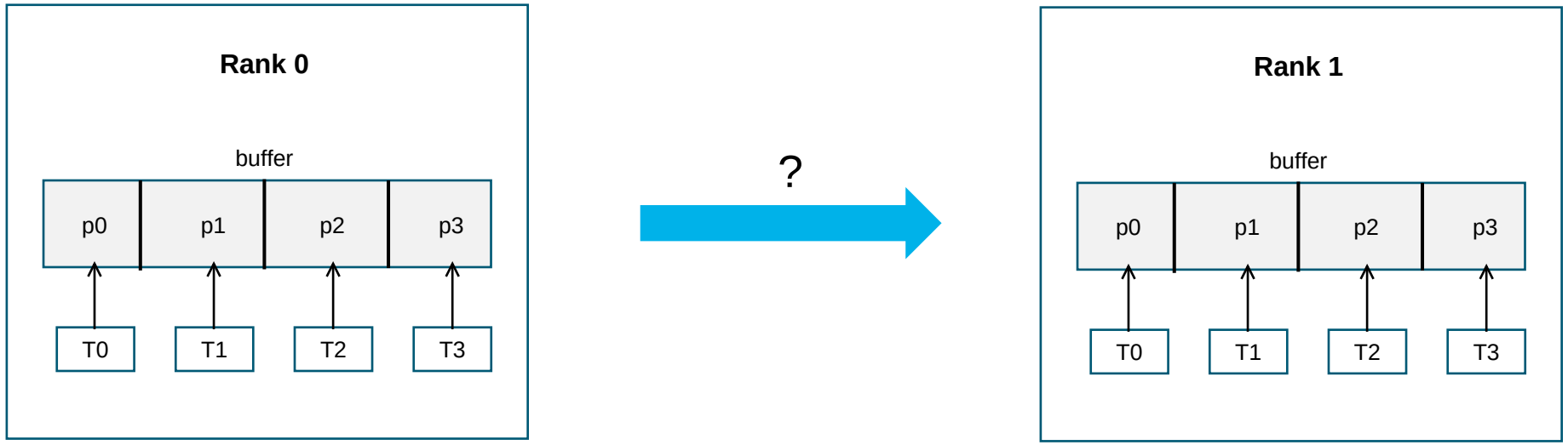
High-Performance
Computing Center
Stuttgart

Performance and Optimization opportunities of MPI partitioned communication

Axel Schneewind

Motivation: Hybrid Programming



Outline

- 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

- 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

- 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

- 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

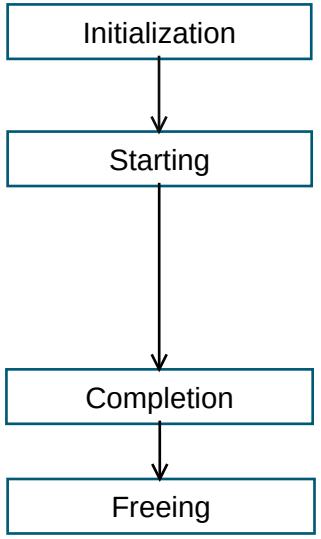
Outline

- 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

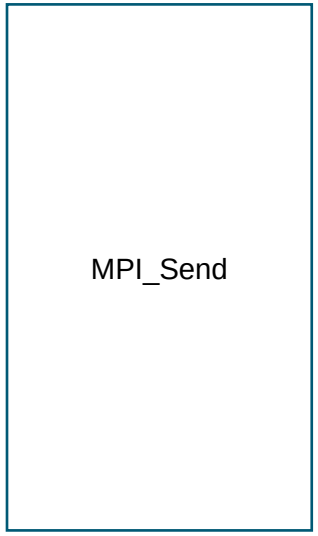
Outline

- 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

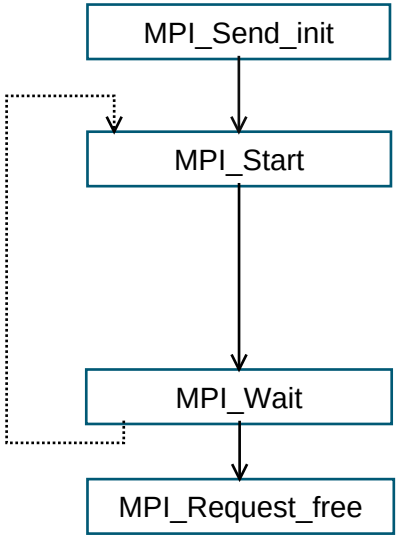
MPI: Transfer Mechanisms



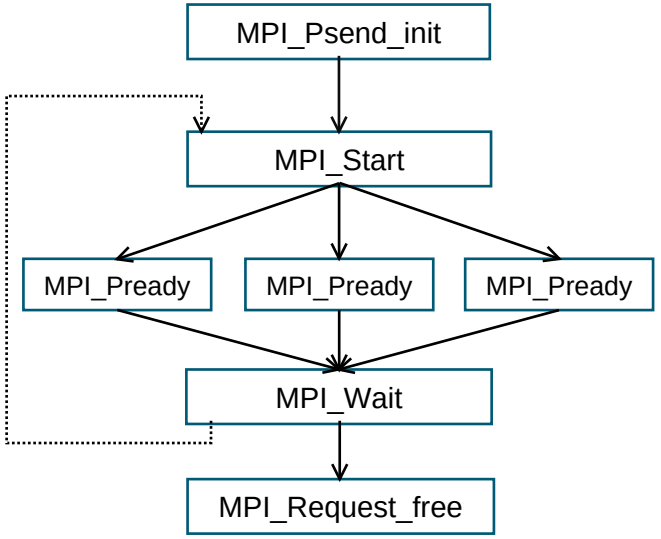
Transfer phases



Blocking Send



Persistent Send (similar: Nonblocking Isend)



Partitioned Send (since MPI-4.0)

MPI-4.1: Blocking/Nonblocking Send

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

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

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

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

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

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

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

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.*)

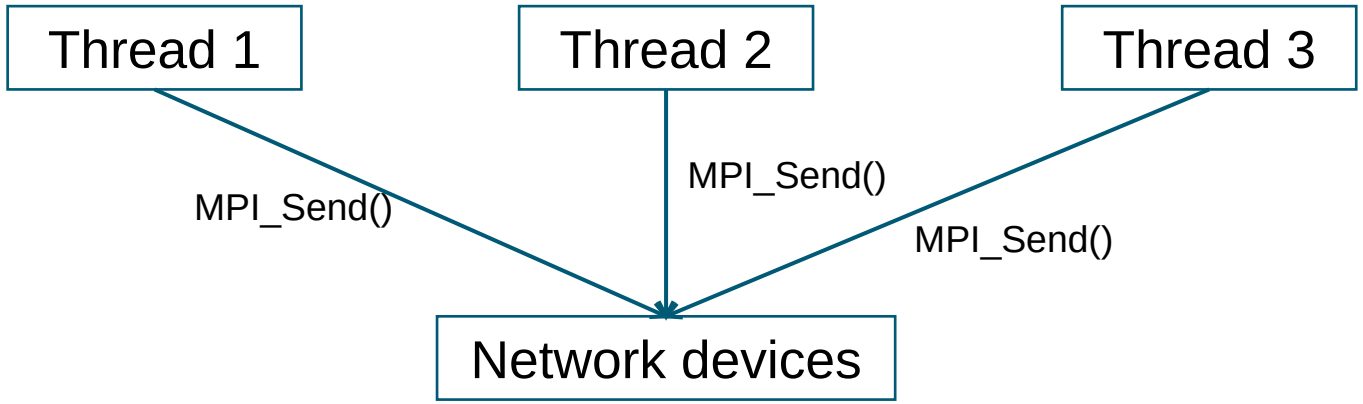
MPI-4.1: Partitioned Communication

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.

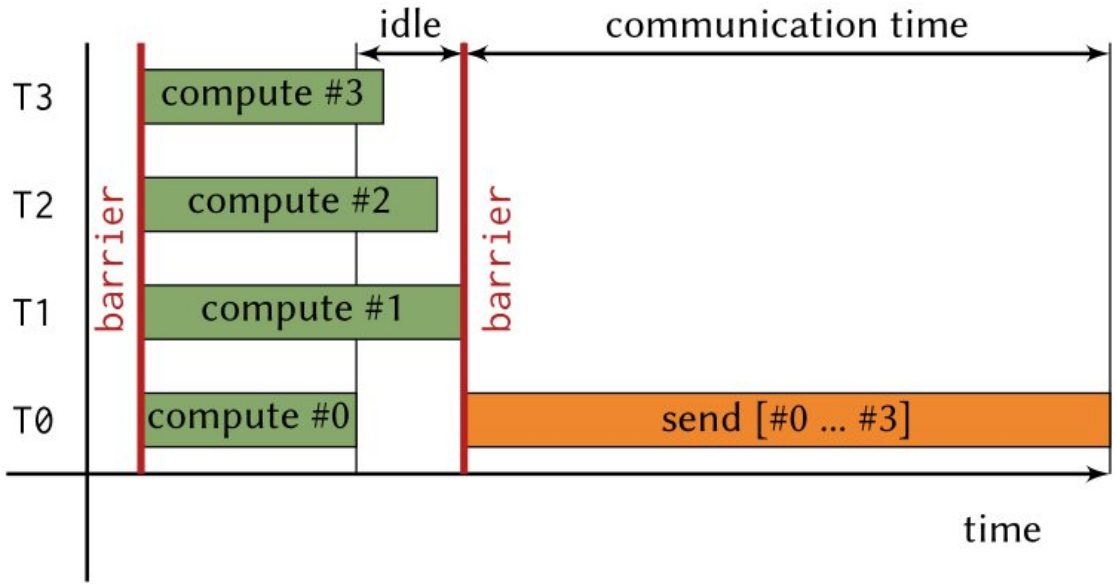
A naive implementation can simply wait for the entire message buffer to be marked ready before any transfer(s) occur and could wait until the completion function is called on a request before transferring data. [...]. (*End of rationale.*)

Possible performance benefits – Less locking overhead

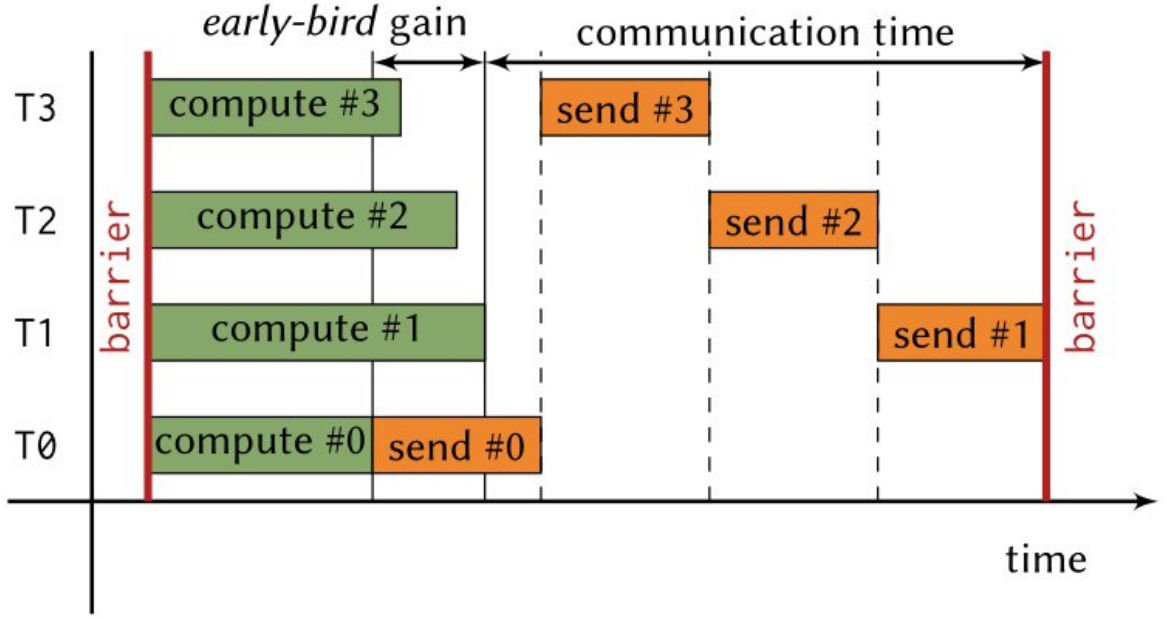


- Pready not required to interact with network devices
  - Can be called concurrently

Possible performance benefits – Early-bird effect



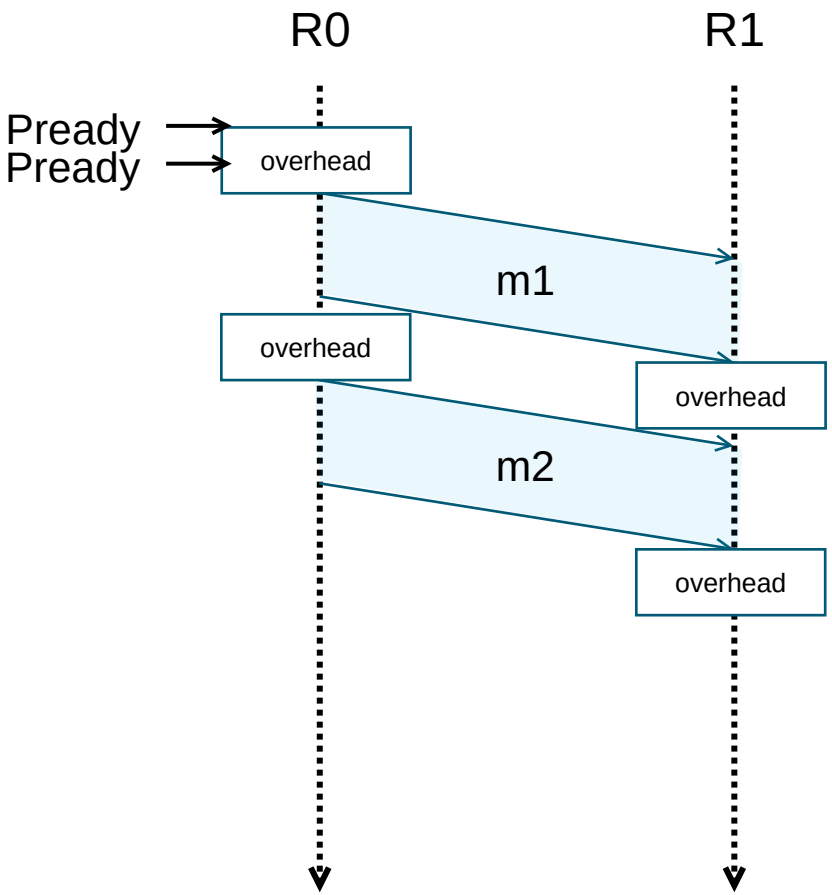
Single transfer for entire buffer

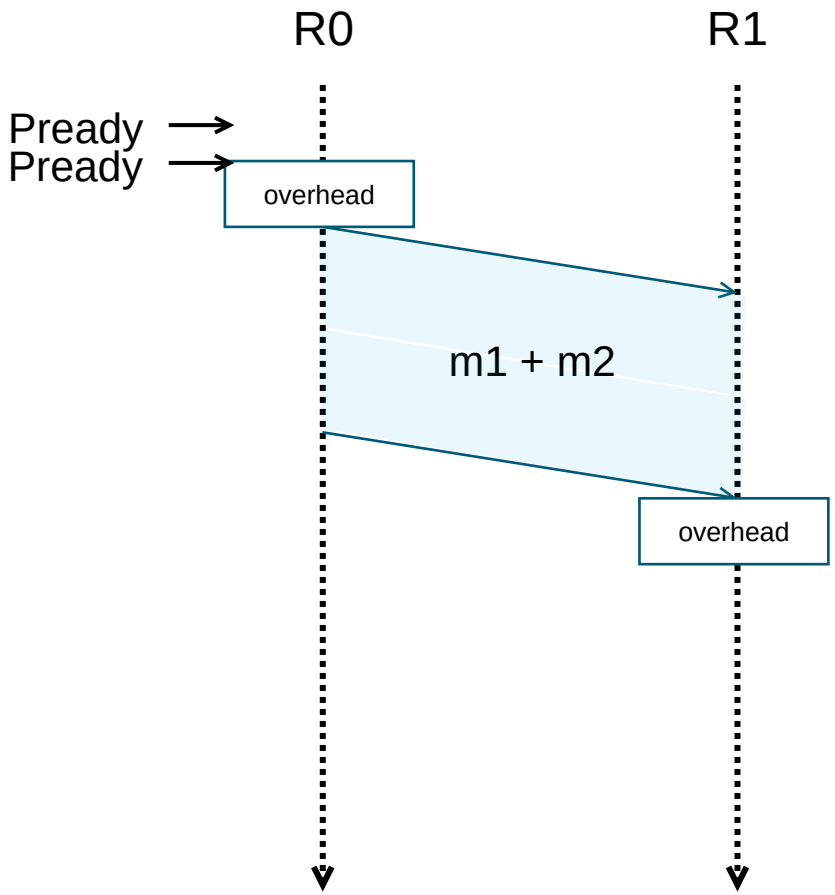


One transfer per partition

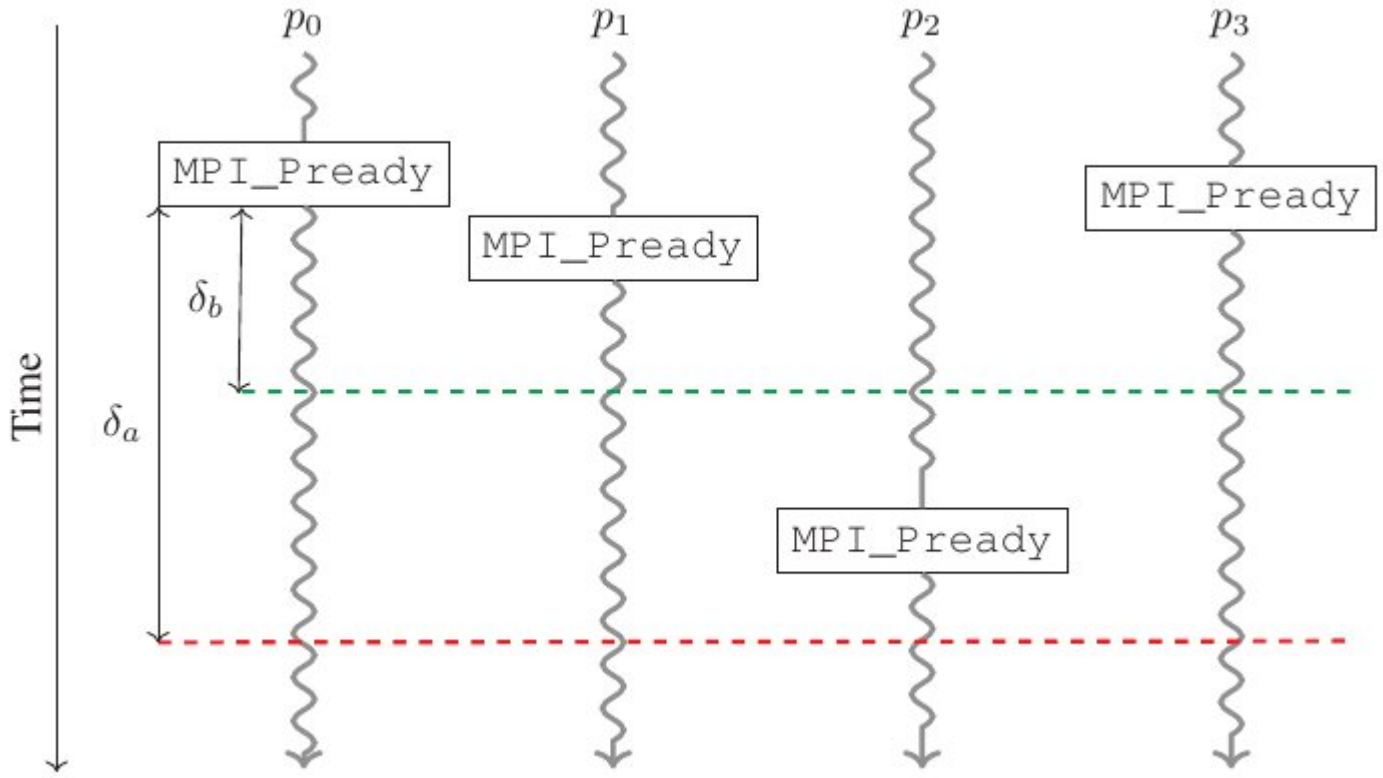
Thomas Gillis, Ken Raffenetti, Hui Zhou, Yanfei Guo, and Rajeev Thakur. 2023. Quantifying the Performance Benefits of Partitioned Communication in MPI.

Possible optimization – Message aggregation





Possible optimization – Message aggregation



Y. H. Temuçin, S. Levy, W. Schonbein, R. E. Grant and A. Afsahi, "A Dynamic Network-Native MPI Partitioned Aggregation Over InfiniBand Verbs," https://doi.org/10.1109/CLUSTER52292.2023.00029

Benchmarks: Goals

- 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

```
# initialization

for i in 0...num_iterations:
    MPI_Barrier()   # synchronization

    start_time[i] = MPI_Wtime()

    for p in 0...partition_count in some order:
        send/receive partition p        # start transfer

    MPI_Wait(request)           # completion

    end_time[i] = MPI_Wtime()

# cleanup

bandwidth = buffer_size / mean(start_time - end_time)
```

Transfer mechanisms

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

- 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

- 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

- 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

- 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

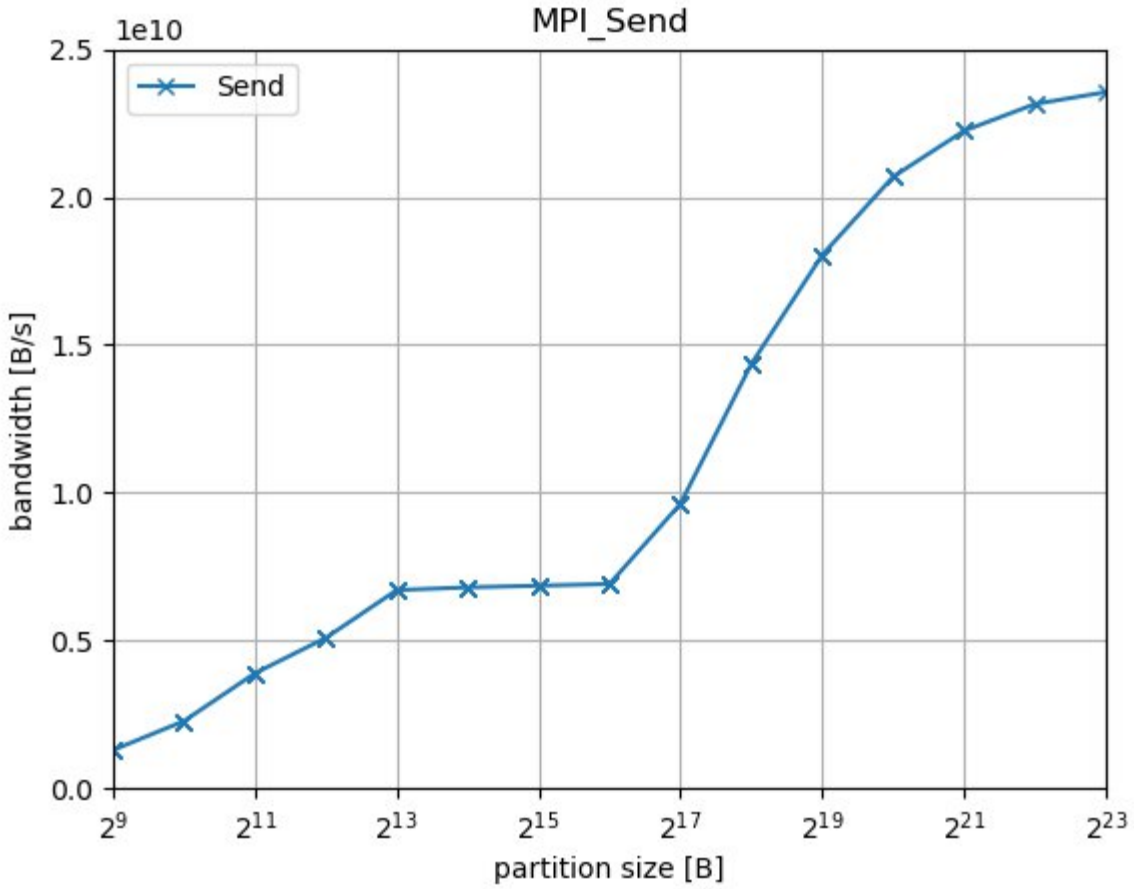
- 2 nodes on HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s

- **Buffer size:** 8MiB
- **Partition sizes**: 512B, 1024B, ..., 8MiB

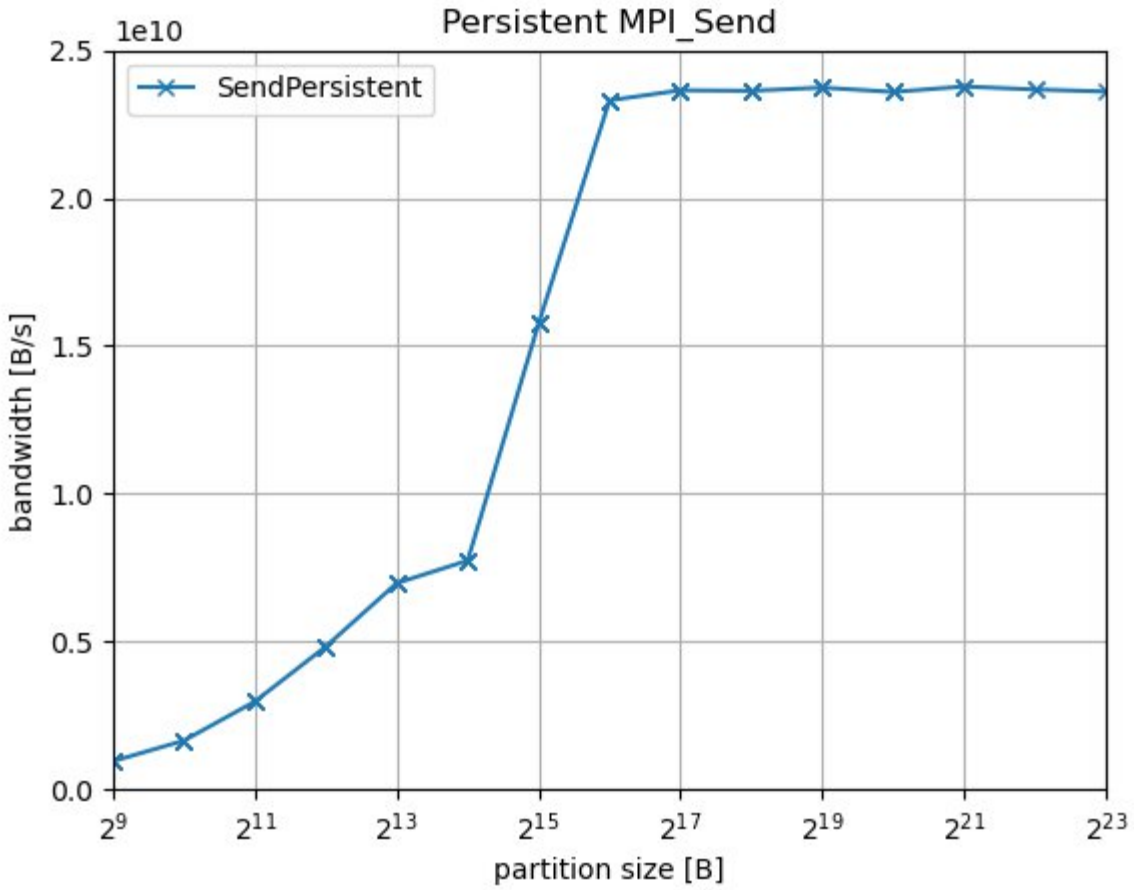
- Partitions marked ready in different orders:
  - Left-to-right
  - Randomized
  - Neighbourhood exchange pattern

- Multithreading with OpenMP

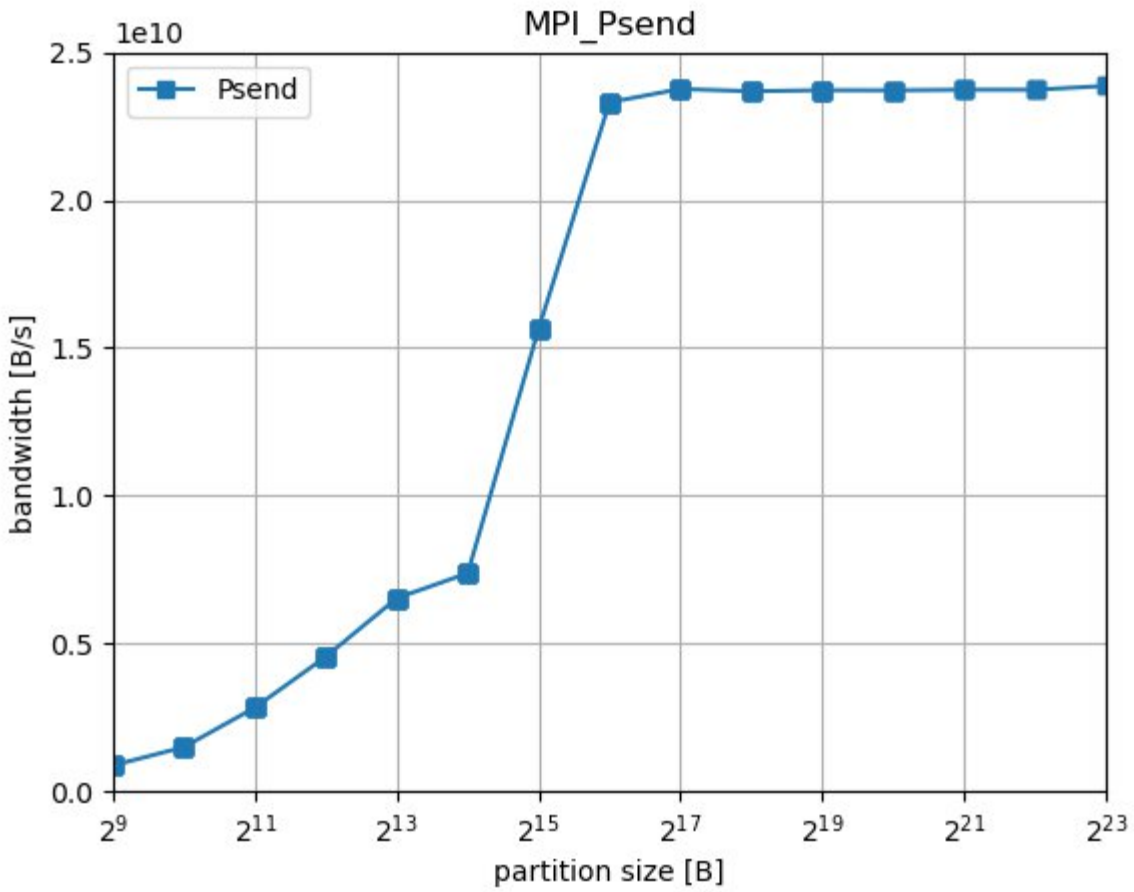
Results: Blocking Send



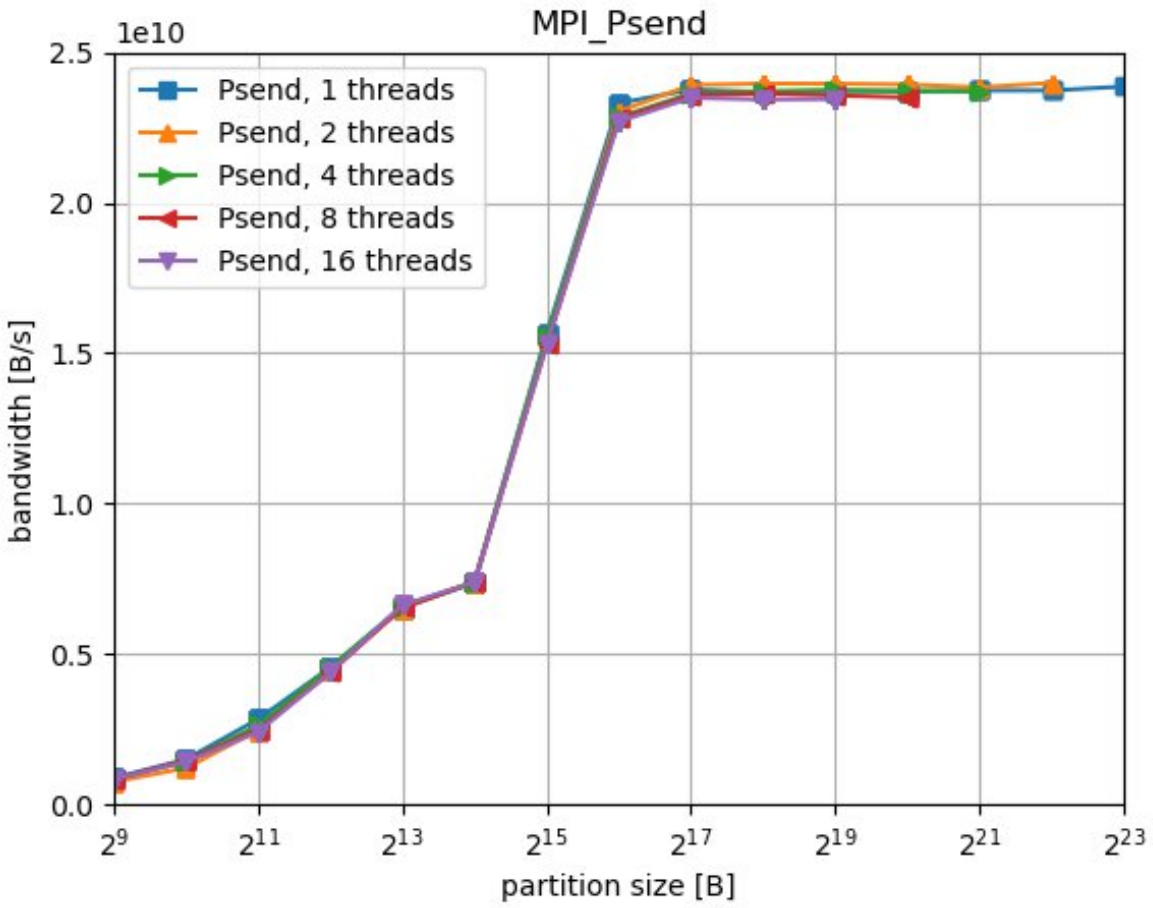
Results: Persistent Send

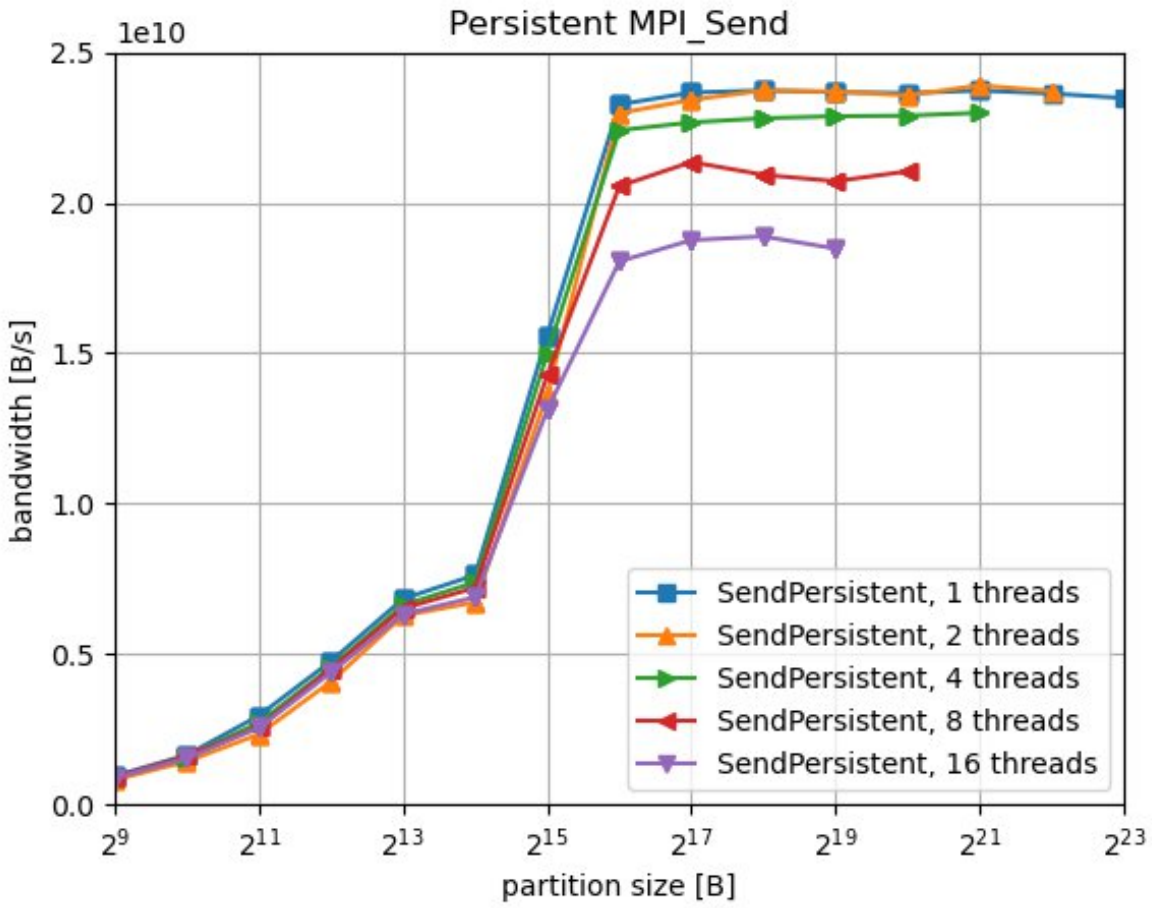


Results: Psend

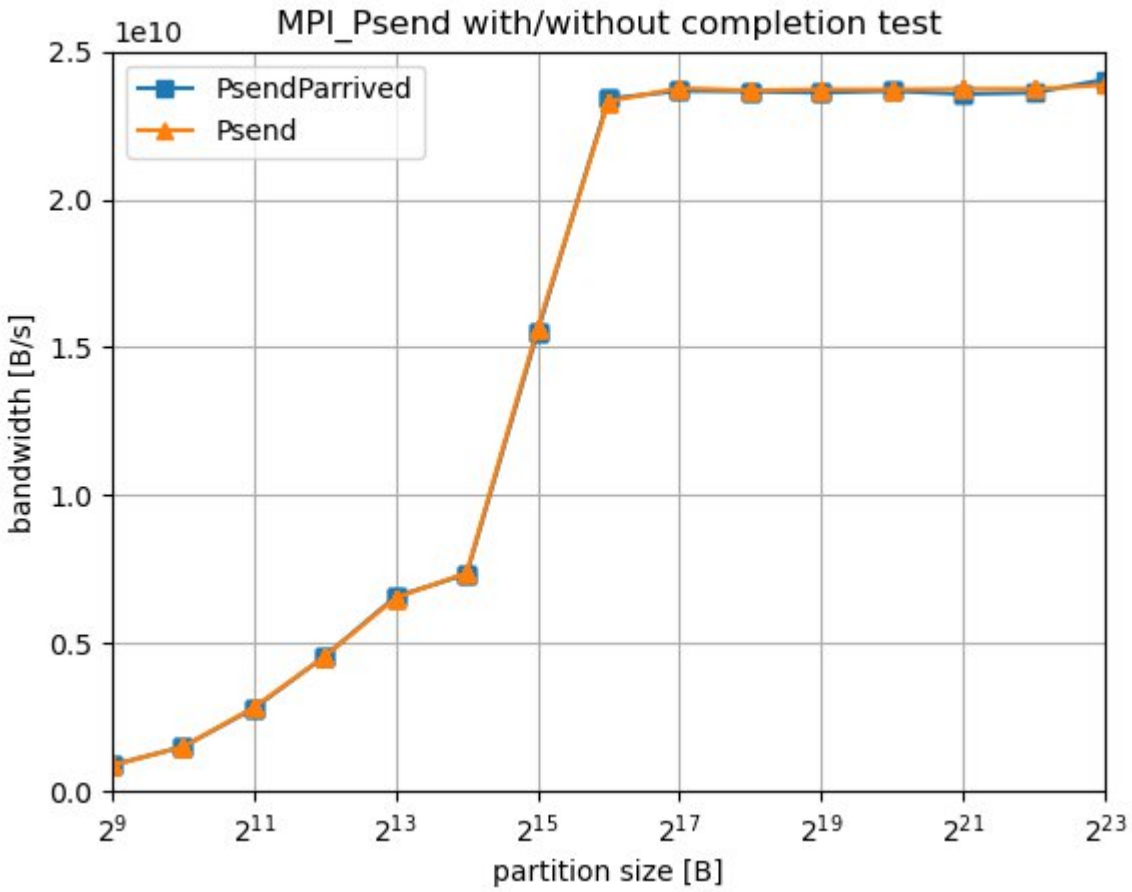


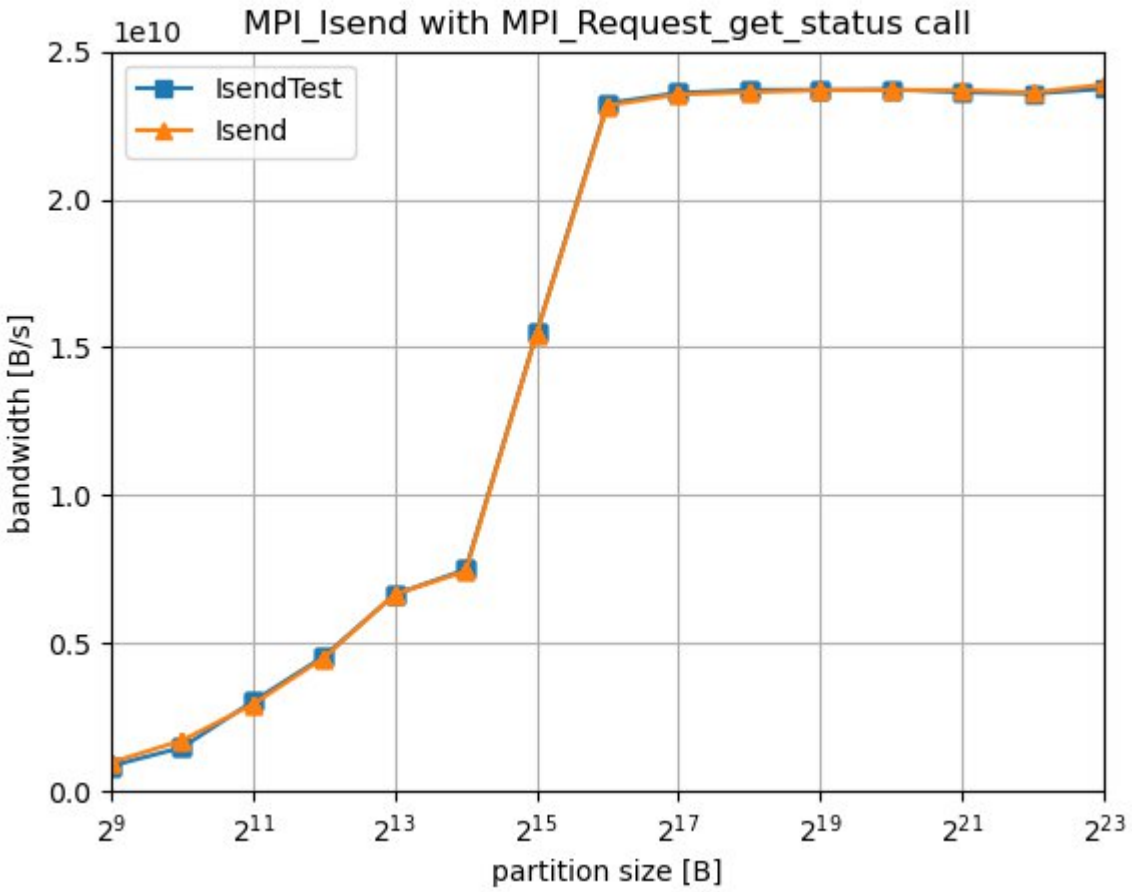
Results: Persistent Send vs Psend (multithreaded)





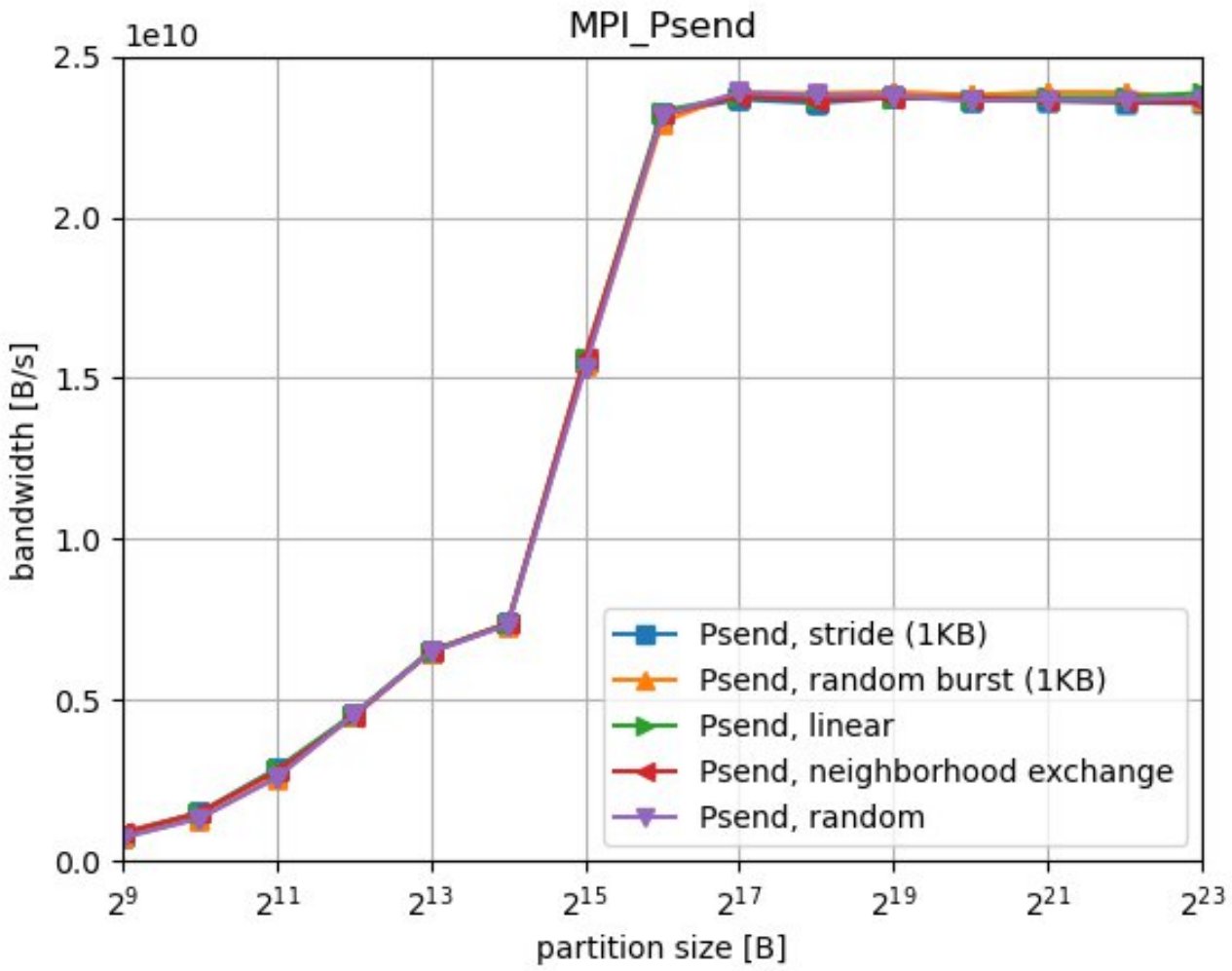
Results: Completion Testing



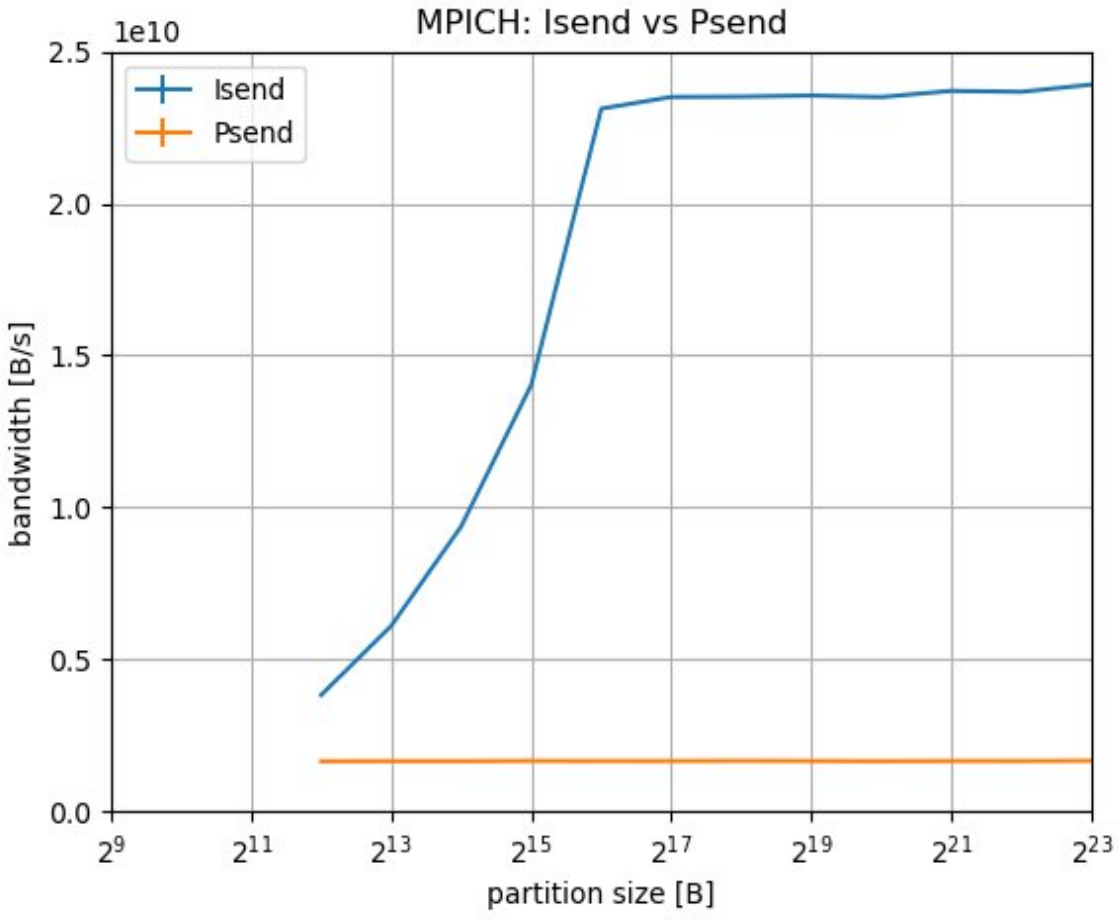


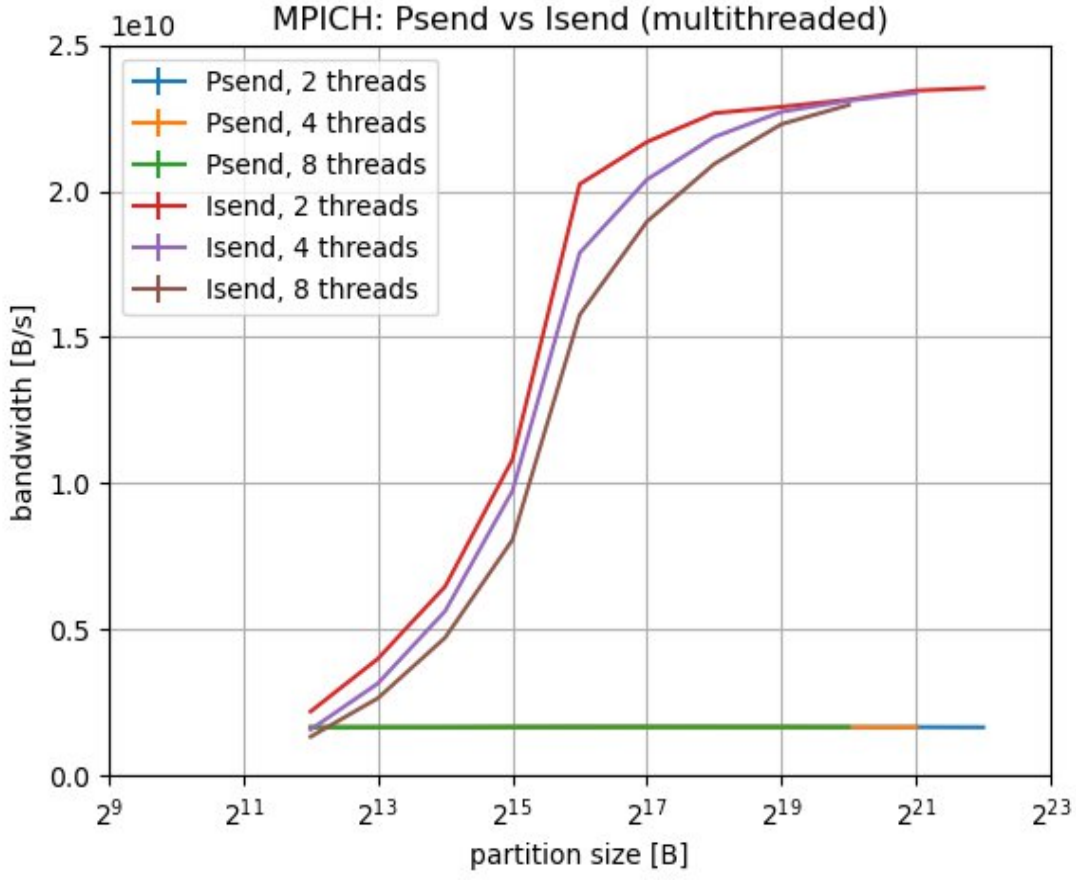
- Receive-Side completion test with MPI_Parrived
- Send-Side completion testing with MPI_Request_get_status crashes program

Results: Psend with different send patterns

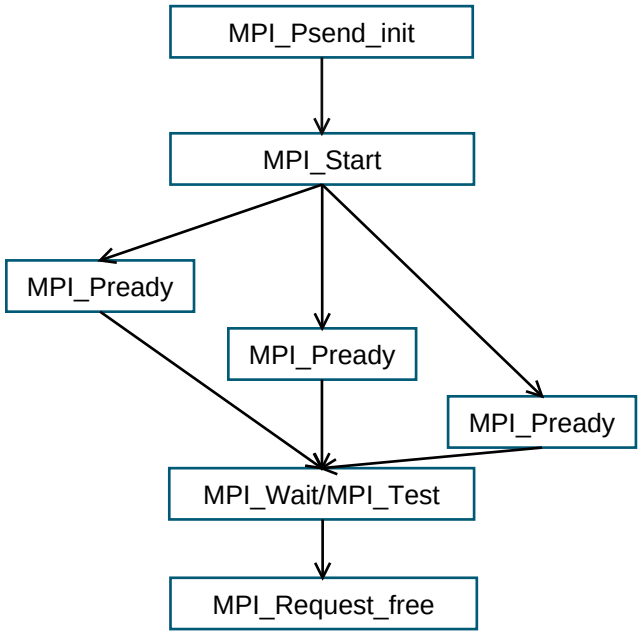


Results: Psend on MPICH-4.1.2





Current implementations: OpenMPI/5.0.2



MPI_Start():

```
for partition p:
    flag[p] = 0;
```

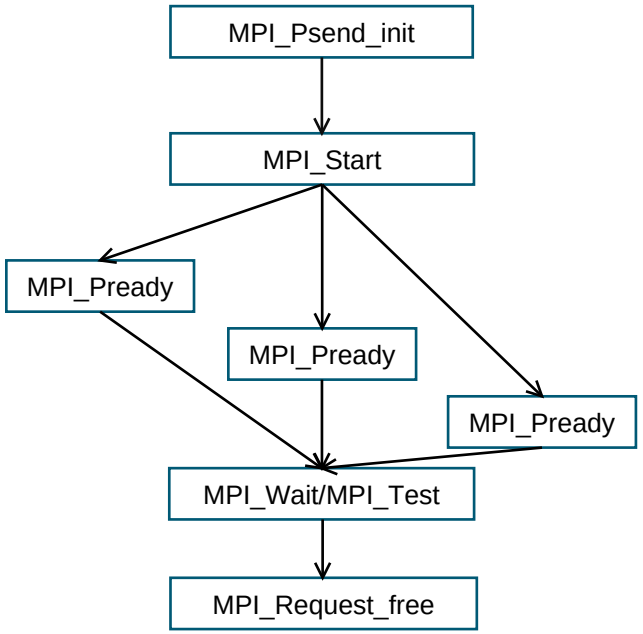
MPI_Pready(p):

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

progress():

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

Current implementations: MPICH/4.2.0

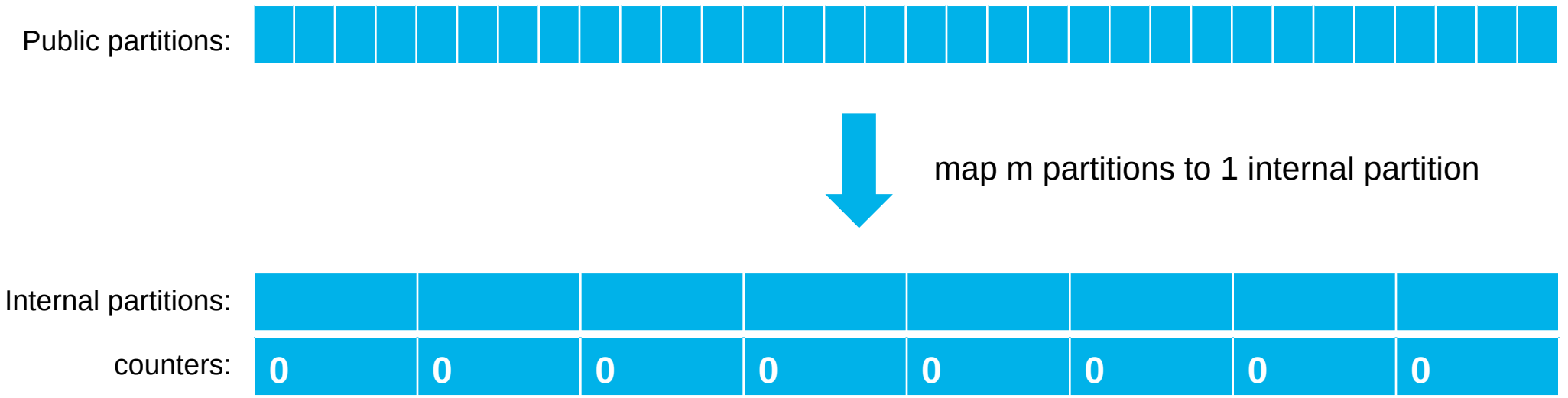


MPI_Start():
counter = partition_count

MPI_Pready():
decrement counter

if counter == 0:
send(buffer)

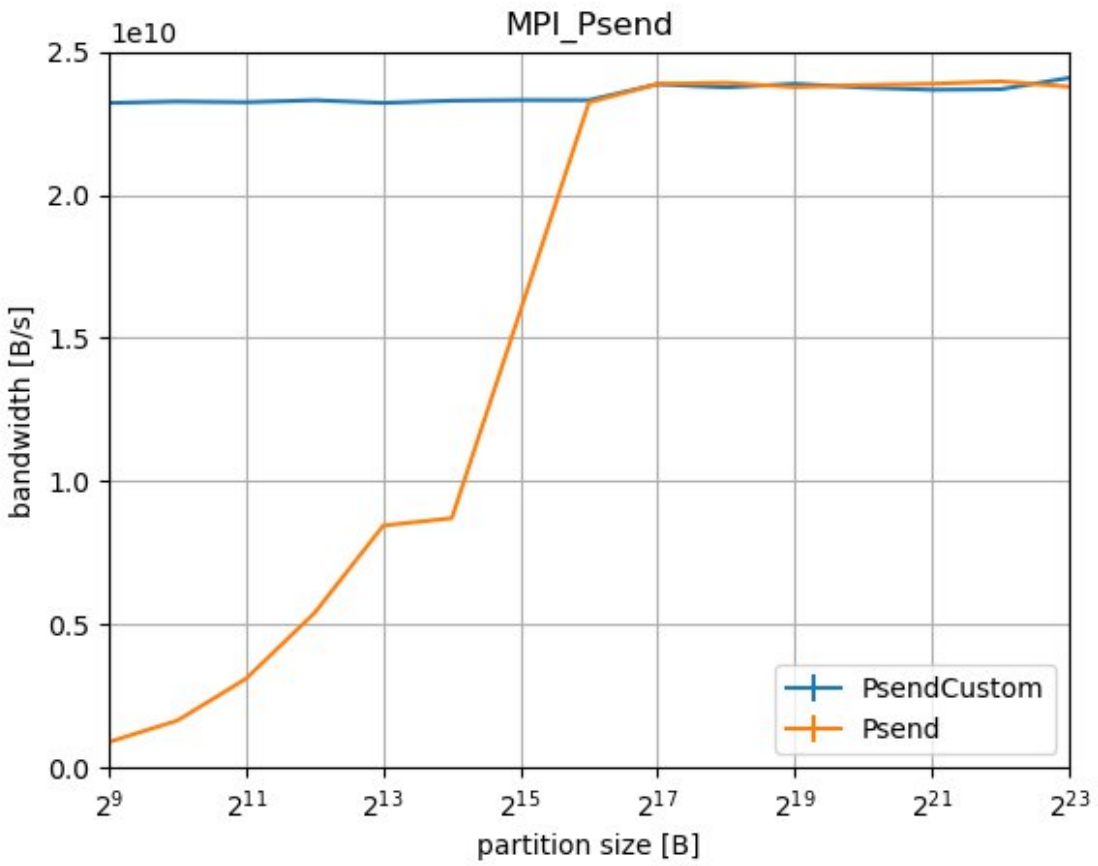
Naive method of message aggregation

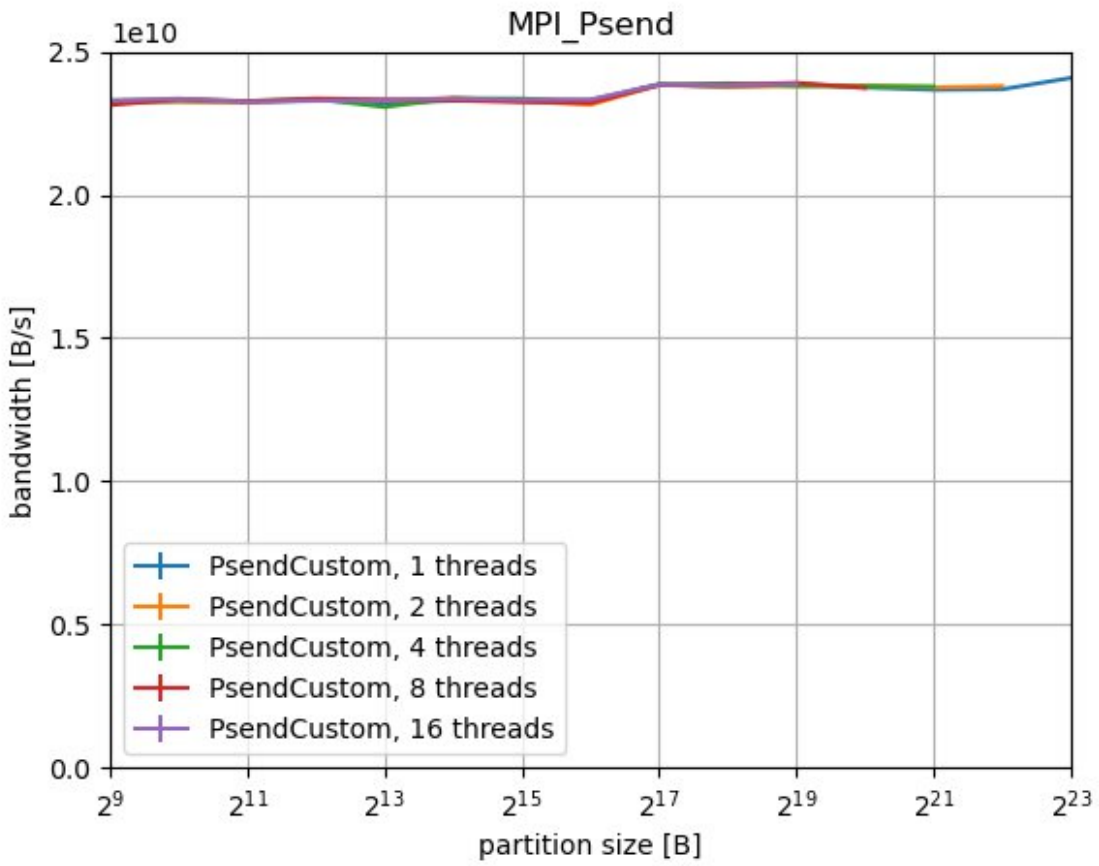


Pready(p):

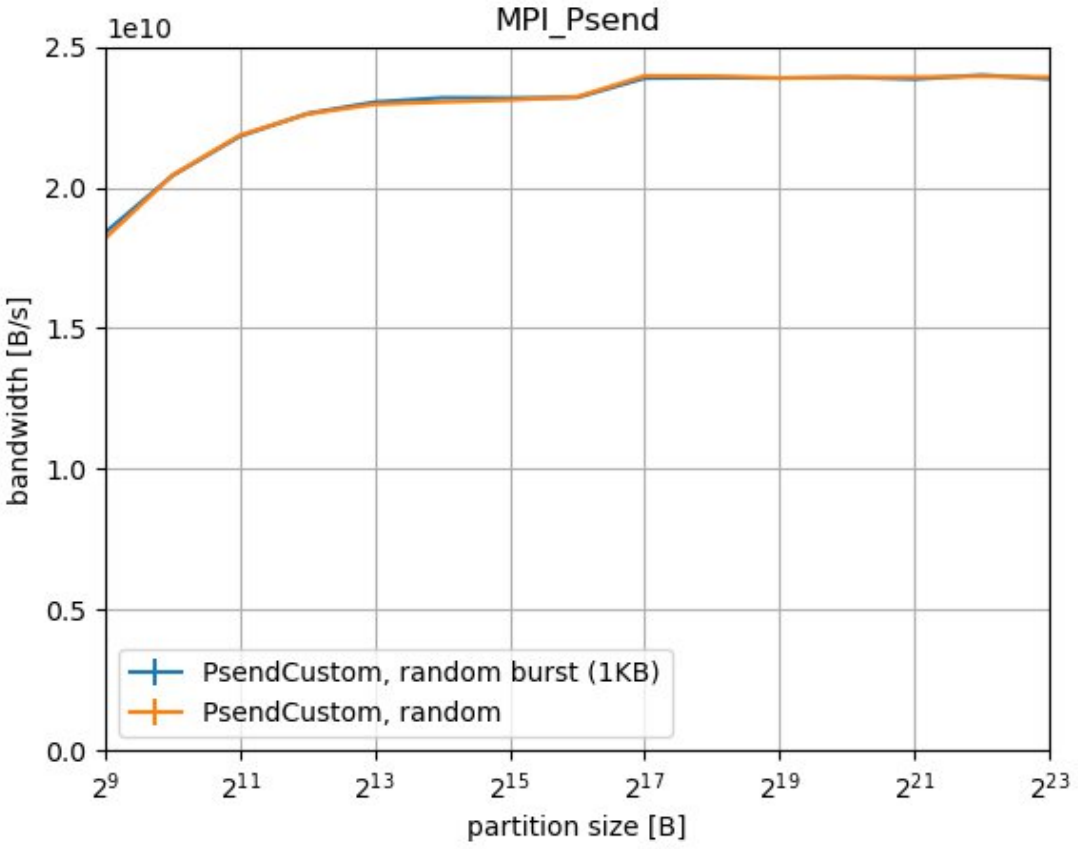
increment counter of internal partition of p
if counter >= m:
    mark internal partition as ready

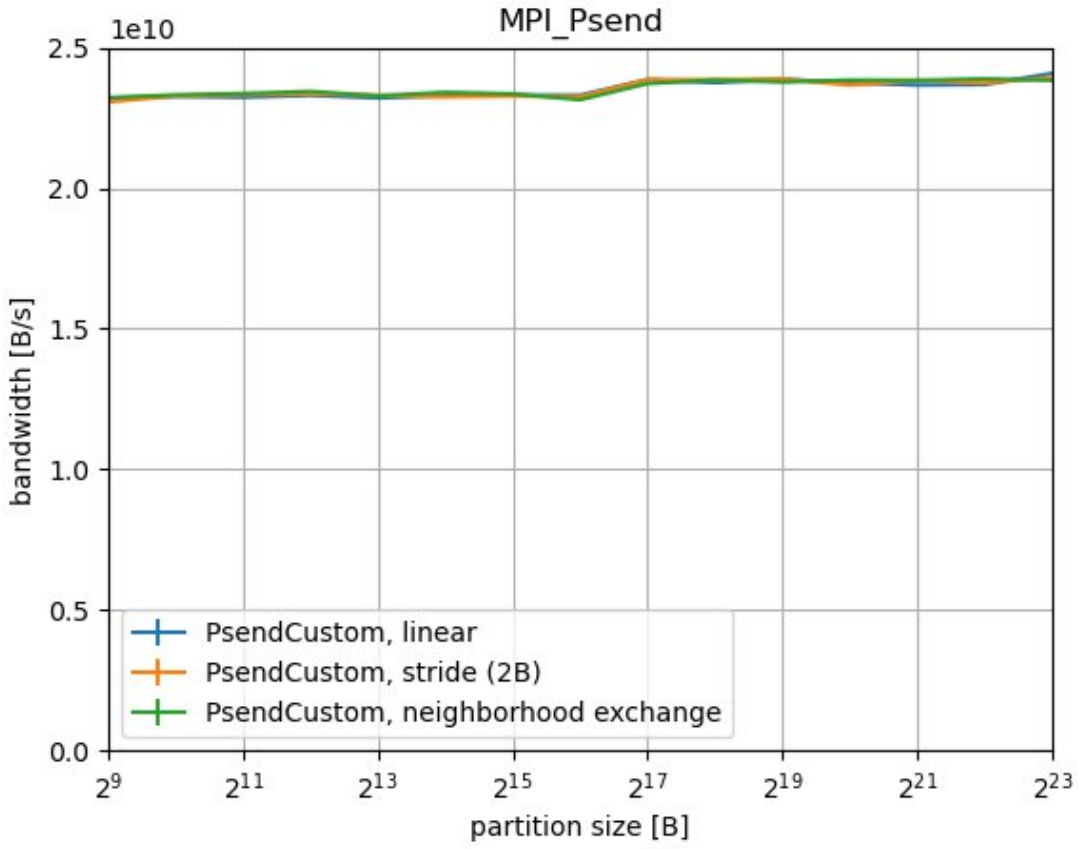
Results: Naive message aggregation





Results: Naive message aggregation





Conclusion

- 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

- 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

- 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

- 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

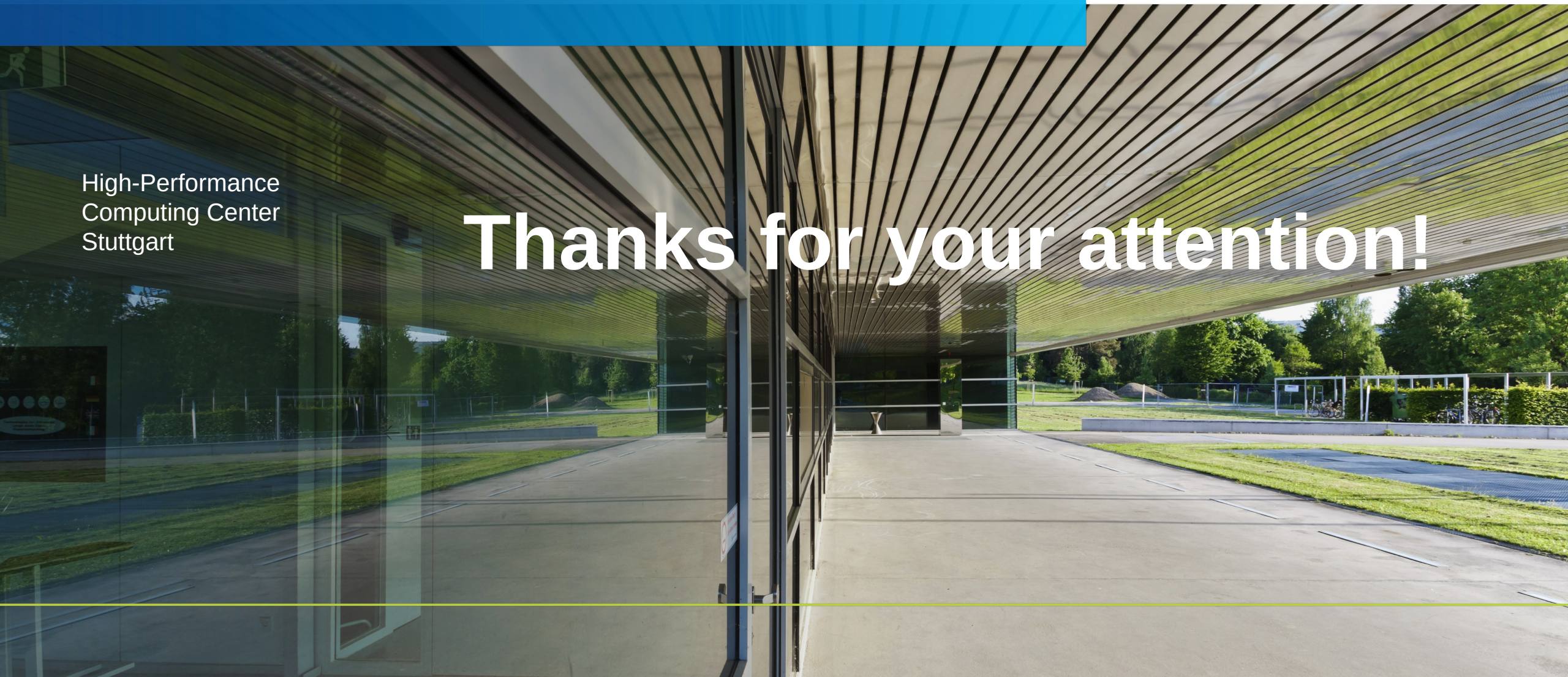
- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better when using multiple threads
  - Psend implemented using nonblocking sends
  - Currently no further optimizations such as message aggregation
  - Send pattern does not influence performance
- MPICH:
  - 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

- 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

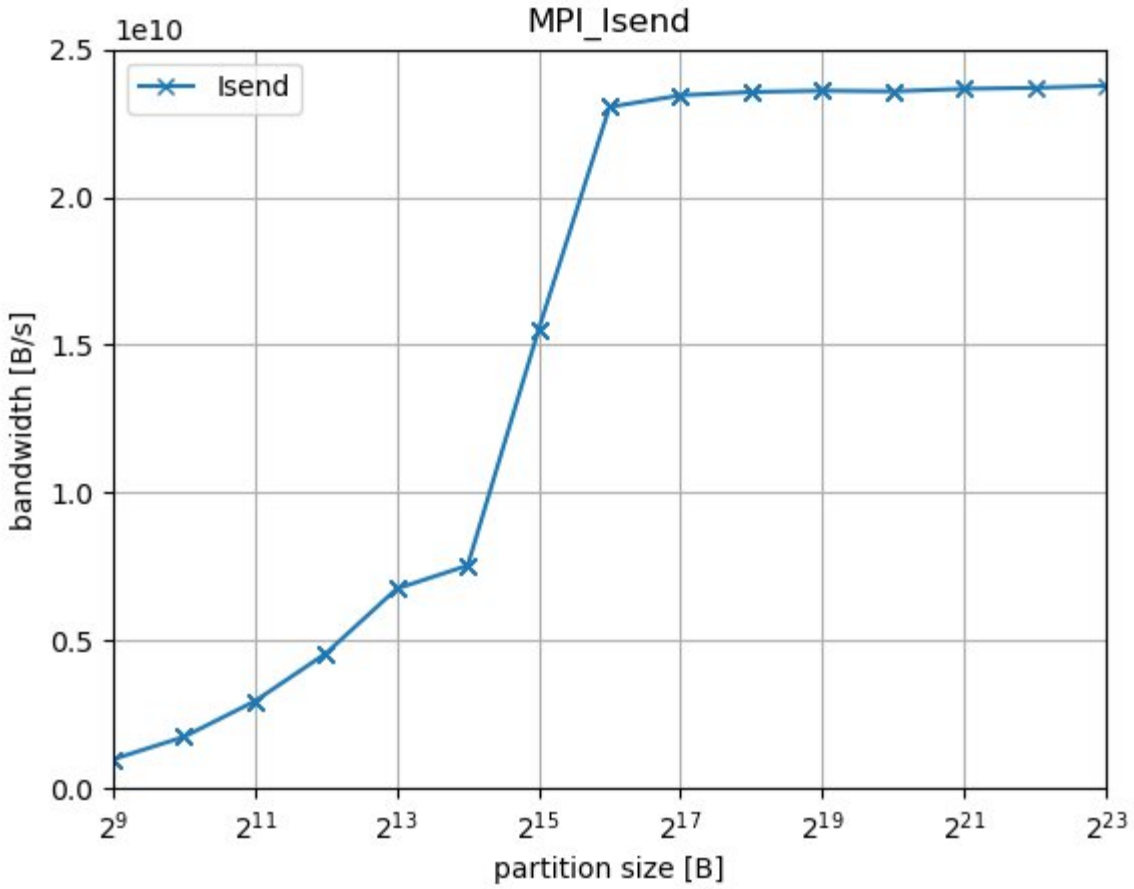
Thanks for your attention!

Appendix – OpenMPI progress

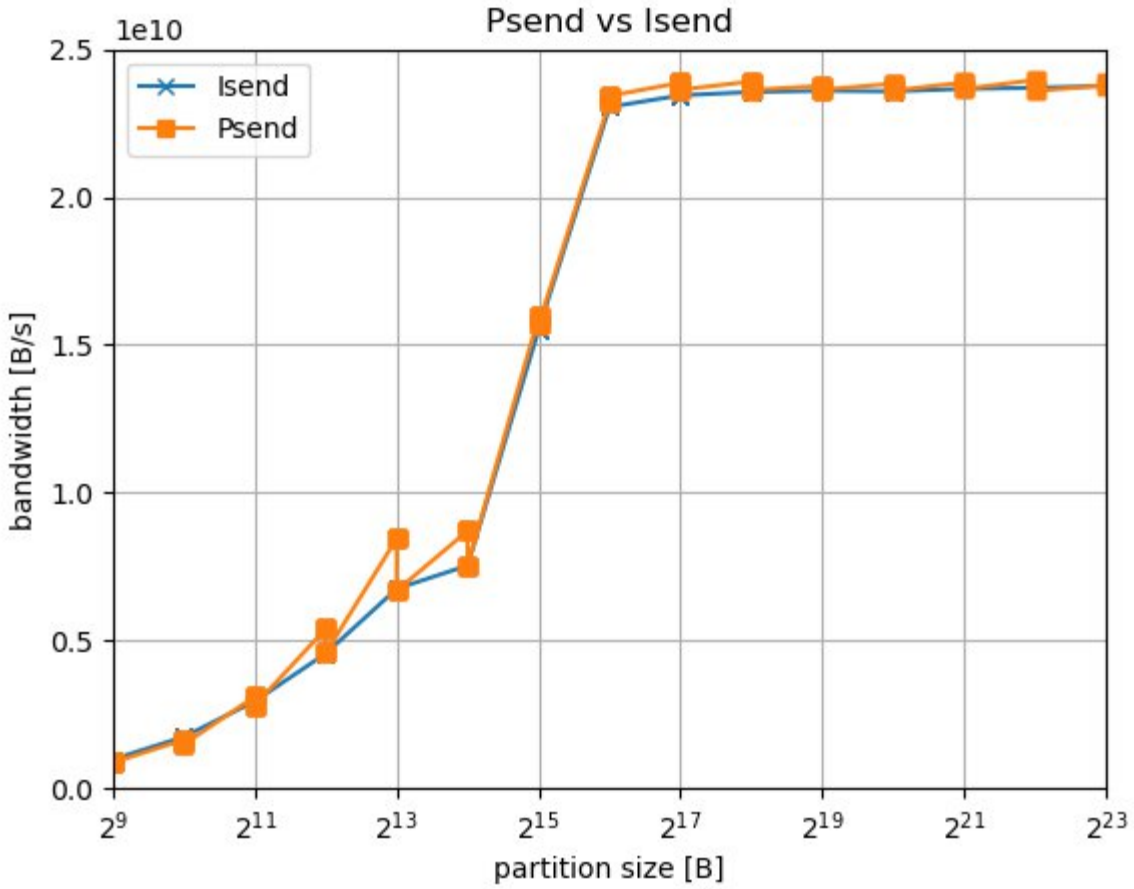
ompi > mca > part > persist > C part_persist.h

```
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->persist_reqs[i]->req_start(1, (&(req->persist_reqs[i])));
293                   req->flags[i] = 0;
294               }
295
296               if(0 == req->flags[i])
297               {
298                   ompi_request_test(&(req->persist_reqs[i]), &(req->flags[i]), MPI_STATUS_IGNORE);
299                   if(0 != req->flags[i]) req->done_count++;
300               }
301           }
```

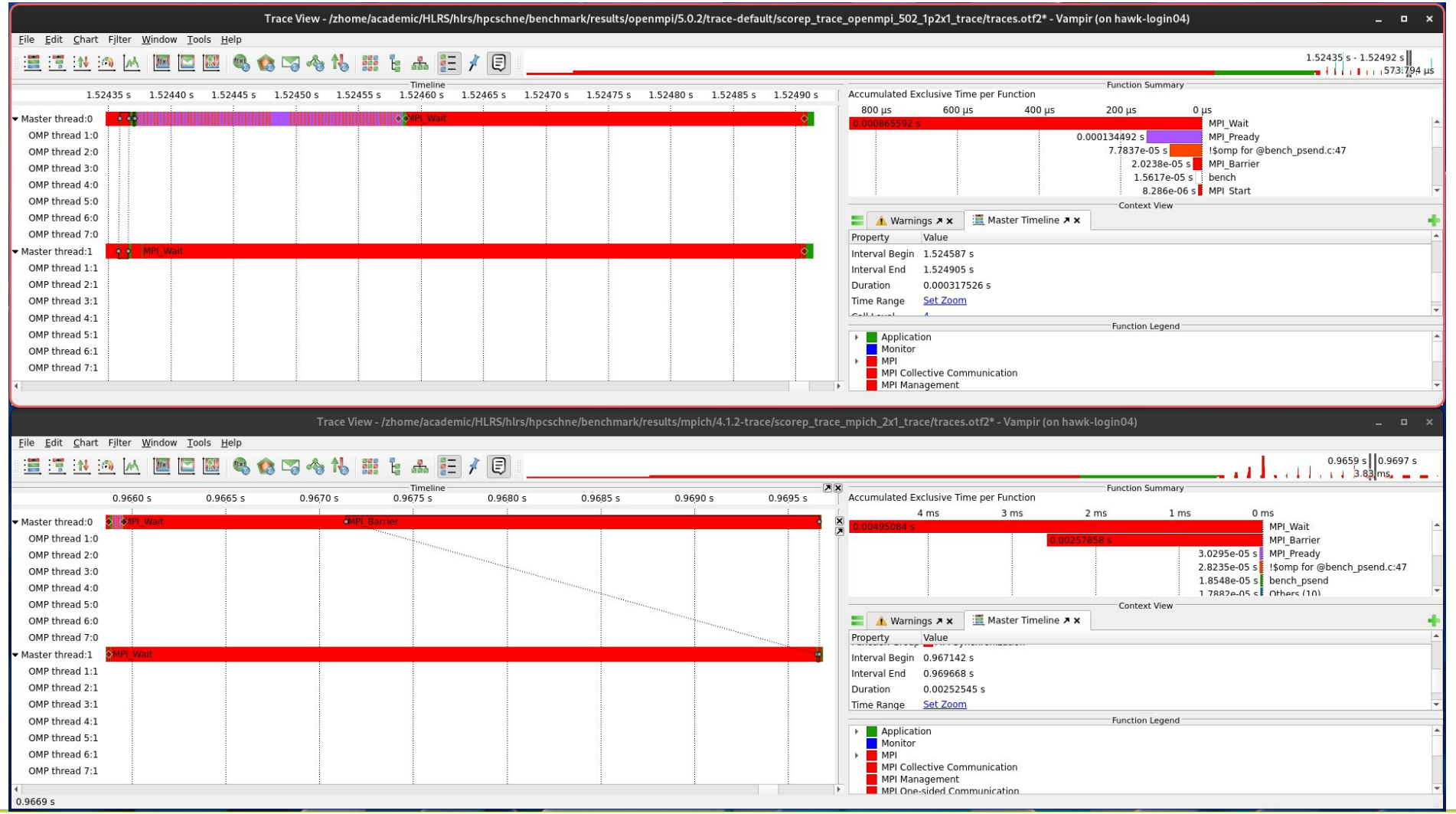
Isend


MPI_Isend
1e10
Isend
2.5
2.0
1.5
1.0
0.5
0.0
bandwidth [B/s]
2^9
2^{11}
2^{13}
2^{15}
2^{17}
2^{19}
2^{21}
2^{23}
partition size [B]

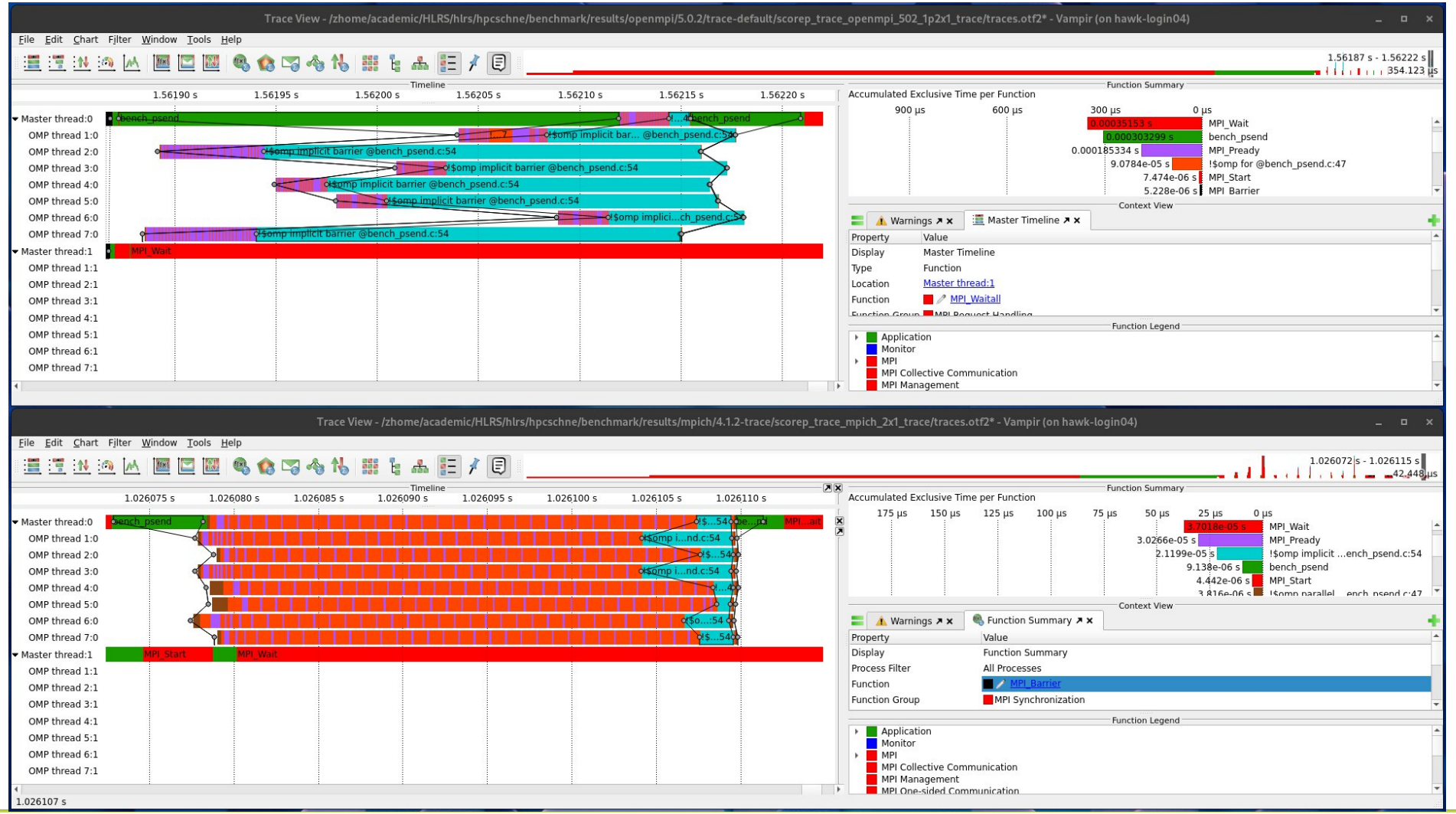
Isend vs Psend



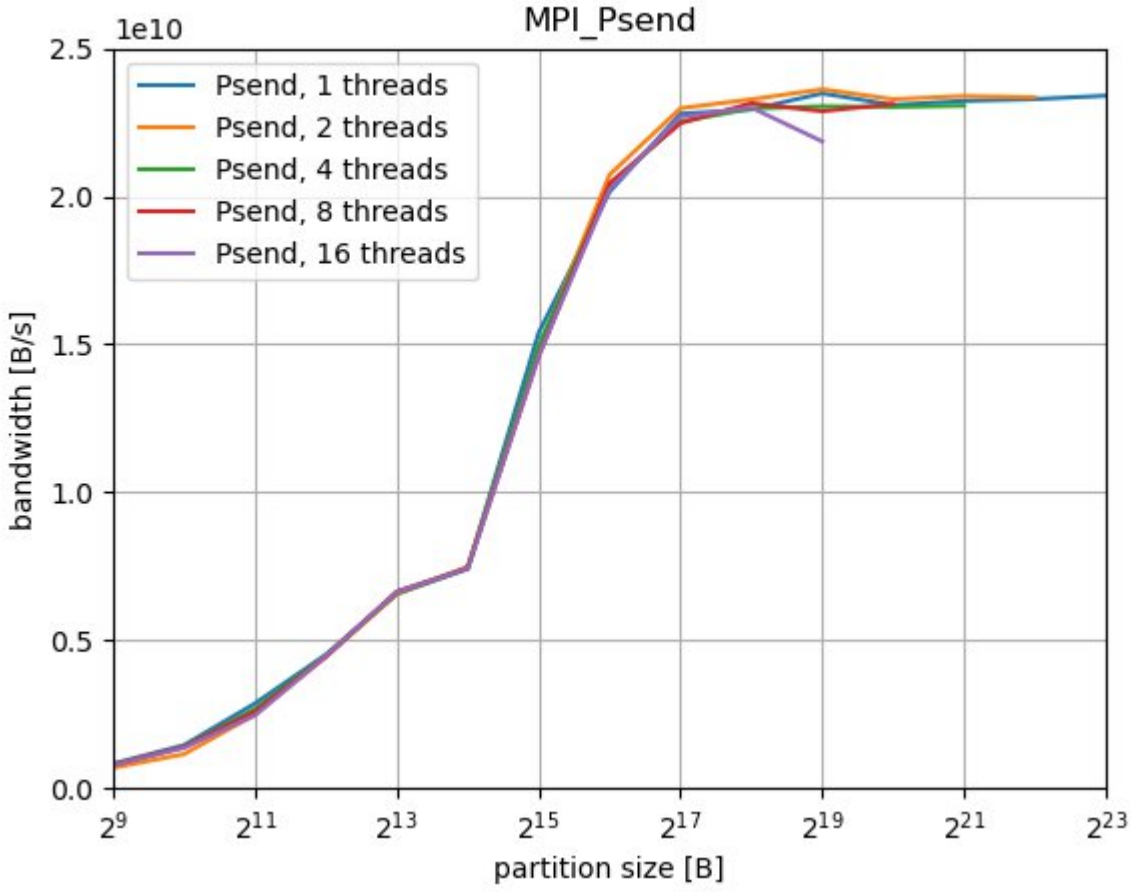
Traces: MPI_Psend

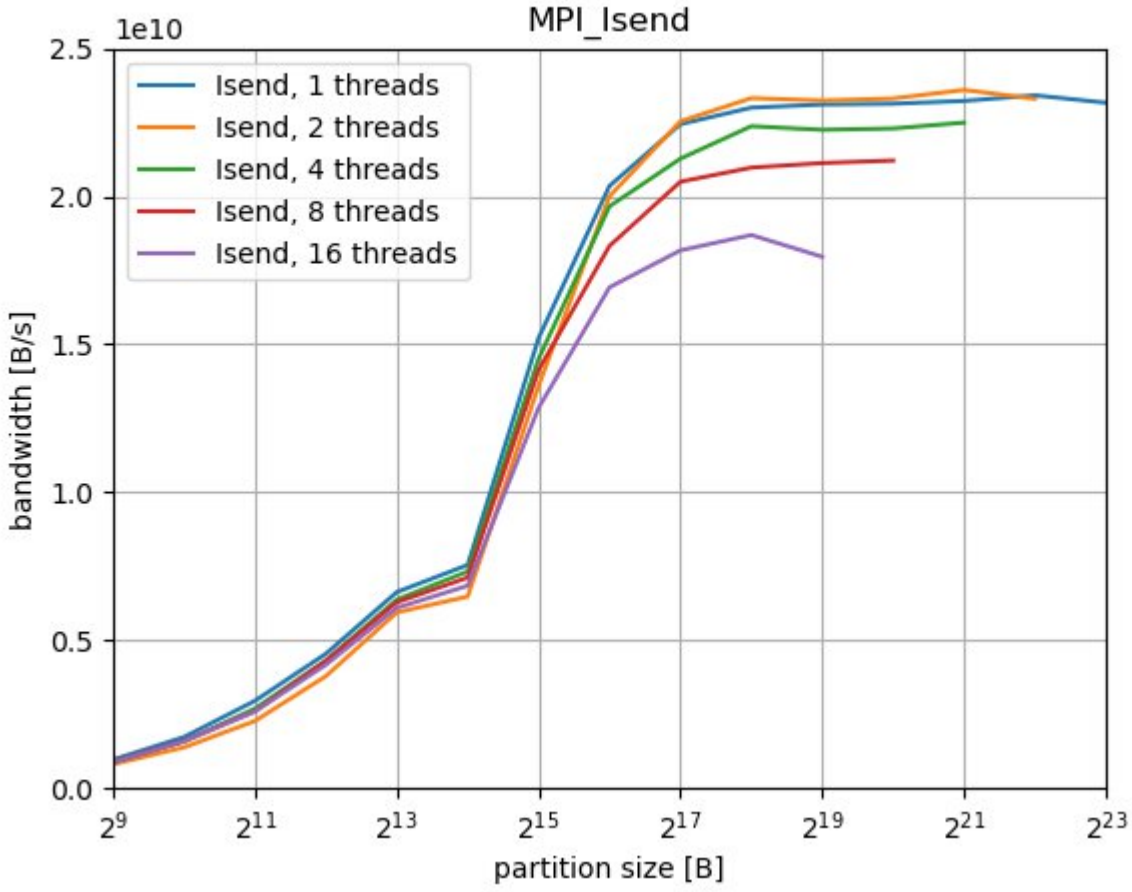


Traces: MPI_Psend – MPI_Pready-Calls



Appendix: Isend vs Psend (multithreaded)





Results: MPI_Win