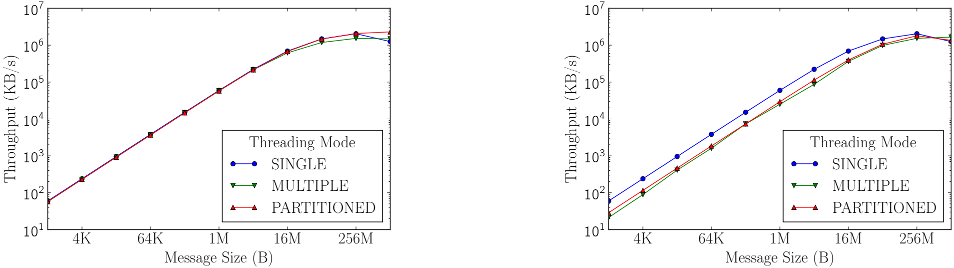
Sweep3D Communication Pattern

- Sweep3D communication pattern has lots of dependencies.
- Generally, multi-threading performs better than single threaded.
- The MPI Partition implementation in this paper 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

Halo3D Communication Pattern



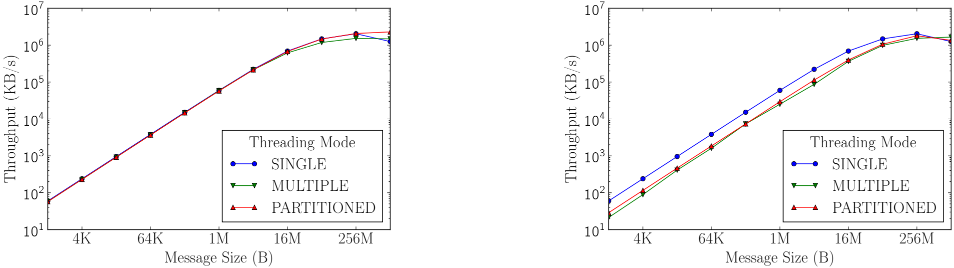
(a) 4 Partitions - 8 Threads

(b) 8 Partitions - 64 Threads (Oversubscribed)

Halo3D Communication Throughput For 10ms, 4% Single Noise with a Hot Cache

Halo3D Communication Pattern

- Halo exchange has lots of synchronization.
  - Minimal difference in our different implementations.



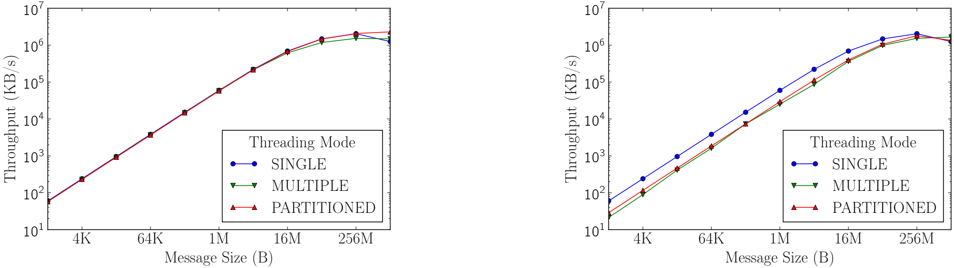
(a) 4 Partitions - 8 Threads

(b) 8 Partitions - 64 Threads (Oversubscribed)

Halo3D Communication Throughput For 10ms, 4% Single Noise with a Hot Cache

Halo3D Communication Pattern

- Halo exchange has lots of synchronization.
  - Minimal difference in our different implementations.
- Additional work could be beneficial to Halo3D using MPI Partitioned
  - 64x increase in total computation with only a 16.8% decrease in throughput.
  - Could benefit from work stealing schemes.

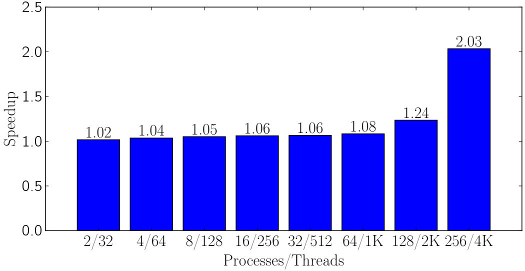


(a) 4 Partitions - 8 Threads

(b) 8 Partitions - 64 Threads (Oversubscribed)

Halo3D Communication Throughput For 10ms, 4% Single Noise with a Hot Cache

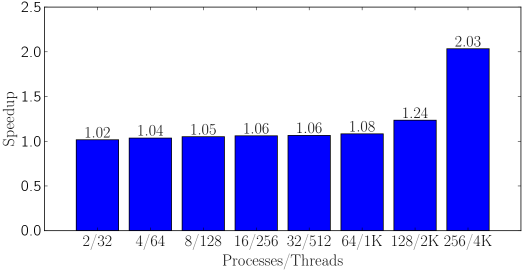
Potential Application Improvements



Expected Speedup From Porting SNAP-C to MPI Partitioned.

Potential Application Improvements

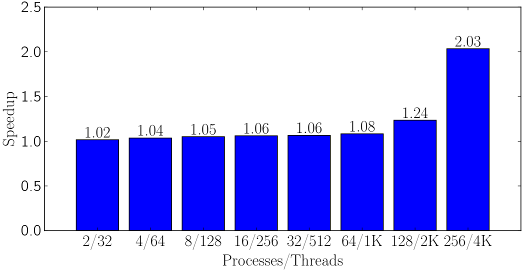
- The Sweep3D communication pattern showed potential for if it were ported to MPI Partitioned.



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

- We provide the first MPI Partitioned Micro-Benchmark Suite.

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache
- Showed what communication patterns could benefit from MPI Partitioned.

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache
- Showed what communication patterns could benefit from MPI Partitioned.

Future Work

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache
- Showed what communication patterns could benefit from MPI Partitioned.

Future Work

- Compare across different MPI implementations

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache
- Showed what communication patterns could benefit from MPI Partitioned.

Future Work

- Compare across different MPI implementations
- Porting Applications to MPI Partitioned

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache
- Showed what communication patterns could benefit from MPI Partitioned.

Future Work

- Compare across different MPI implementations
- Porting Applications to MPI Partitioned
- Accelerator-Triggered MPI Partitioned

Conclusion And Future Work

- We provide the first MPI Partitioned Micro-Benchmark Suite.
- Analyzed MPI Partitioned with a range of different metrics
  - Partition Count
  - Noise Distributions
  - Hot vs Cold Cache
- Showed what communication patterns could benefit from MPI Partitioned.

Future Work

- Compare across different MPI implementations
- Porting Applications to MPI Partitioned
- Accelerator-Triggered MPI Partitioned
- MPI Partitioned Collectives

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