



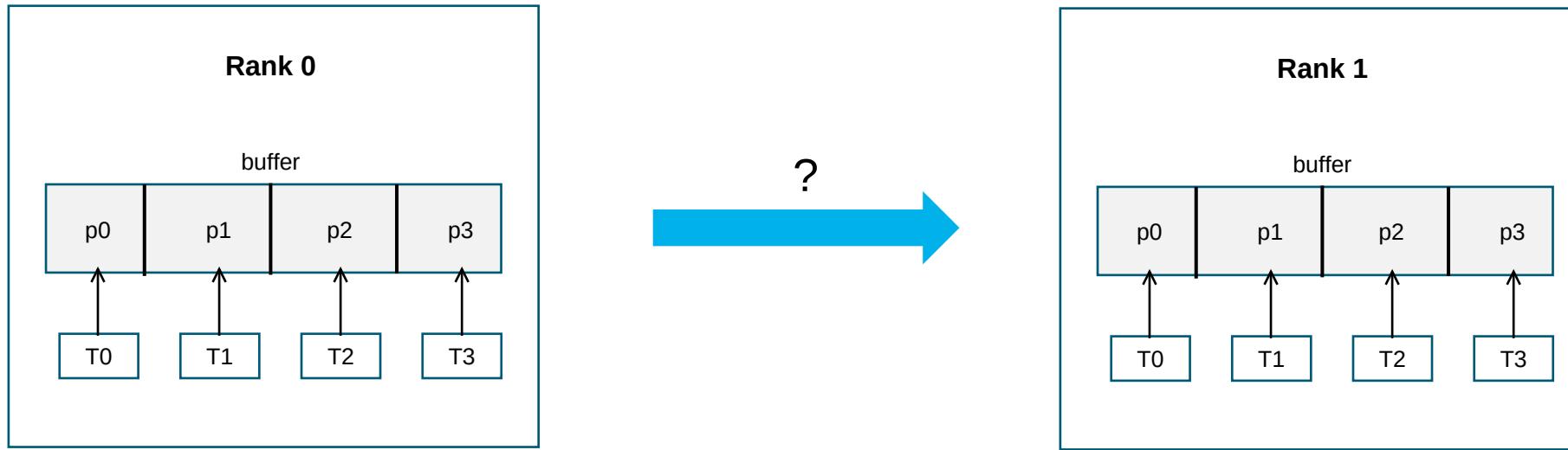
High-Performance  
Computing Center  
Stuttgart

# Performance and Optimization opportunities of MPI partitioned communication

Axel Schneewind

# Motivation: Hybrid Programming

H L R I S



# Outline

H L R I S

- Introduction
  - MPI partitioned communication
  - Relevant sections from the MPI-4.1 Standard
- Possible performance benefits/optimizations
- Benchmarks
  - Psend vs Blocking/Nonblocking Sends (single-/multithreaded)
  - Psend, Isend with completion tests
- Current implementations in OpenMPI and MPICH
- Simple implementation of message aggregation

# Outline

H L R I S

- Introduction
  - MPI partitioned communication
  - Relevant sections from the MPI-4.1 Standard
- Possible performance benefits/optimizations
- Benchmarks
  - Psend vs Blocking/Nonblocking Sends (single-/multithreaded)
  - Psend, Isend with completion tests
- Current implementations in OpenMPI and MPICH
- Simple implementation of message aggregation

# Outline

H L R I S

- Introduction
  - MPI partitioned communication
  - Relevant sections from the MPI-4.1 Standard
- Possible performance benefits/optimizations
- Benchmarks
  - Psend vs Blocking/Nonblocking Sends (single-/multithreaded)
  - Psend, Isend with completion tests
- Current implementations in OpenMPI and MPICH
- Simple implementation of message aggregation

# Outline

H L R I S

- Introduction
  - MPI partitioned communication
  - Relevant sections from the MPI-4.1 Standard
- Possible performance benefits/optimizations
- Benchmarks
  - Psend vs Blocking/Nonblocking Sends (single-/multithreaded)
  - Psend, Isend with completion tests
- Current implementations in OpenMPI and MPICH
- Simple implementation of message aggregation

# Outline

H L R I S

- Introduction
  - MPI partitioned communication
  - Relevant sections from the MPI-4.1 Standard
- Possible performance benefits/optimizations
- Benchmarks
  - Psend vs Blocking/Nonblocking Sends (single-/multithreaded)
  - Psend, Isend with completion tests
- Current implementations in OpenMPI and MPICH
- Simple implementation of message aggregation

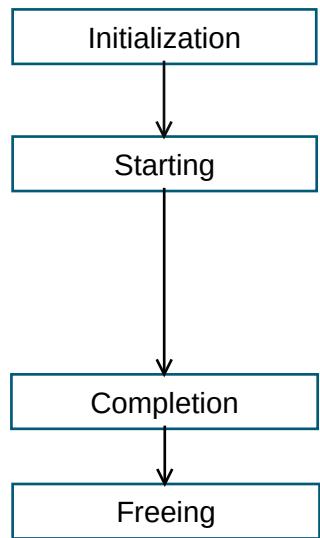
# Outline

H L R I S

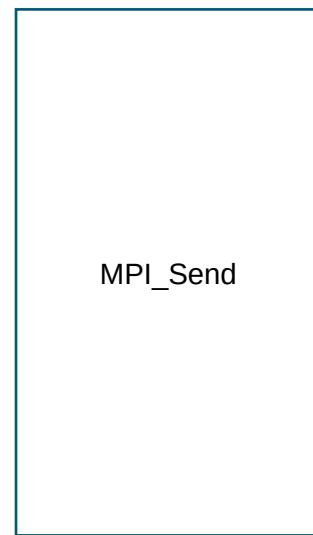
- Introduction
  - MPI partitioned communication
  - Relevant sections from the MPI-4.1 Standard
- Possible performance benefits/optimizations
- Benchmarks
  - Psend vs Blocking/Nonblocking Sends (single-/multithreaded)
  - Psend, Isend with completion tests
- Current implementations in OpenMPI and MPICH
- Simple implementation of message aggregation

# MPI: Transfer Mechanisms

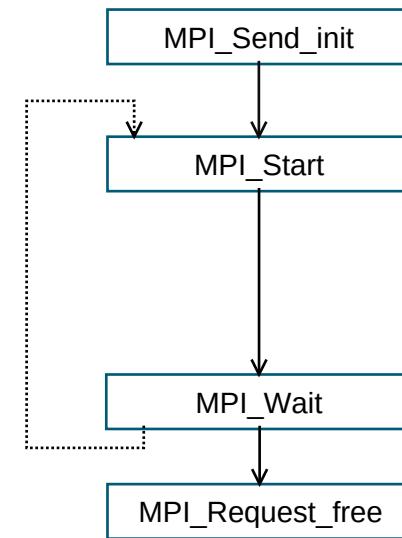
H L R I S



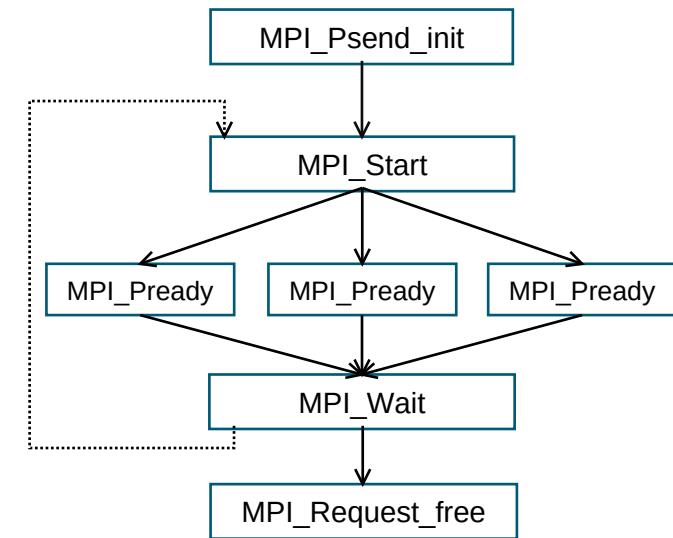
Transfer phases



Blocking Send



Persistent Send  
(similar: Nonblocking Isend)



Partitioned Send  
(since MPI-4.0)

## MPI-4.1: Blocking/Nonblocking Send

H L R I S

**Order (3.5).** Messages are *nonovertaking*: If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending. [analogously for receive operations]

**Order (3.7.4).** Nonblocking communication operations are *ordered* according to the execution order of the calls that initiate the communication.

## MPI-4.1: Blocking/Nonblocking Send

H L R I S

**Order (3.5).** Messages are *nonovertaking*: If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending. [analogously for receive operations]

**Order (3.7.4).** Nonblocking communication operations are *ordered* according to the execution order of the calls that initiate the communication.

# MPI-4.1: Nonblocking Communication Operations

H L R I S

## Progress (3.7.4).

A call to MPI\_WAIT that completes a receive will eventually terminate and return if a matching send has been started, unless the send is satisfied by another receive. [analogously for sender]

- No such requirement for MPI\_Start/MPI\_Pready!
- Similar for completion tests (MPI\_Test / MPI\_Request\_get\_status)
- Calls to these functions might cause earlier message progress

# MPI-4.1: Nonblocking Communication Operations

H L R I S

## Progress (3.7.4).

A call to MPI\_WAIT that completes a receive will eventually terminate and return if a matching send has been started, unless the send is satisfied by another receive. [analogously for sender]

- No such requirement for MPI\_Start/MPI\_Pready!
- Similar for completion tests (MPI\_Test / MPI\_Request\_get\_status)
- Calls to these functions might cause earlier message progress

# MPI-4.1: Nonblocking Communication Operations

H L R I S

## Progress (3.7.4).

A call to MPI\_WAIT that completes a receive will eventually terminate and return if a matching send has been started, unless the send is satisfied by another receive. [analogously for sender]

- No such requirement for MPI\_Start/MPI\_Pready!
- Similar for completion tests (MPI\_Test / MPI\_Request\_get\_status)
- Calls to these functions might cause earlier message progress

# MPI-4.1: Nonblocking Communication Operations

H L R I S

## Progress (3.7.4).

A call to MPI\_WAIT that completes a receive will eventually terminate and return if a matching send has been started, unless the send is satisfied by another receive. [analogously for sender]

- No such requirement for MPI\_Start/MPI\_Pready!
- Similar for completion tests (MPI\_Test / MPI\_Request\_get\_status)
- Calls to these functions might cause earlier message progress

## MPI-4.1: Partitioned Communication

H L R I S

*Rationale.* Partitioned communication is designed to provide opportunities for MPI implementations to optimize data transfers.

MPI is free to choose **how many transfers** to do within a partitioned communication send independent of how many partitions are reported as ready to MPI through MPI\_PREADY calls.

**Aggregation** of partitions is permitted but not required.

**Ordering** of partitions is permitted but not required.

A naive implementation can simply wait for the entire message buffer to be marked ready before any transfer(s) occur and could wait until the completion function is called on a request before transferring data. [...]. (*End of rationale.*)

## MPI-4.1: Partitioned Communication

H L R I S

*Rationale.* Partitioned communication is designed to provide opportunities for MPI implementations to optimize data transfers.

MPI is free to choose **how many transfers** to do within a partitioned communication send independent of how many partitions are reported as ready to MPI through MPI\_PREADY calls.

**Aggregation** of partitions is permitted but not required.

**Ordering** of partitions is permitted but not required.

A naive implementation can simply wait for the entire message buffer to be marked ready before any transfer(s) occur and could wait until the completion function is called on a request before transferring data. [...]. (*End of rationale.*)

## MPI-4.1: Partitioned Communication

H L R I S

*Rationale.* Partitioned communication is designed to provide opportunities for MPI implementations to optimize data transfers.

MPI is free to choose **how many transfers** to do within a partitioned communication send independent of how many partitions are reported as ready to MPI through MPI\_PREADY calls.

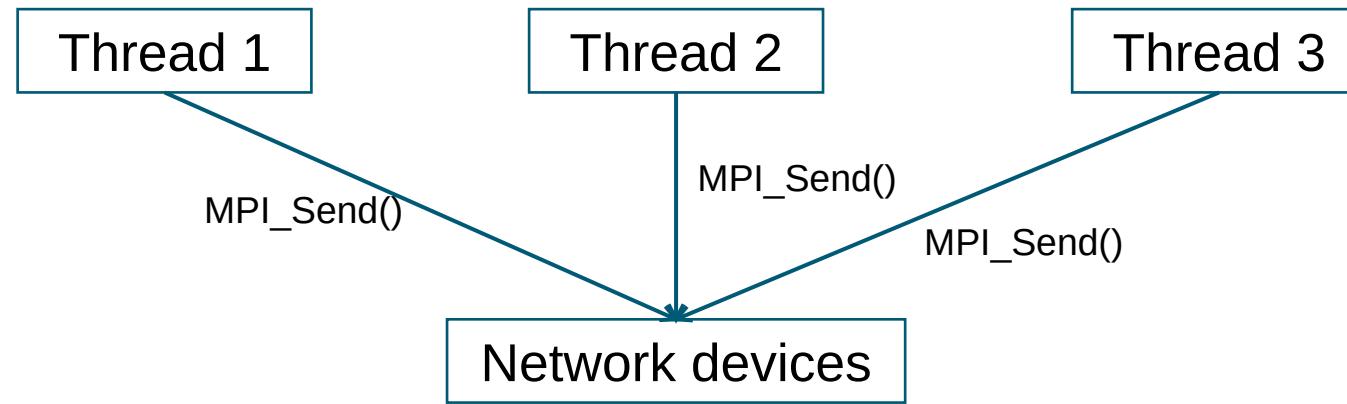
**Aggregation** of partitions is permitted but not required.

**Ordering** of partitions is permitted but not required.

A naive implementation can simply wait for the entire message buffer to be marked ready before any transfer(s) occur and could wait until the completion function is called on a request before transferring data. [...]. (*End of rationale.*)

## Possible performance benefits – Less locking overhead

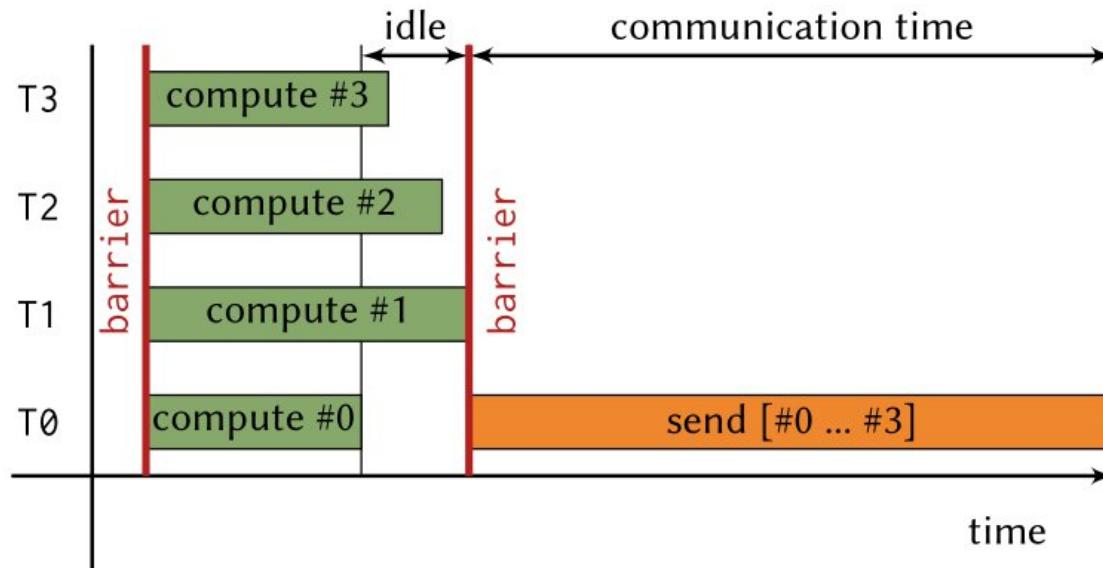
H L R I S



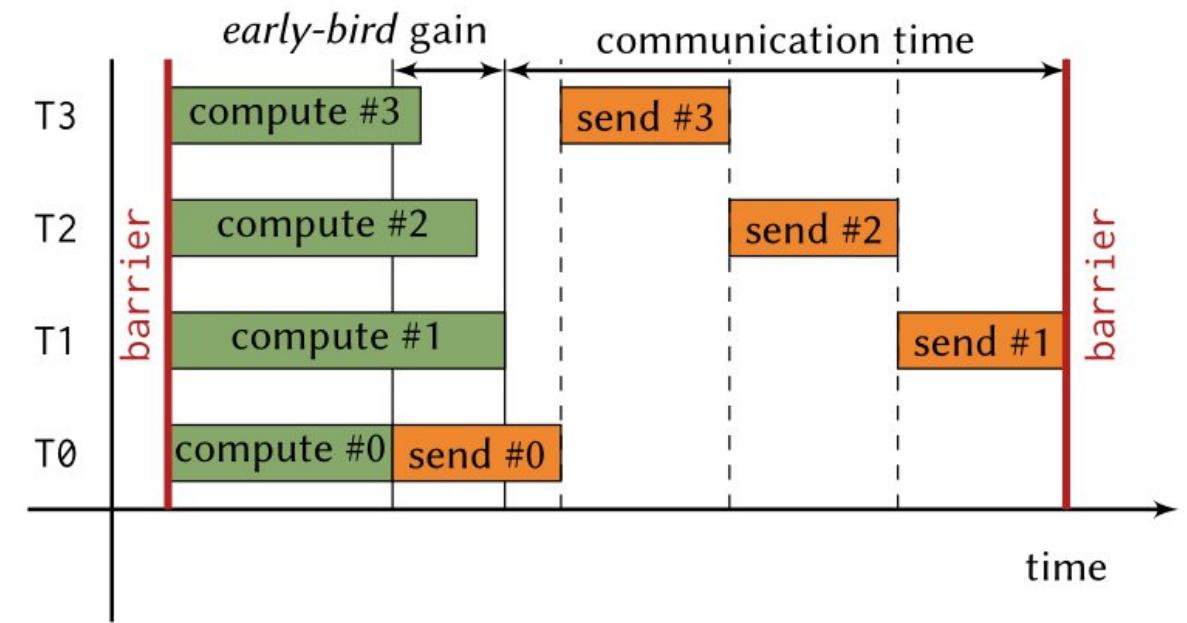
- Preadv not required to interact with network devices
  - Can be called concurrently

# Possible performance benefits – Early-bird effect

H L R I S



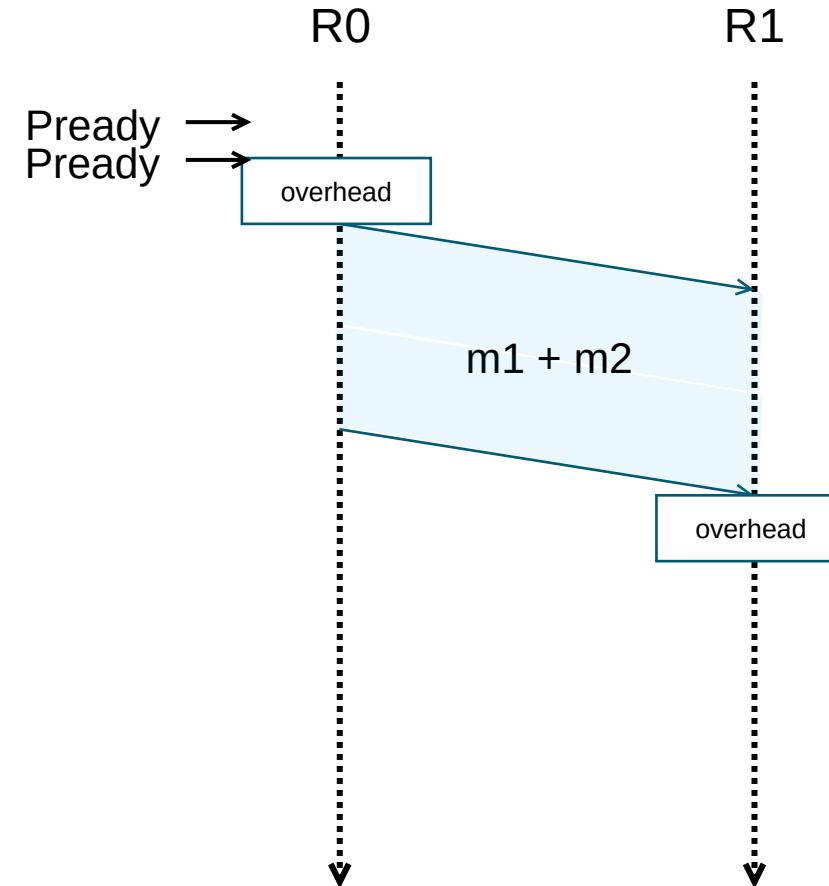
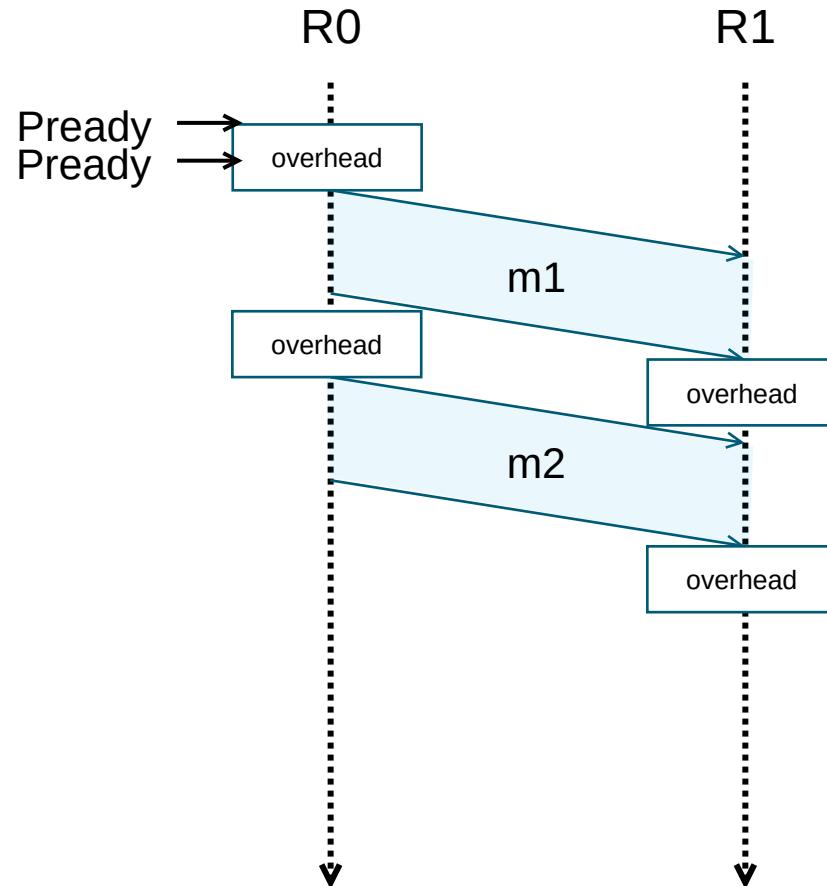
Single transfer for entire buffer



One transfer per partition

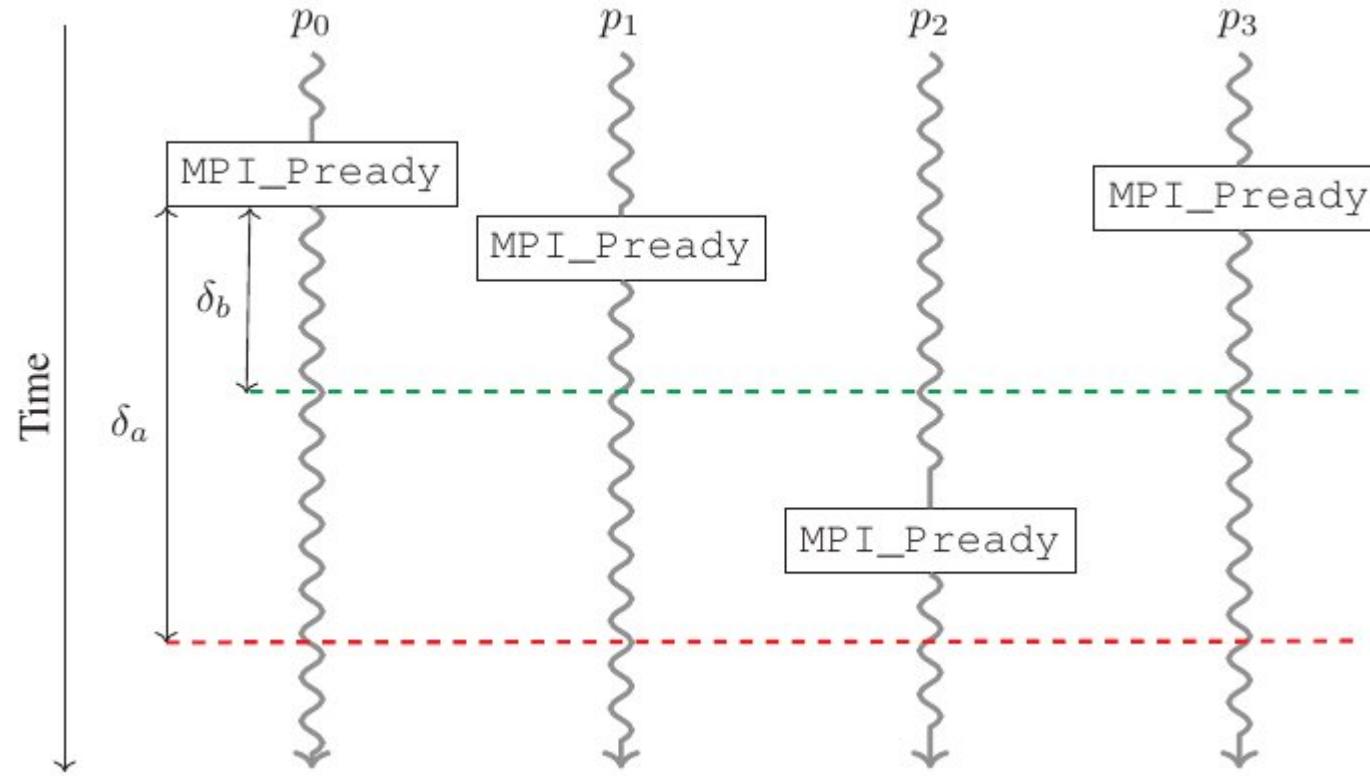
# Possible optimization – Message aggregation

H L R I S



# Possible optimization – Message aggregation

H L R I S



## Benchmarks: Goals

H L R I S

- Measure **effective bandwidth** depending on
  - Partition size
  - Transfer mechanism
  - Order of partitions being marked as ready
  - Usage of completion tests to trigger earlier message progress
  - MPI-implementation: OpenMPI, (MPICH)

## Benchmarking scheme

H L R I S

```
1 # initialization
2
3 for i in 0...num_iterations:
4     MPI_Barrier()    # synchronization
5
6     start_time[i] = MPI_Wtime()
7
8     for p in 0...partition_count in some order:
9         send/receive partition p          # start transfer
10
11    MPI_Wait(request)                 # completion
12
13    end_time[i] = MPI_Wtime()
14
15 # cleanup
16
17 bandwidth = buffer_size / mean(start_time - end_time)
```

# Transfer mechanisms

H L R I S

| Mechanism      | Initialization    | Partition ready | Completion  | Freeing              |
|----------------|-------------------|-----------------|-------------|----------------------|
| Send           | -                 | MPI_Send (p)    | -           | -                    |
| SendPersistent | MPI_Send_init (p) | MPI_Start (p)   | MPI_Waitall | MPI_Request_free (p) |
| Isend          | -                 | MPI_Isend (p)   | MPI_Waitall | MPI_Request_free (p) |
| Psend          | MPI_Psend_init    | MPI_Pready (p)  | MPI_Wait    | MPI_Request_free     |

operations marked with (p) are performed for each partition

# Experimental setup

H L R I S

- 2 nodes on HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s
- **Buffer size:** 8MiB
- **Partition sizes:** 512B, 1024B, ..., 8MiB
- Partitions marked ready in different orders:
  - Left-to-right
  - Randomized
  - Neighbourhood exchange pattern
- Multithreading with OpenMP

# Experimental setup

H L R I S

- 2 nodes on HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s
- **Buffer size:** 8MiB
- **Partition sizes:** 512B, 1024B, ..., 8MiB
- Partitions marked ready in different orders:
  - Left-to-right
  - Randomized
  - Neighbourhood exchange pattern
- Multithreading with OpenMP

# Experimental setup

H L R I S

- 2 nodes on HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s
- **Buffer size:** 8MiB
- **Partition sizes:** 512B, 1024B, ..., 8MiB
- Partitions marked ready in different orders:
  - Left-to-right
  - Randomized
  - Neighbourhood exchange pattern
- Multithreading with OpenMP

# Experimental setup

H L R I S

- 2 nodes on HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s
- **Buffer size:** 8MiB
- **Partition sizes:** 512B, 1024B, ..., 8MiB
- Partitions marked ready in different orders:
  - Left-to-right
  - Randomized
  - Neighbourhood exchange pattern
- Multithreading with OpenMP

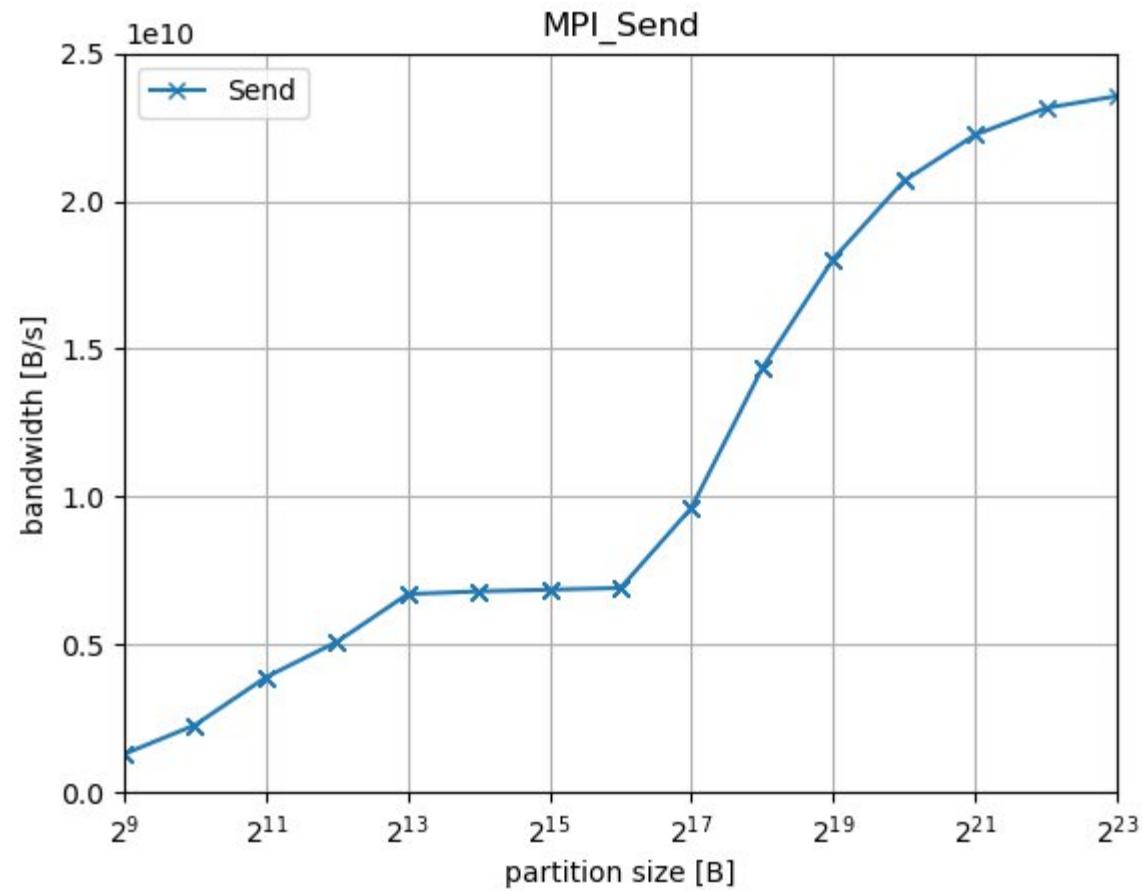
# Experimental setup

H L R I S

- 2 nodes on HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s
- **Buffer size:** 8MiB
- **Partition sizes:** 512B, 1024B, ..., 8MiB
- Partitions marked ready in different orders:
  - Left-to-right
  - Randomized
  - Neighbourhood exchange pattern
- Multithreading with OpenMP

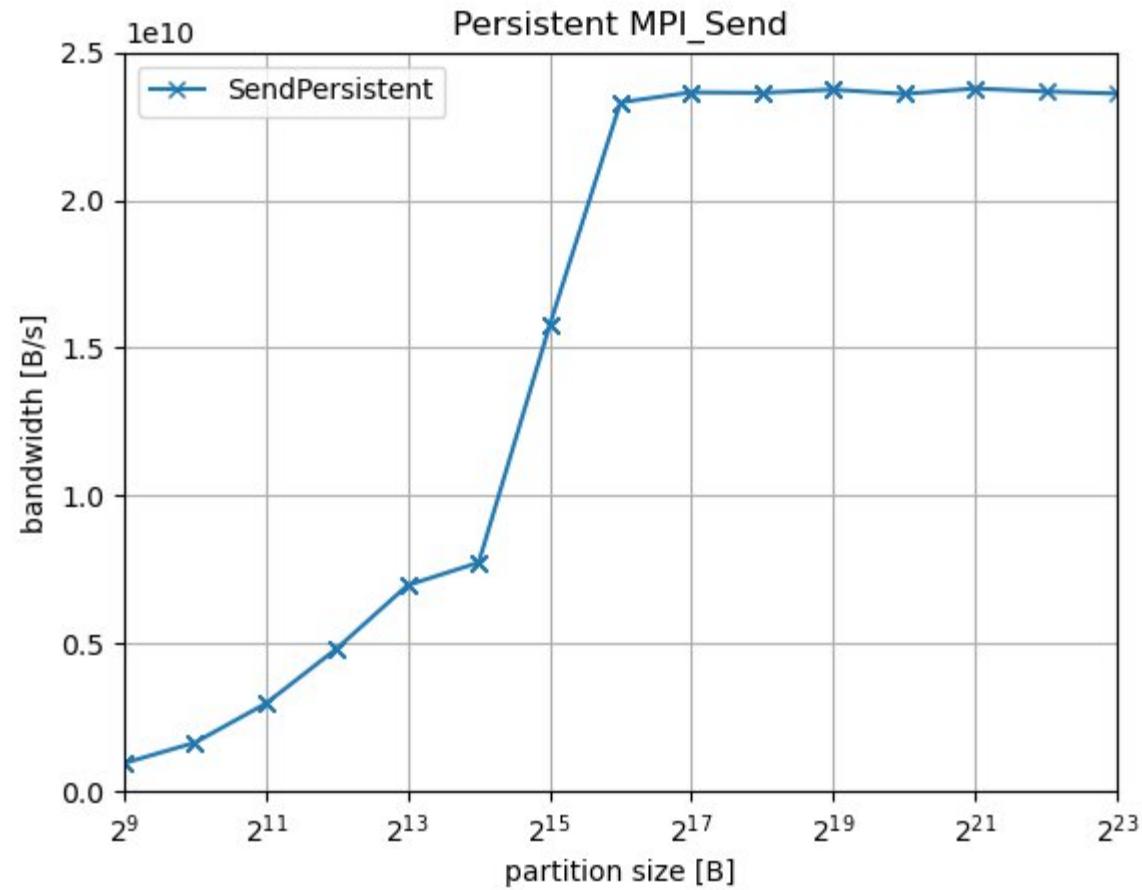
## Results: Blocking Send

H L R I S



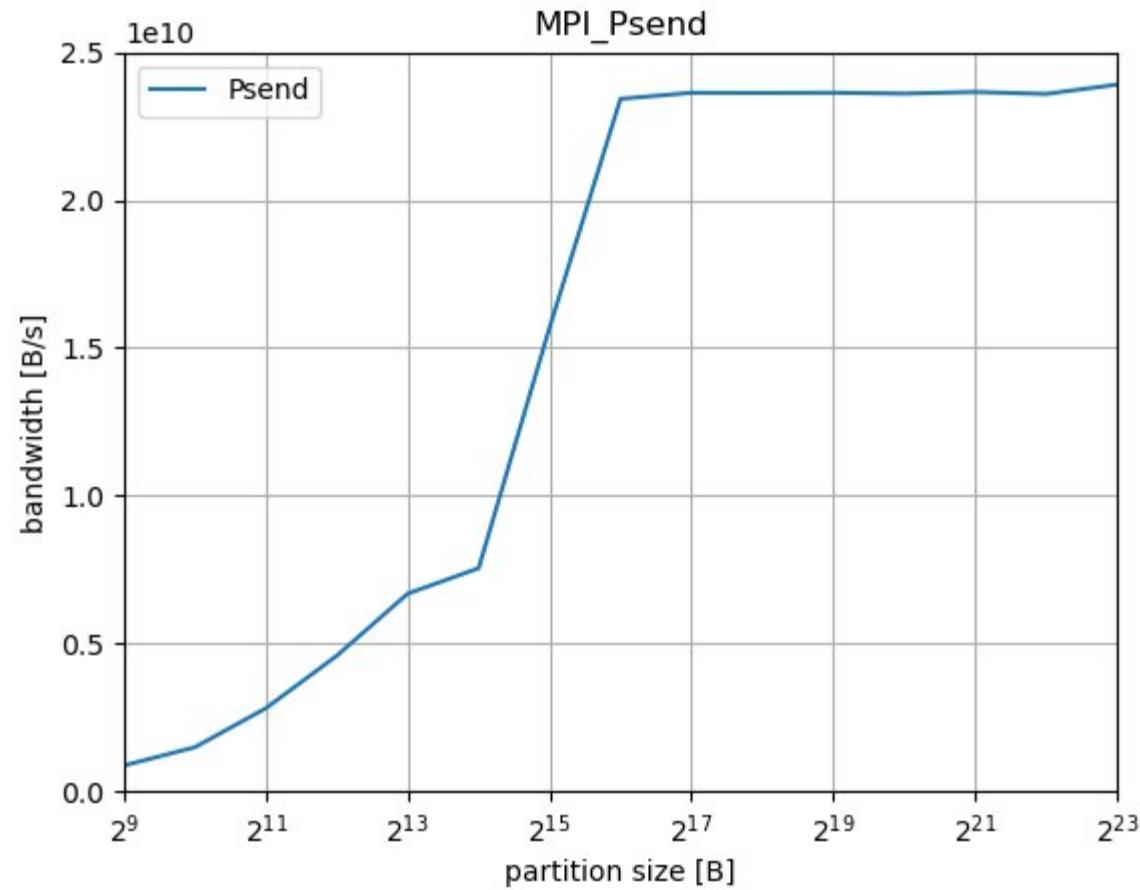
## Results: Persistent Send

H L R I S



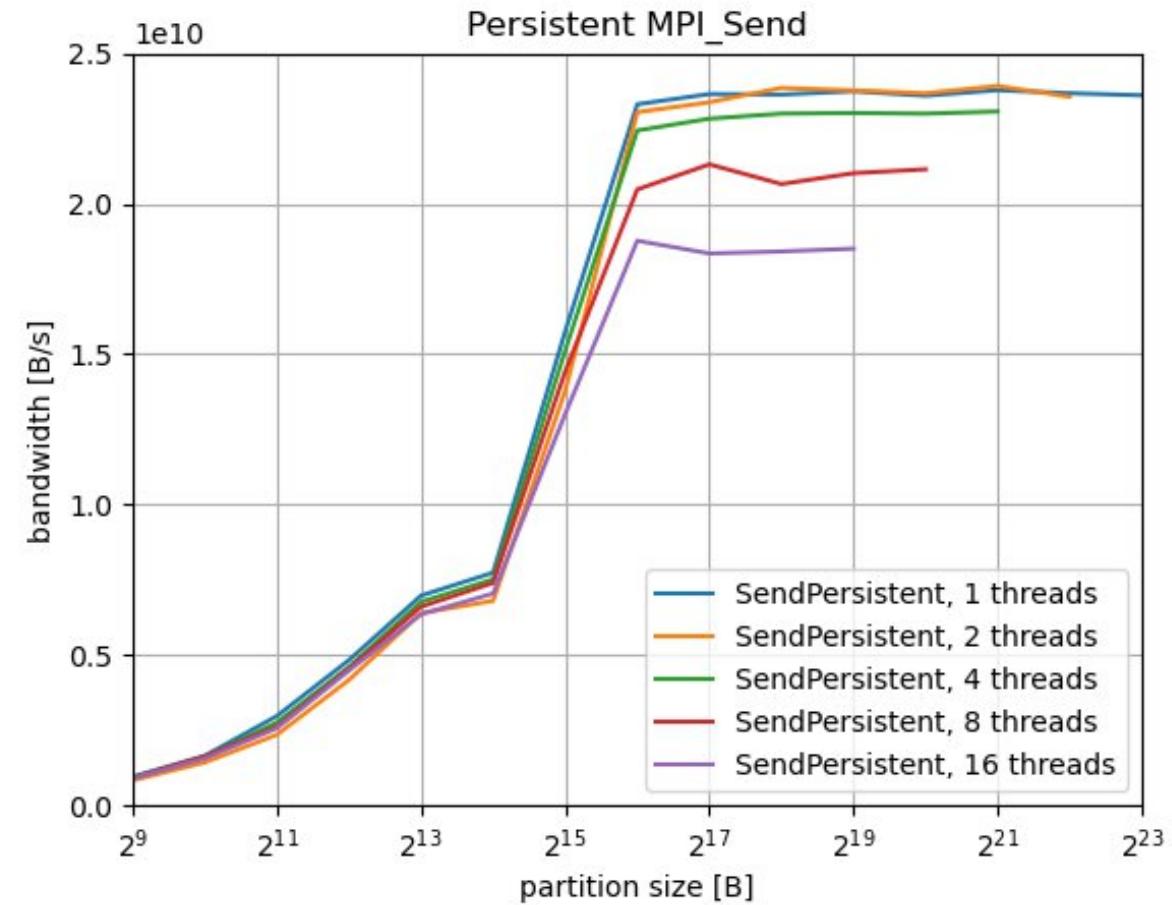
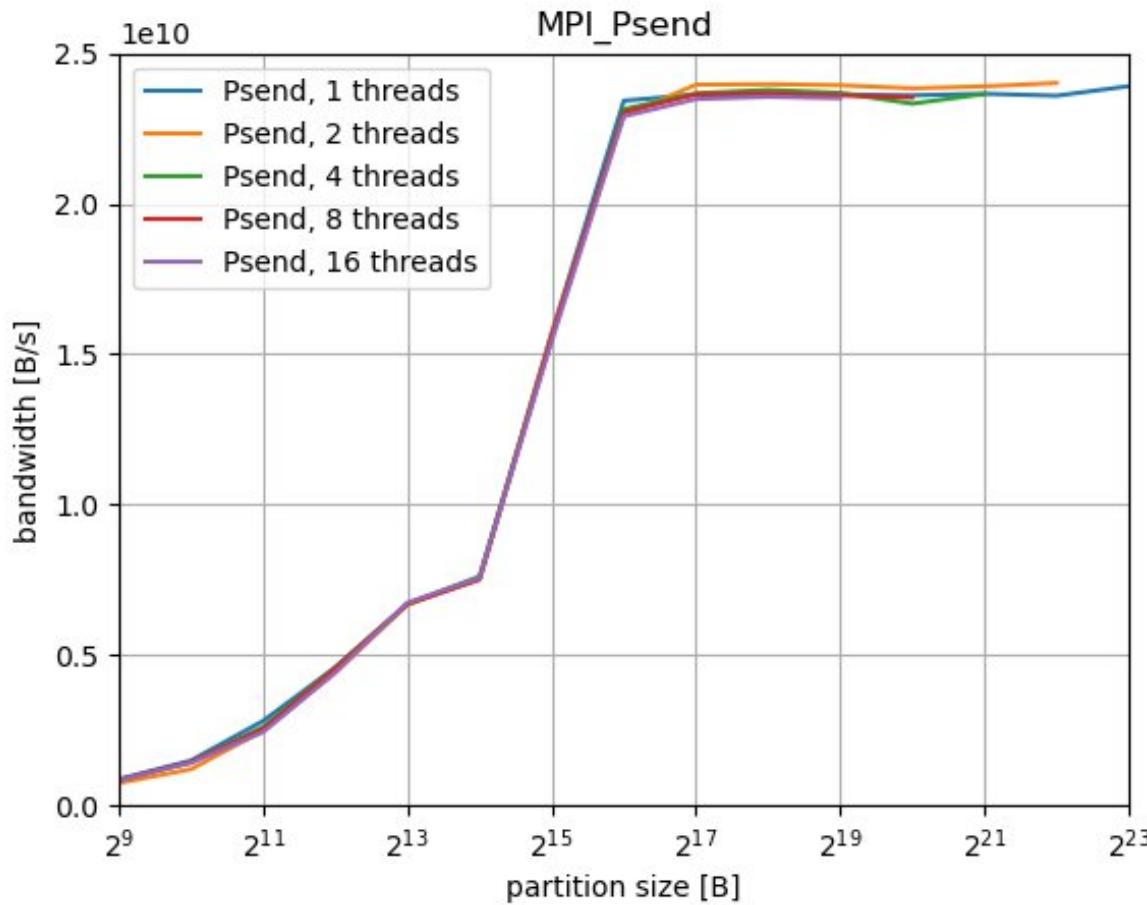
## Results: Psend

H L R I S



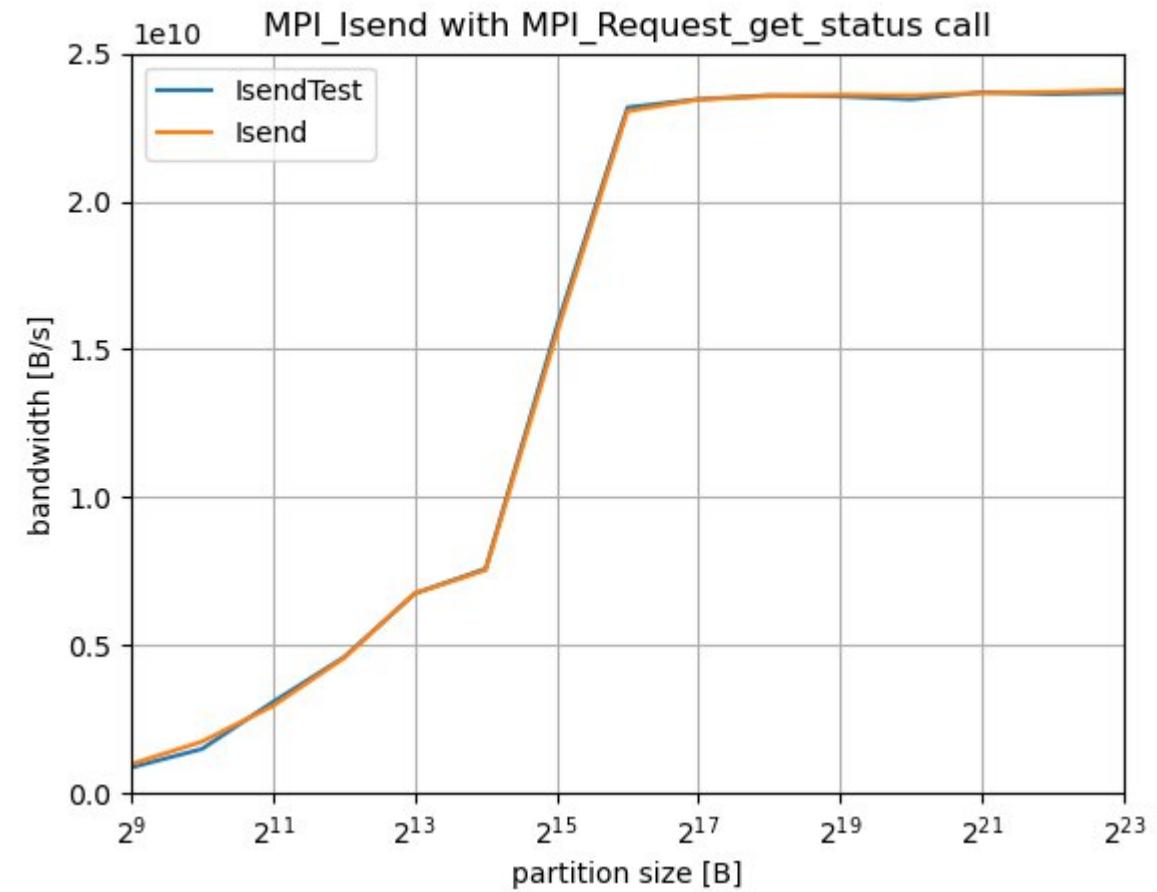
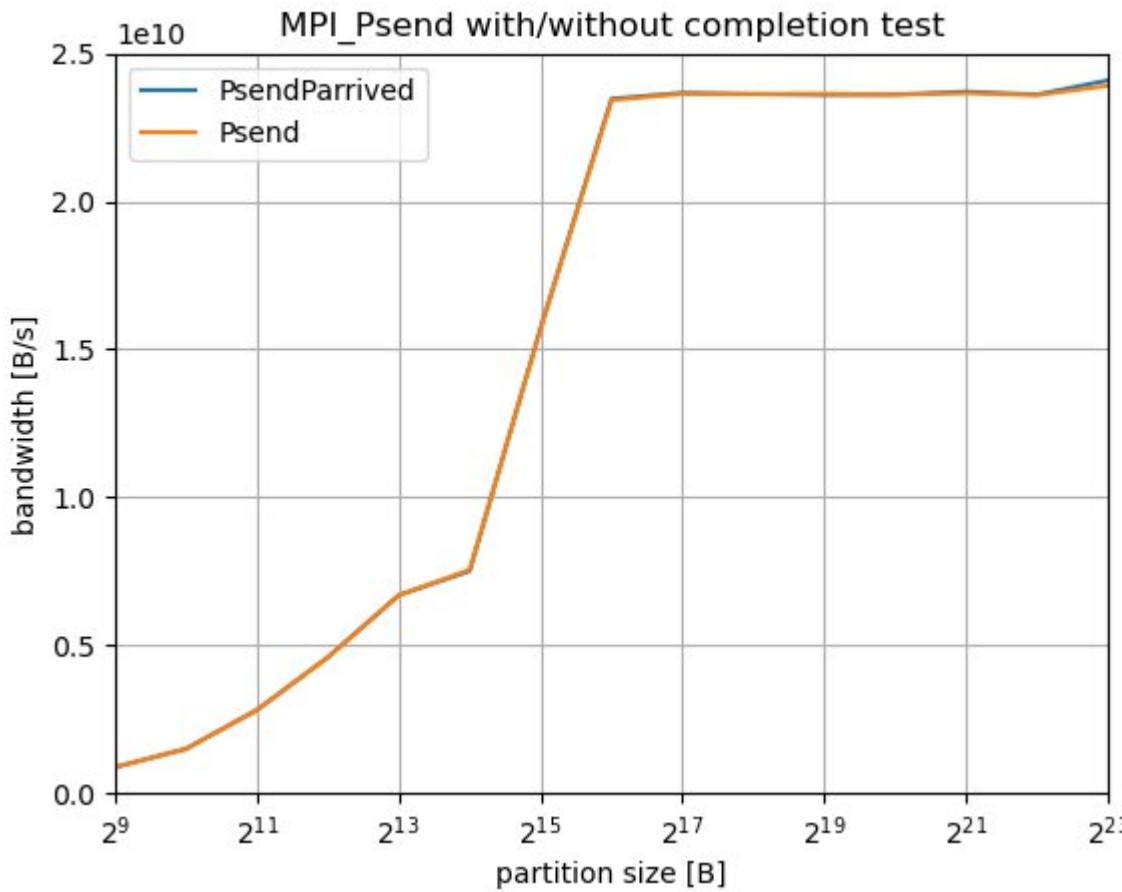
# Results: Persistent Send vs Psend (multithreaded)

H L R S



# Results: Completion Testing

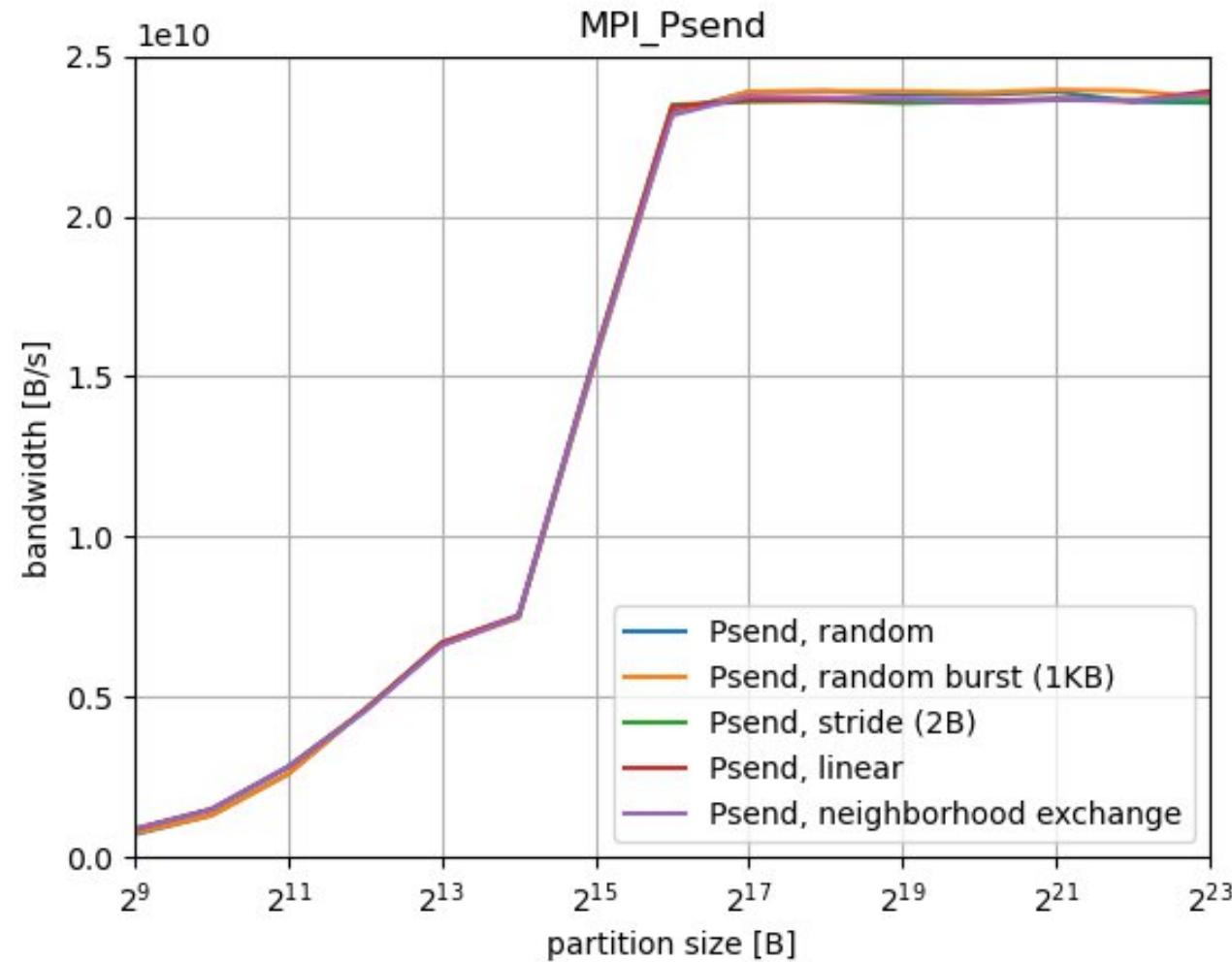
H L R T S



- Receive-Side completion test with MPI\_Parrived
- Send-Side completion testing with MPI\_Request\_get\_status crashes program

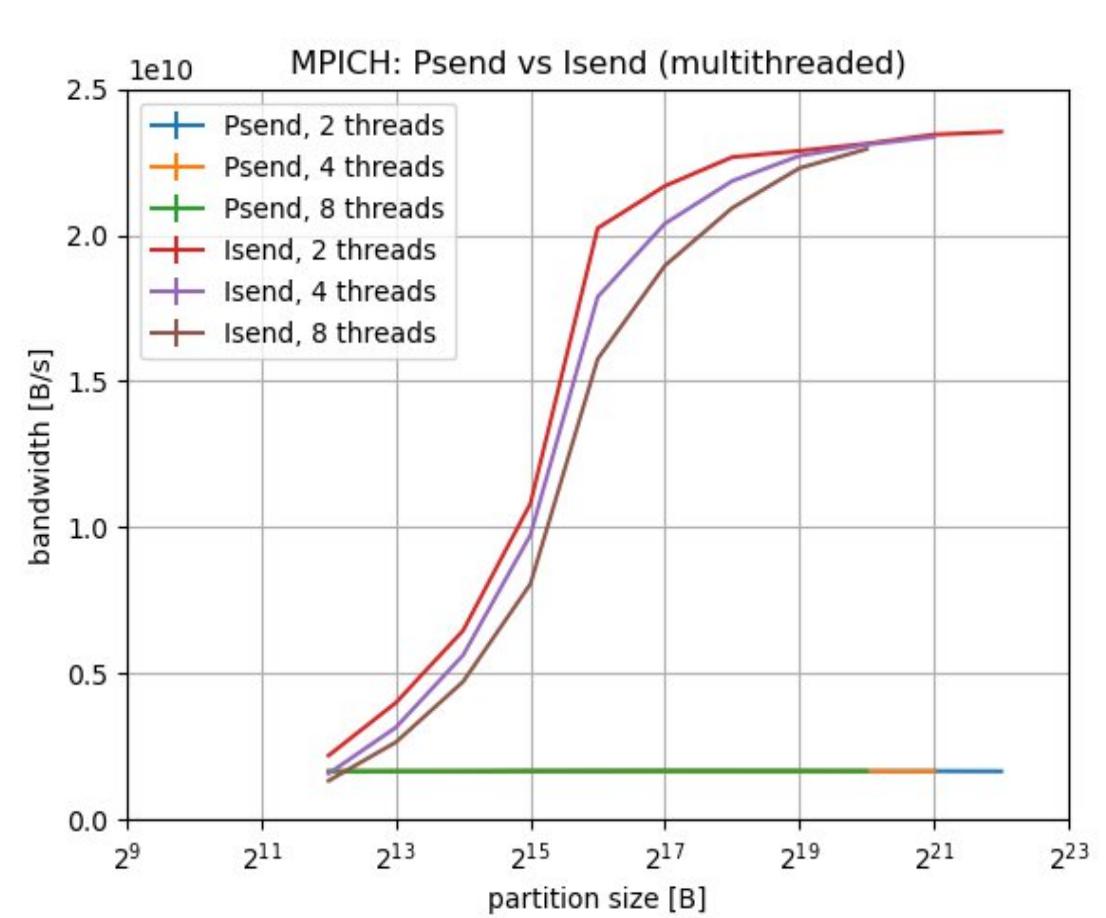
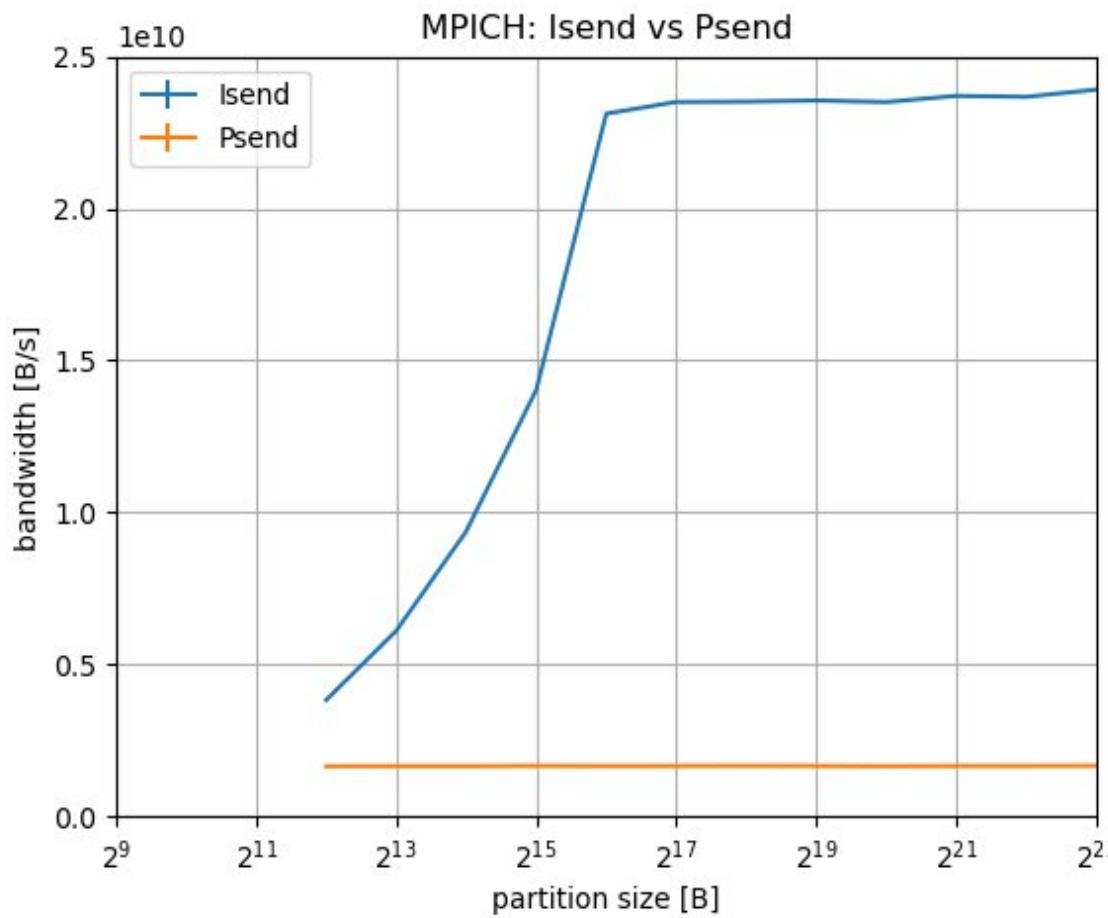
## Results: Psend with different send patterns

H L R S



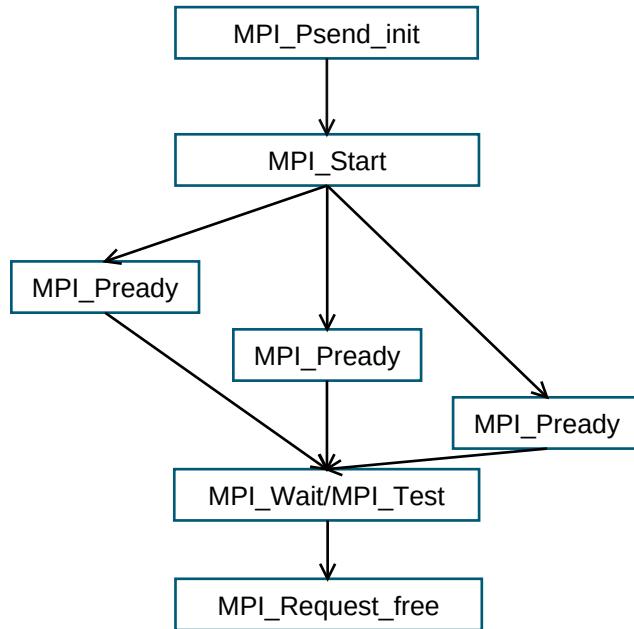
# Results: Psend on MPICH-4.1.2

H L R I S



# Current implementations: OpenMPI/5.0.2

H L R I S



**MPI\_Start():**

```
for p in 0...partition_count-1:  
    flag[p] = 0;
```

**MPI\_Pready(p):**

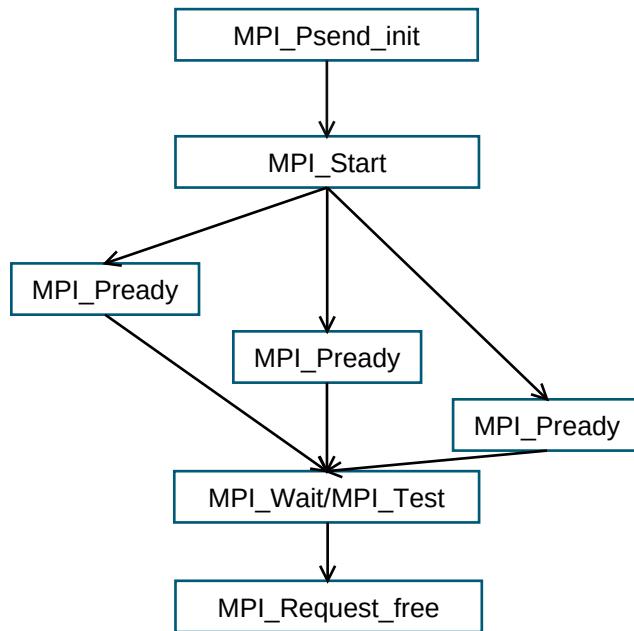
```
flag[p] = 1;
```

**progress():**

```
for p with flag[p] == 1:  
    isend(request[p]);  
    flag[p] = 2;
```

# Current implementations: MPICH/4.2.0

H L R I S



**MPI\_Start():**  
counter = partition\_count;

**MPI\_Pready():**  
counter—;  
  
if counter == 0:  
 send(buffer);

# Naive method of message aggregation

H L R I S

Public partitions:

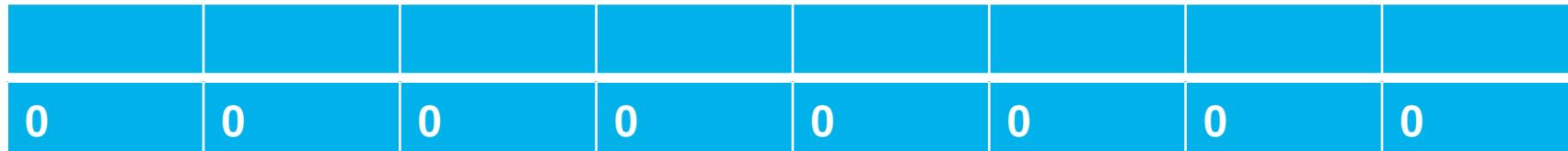


map m partitions to 1 internal partition

Internal partitions:



counters:

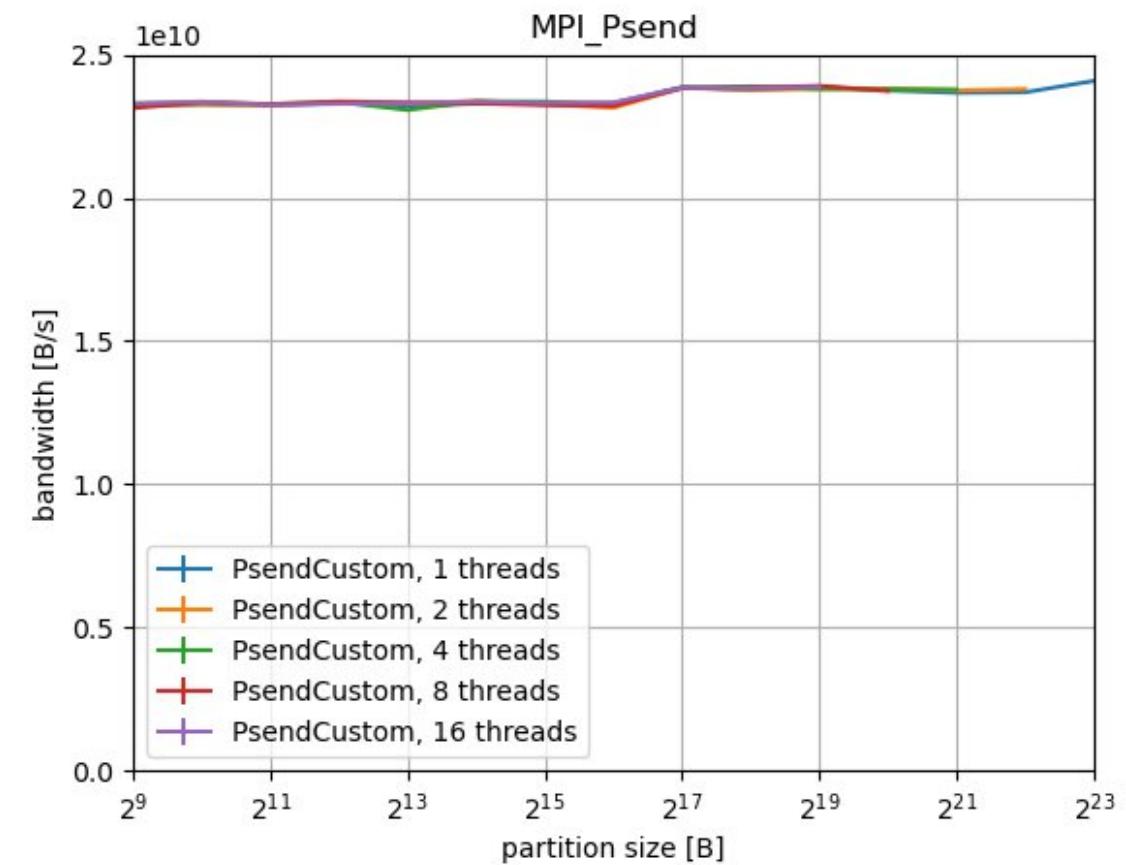
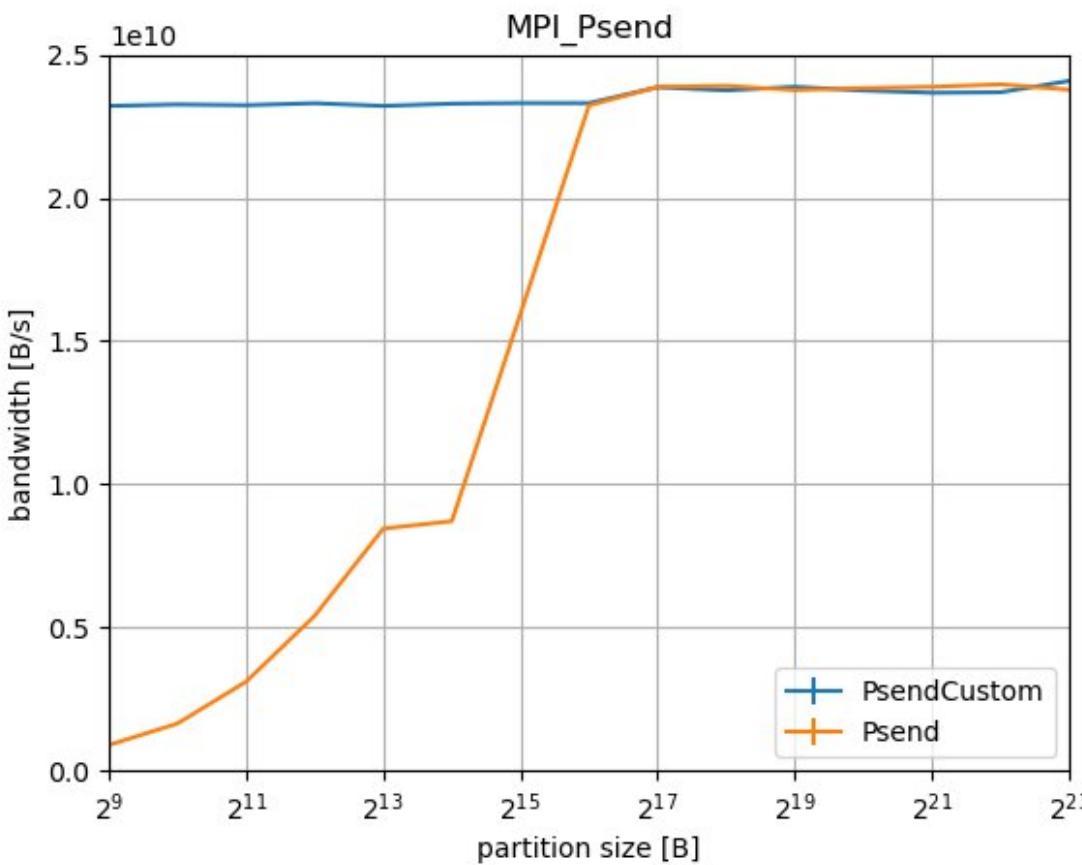


**Pready(p):**

increment counter of internal partition of p  
if counter  $\geq m$ :  
mark internal partition as ready

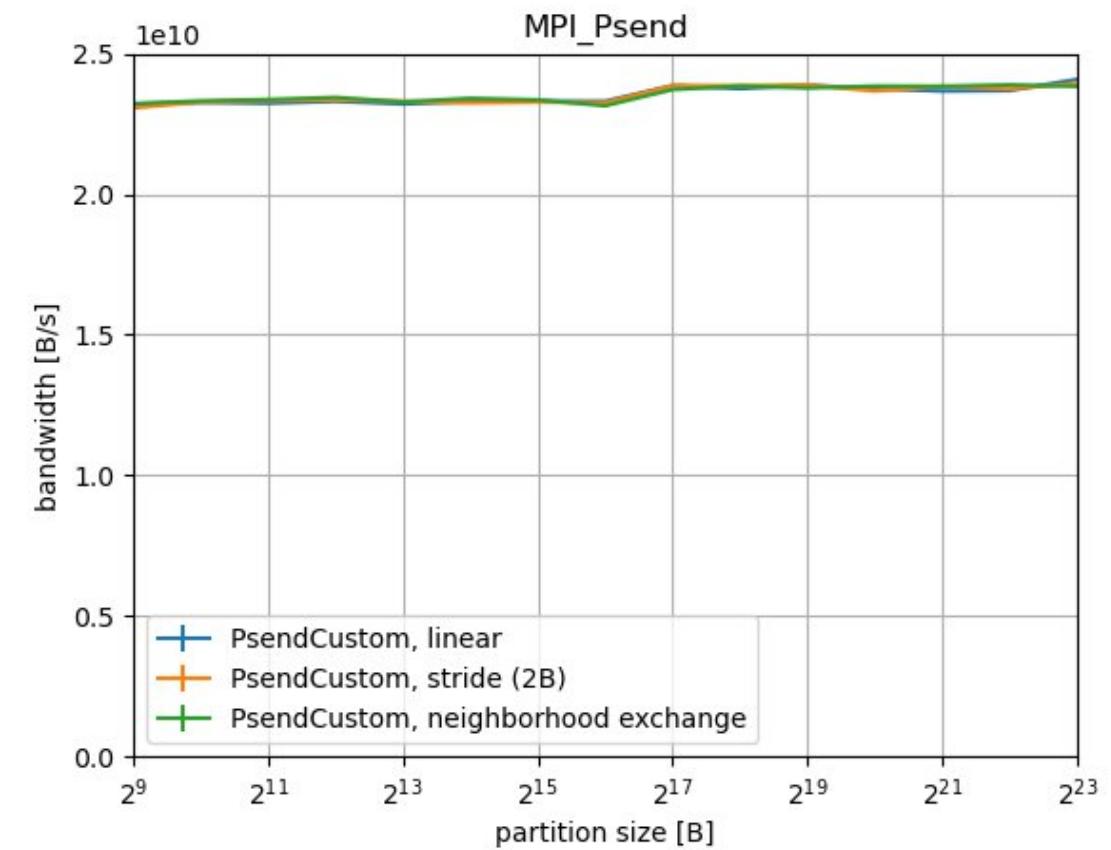
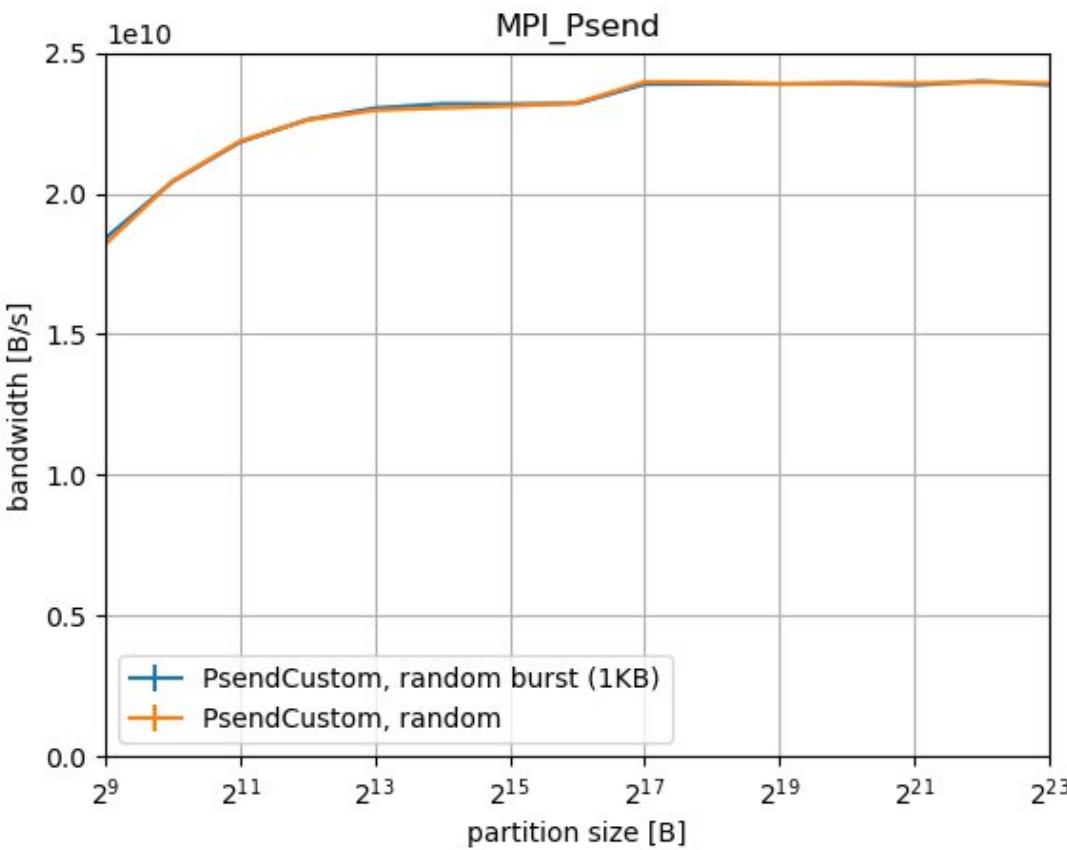
# Results: Naive message aggregation

H L R I S



# Results: Naive message aggregation

H L R I S



# Conclusion

H L R I S

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - Psend implemented by performing single transfer after all Pready-calls
  - Psend performs worse than other mechanisms
- Message aggregation:
  - Mapping partitions to larger messages can improve performance

# Conclusion

H L R I S

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - Psend implemented by performing single transfer after all Pready-calls
  - Psend performs worse than other mechanisms
- Message aggregation:
  - Mapping partitions to larger messages can improve performance

# Conclusion

H L R I S

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - Psend implemented by performing single transfer after all Pready-calls
  - Psend performs worse than other mechanisms
- Message aggregation:
  - Mapping partitions to larger messages can improve performance

# Conclusion

H L R I S

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - Psend implemented by performing single transfer after all Pready-calls
  - Psend performs worse than other mechanisms
- Message aggregation:
  - Mapping partitions to larger messages can improve performance

# Conclusion

H L R I S

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - Psend implemented by performing single transfer after all Pready-calls
  - Psend performs worse than other mechanisms
- Message aggregation:
  - Mapping partitions to larger messages can improve performance

# Conclusion

H L R I S

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - Psend implemented by performing single transfer after all Pready-calls
  - Psend performs worse than other mechanisms
- Message aggregation:
  - Mapping partitions to larger messages can improve performance

High-Performance  
Computing Center  
Stuttgart

Thanks for your attention!

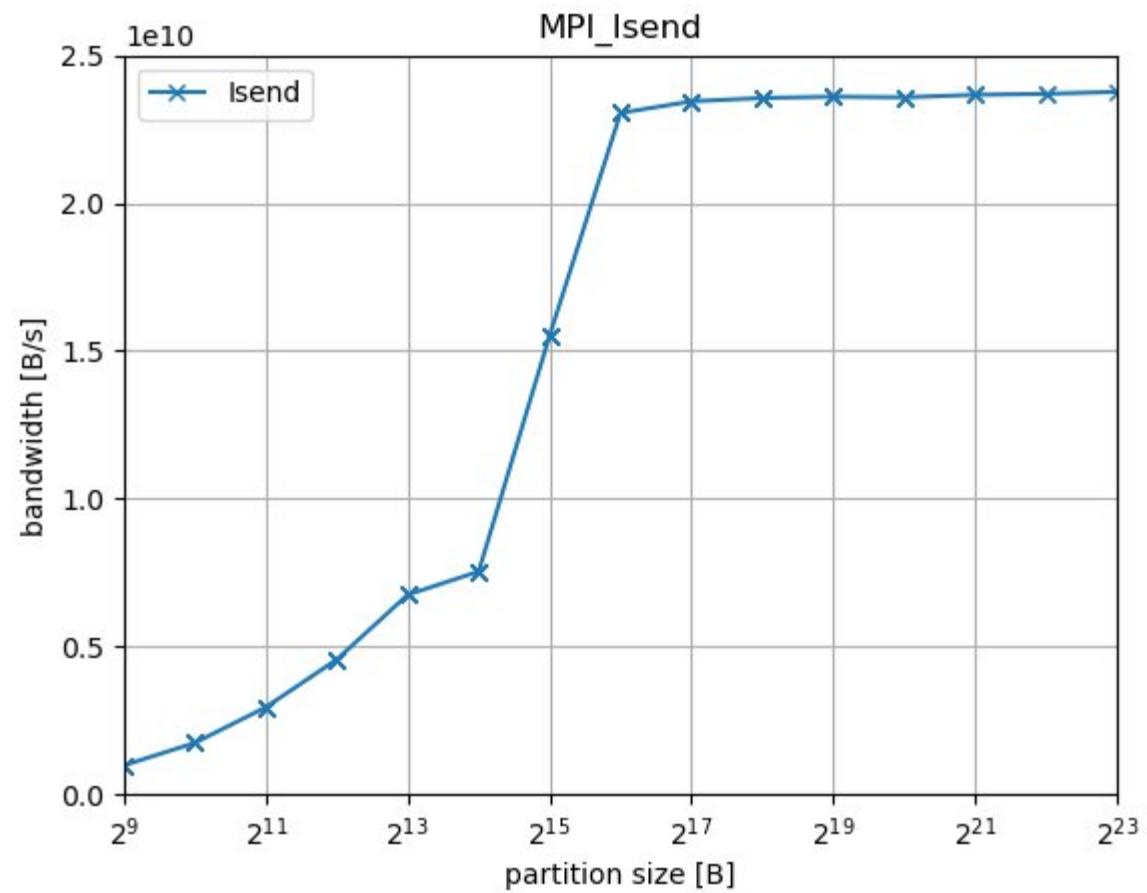
# Appendix – OpenMPI progress

H L R I S

```
ompi > mca > part > persist > C part_persist.h
162
223 AL_LIST_FOREACH(current, ompi_part_persist.progress_list, mca_part_persist_list_t) {
287     if(false == req->req_part_complete && REQUEST_COMPLETED != req->req_ompi.req_complete && OMPI_REQU
288         for(i = 0; i < req->real_parts; i++) {
289
290             /* Check to see if partition is queued for being started. Only applicable to sends. */
291             if(-2 == req->flags[i]) {
292                 err = req->persistent_reqs[i]->req_start(1, (&(req->persistent_reqs[i])));
293                 req->flags[i] = 0;
294             }
295
296             if(0 == req->flags[i])
297             {
298                 ompi_request_test(&(req->persistent_reqs[i]), &(req->flags[i]), MPI_STATUS_IGNORE);
299                 if(0 != req->flags[i]) req->done_count++;
300             }
301         }
```

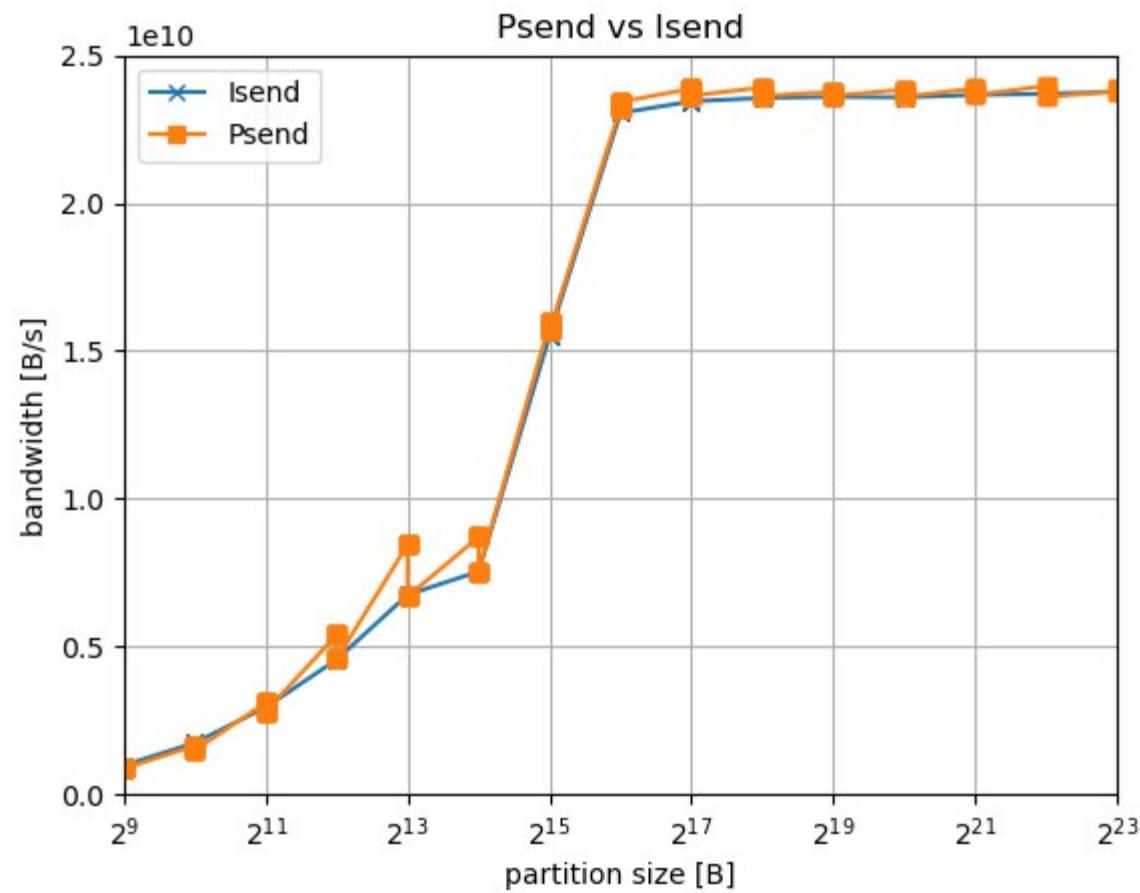
**I**send

T L R I S



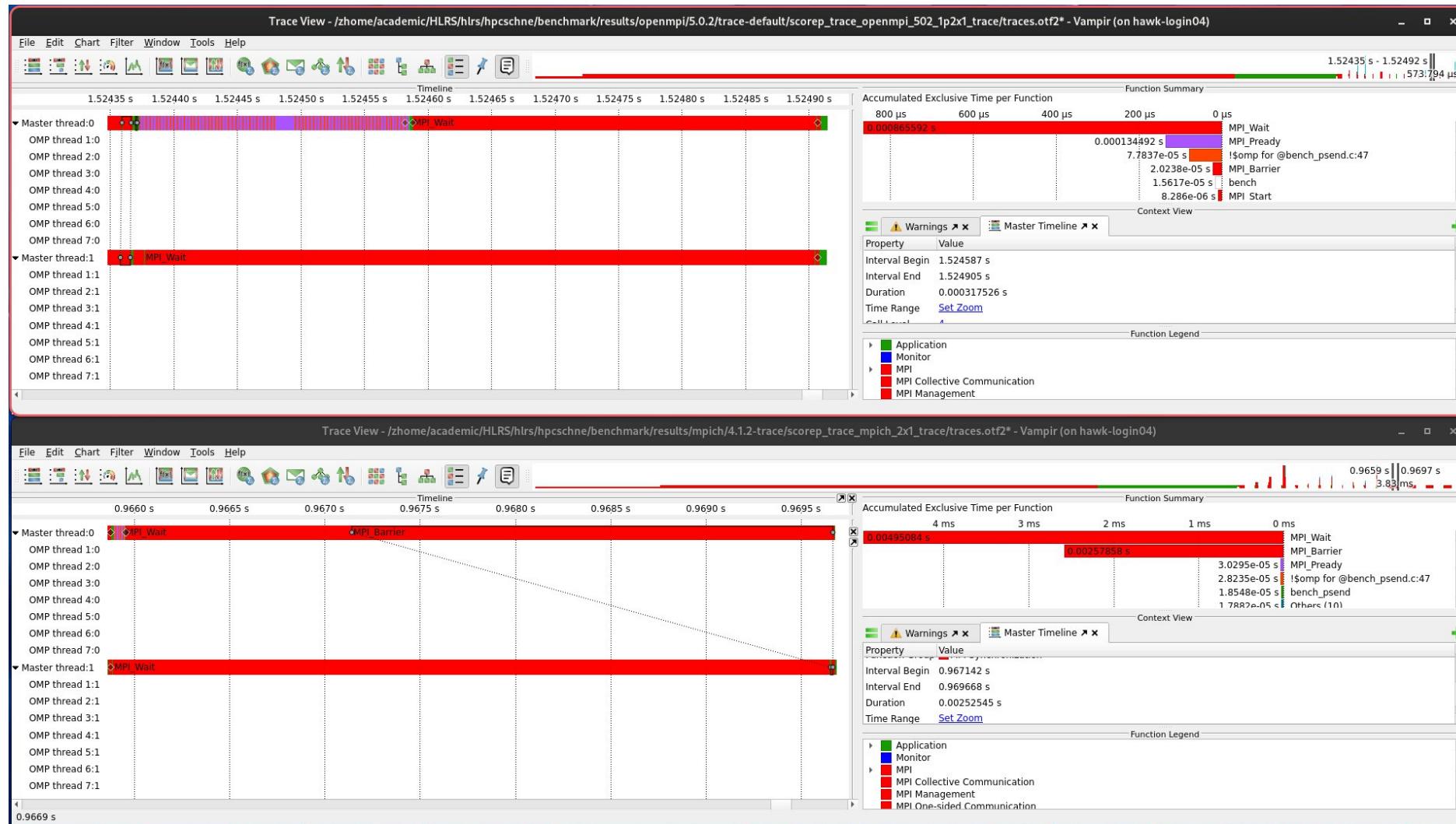
# I send vs P send

H L R S



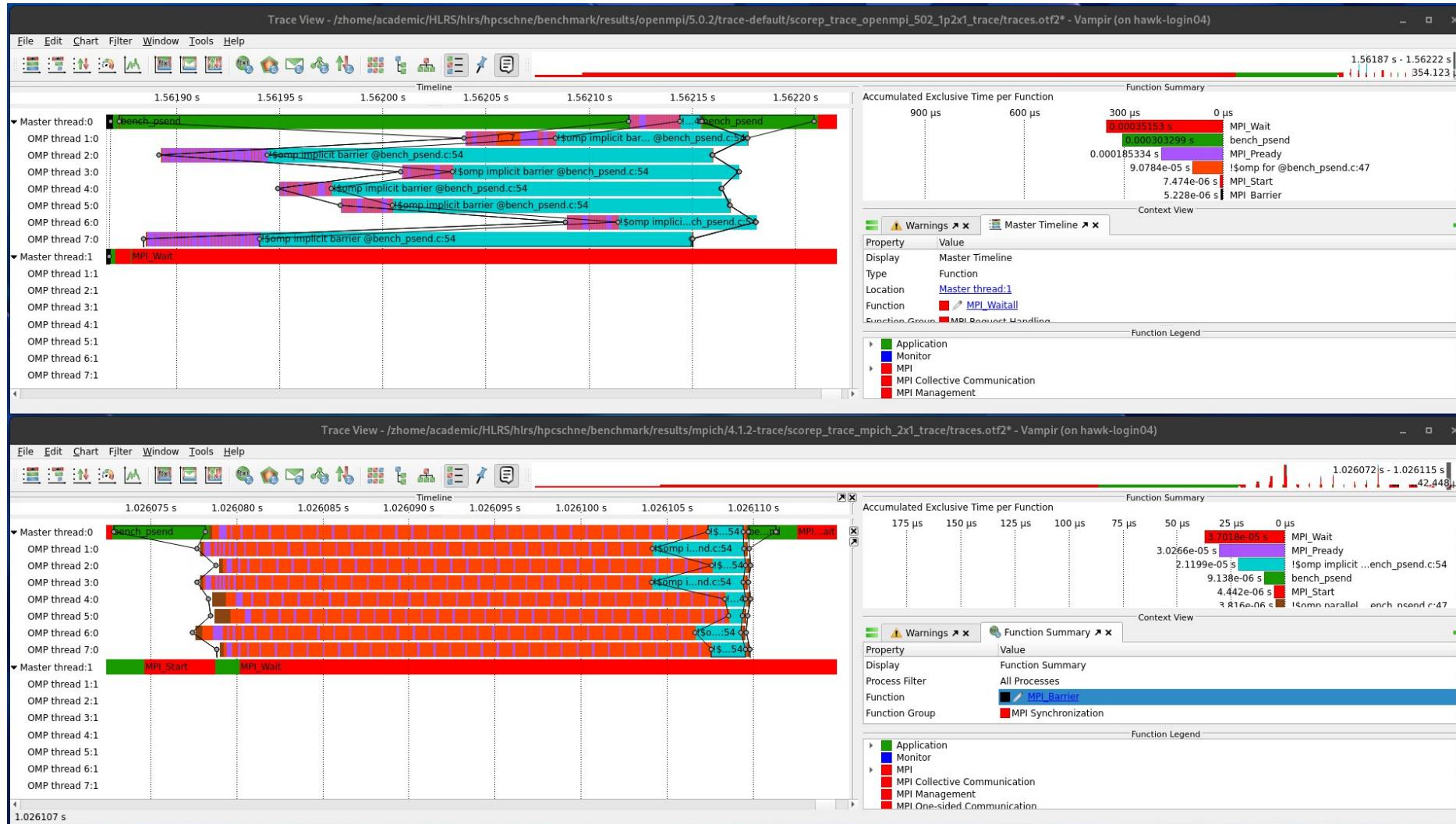
# Traces: MPI\_Psend

H L R S



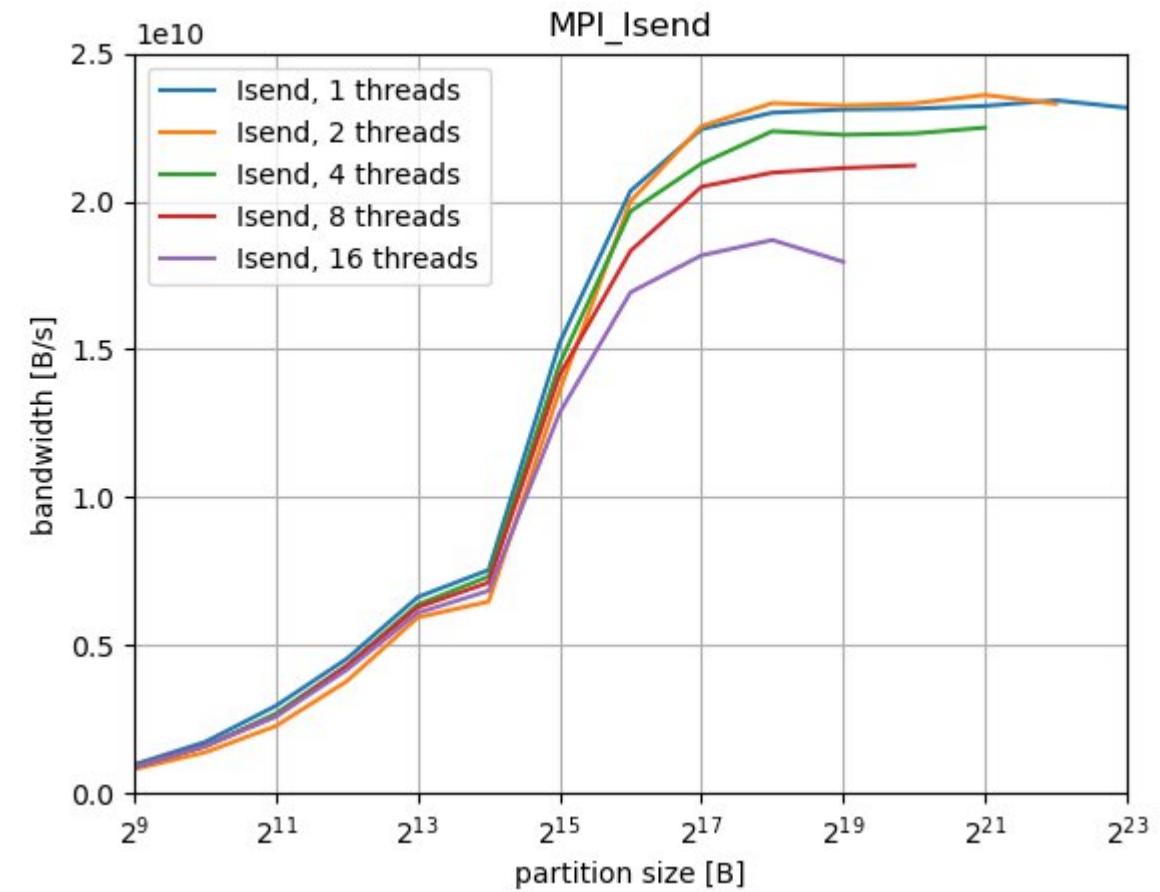
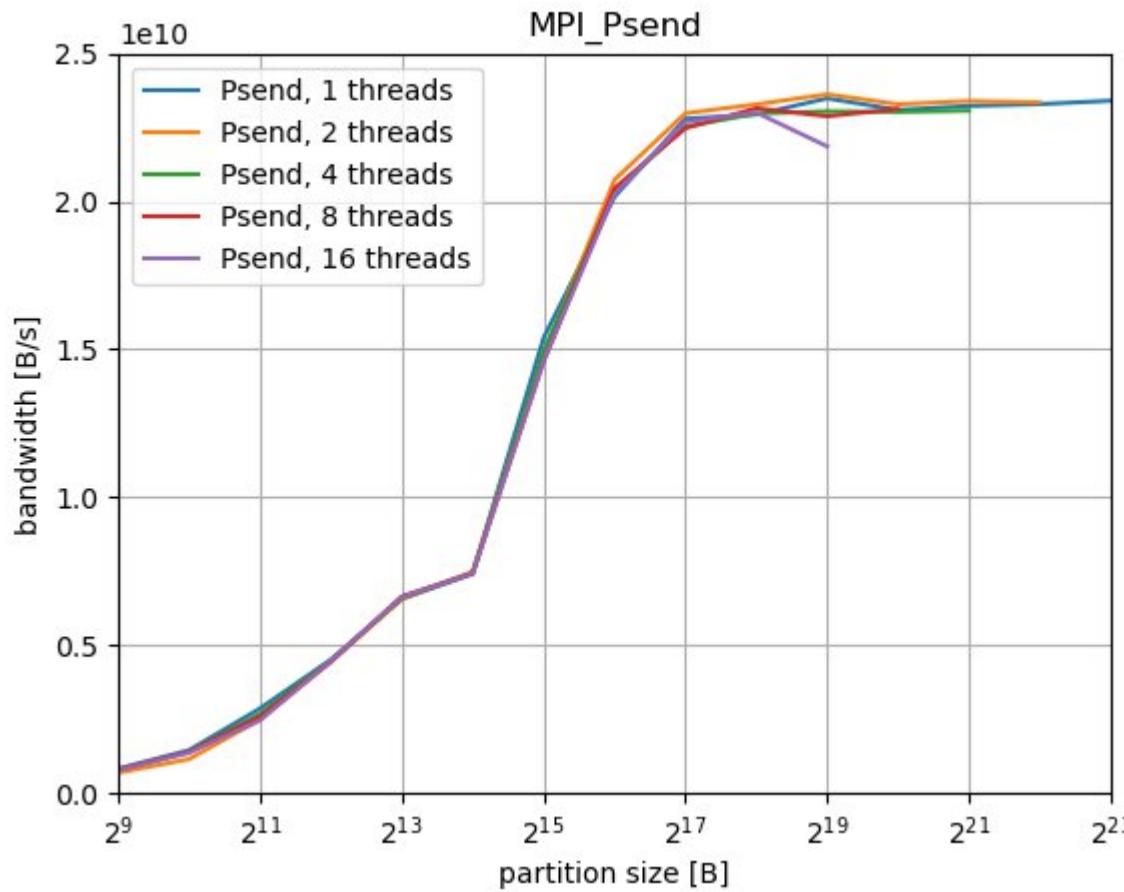
# Traces: MPI\_Psend – MPI\_Pready-Calls

H L R I S



## Appendix: Isend vs Psend (multithreaded)

H L R T S



# Results: MPI\_Win

H L R I S

