High-Performance
Computing Center
Stuttgart

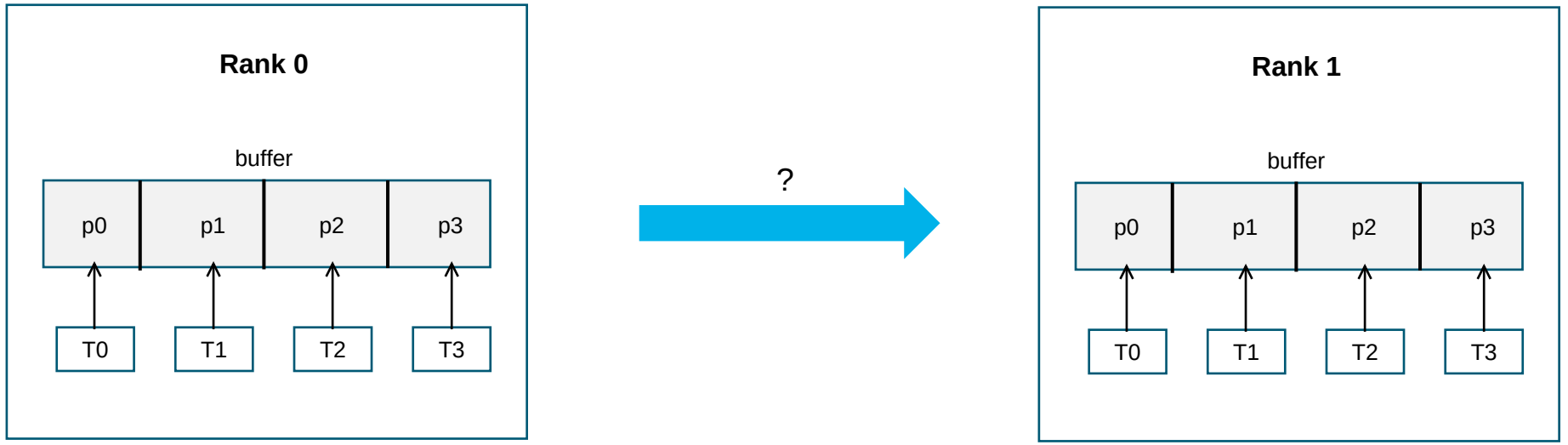
Transfer optimizations on MPI partitioned communication

Axel Schneewind

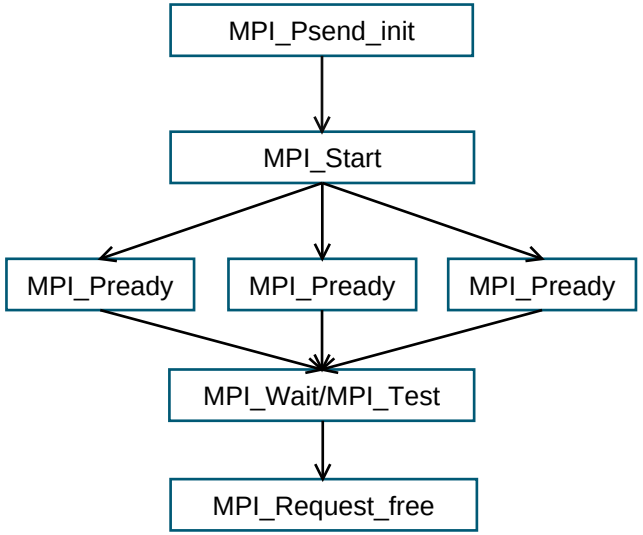
Outline

- MPI partitioned communication
  - Introduction
  - MPI-4.1 standard
- Optimization opportunities
- Current implementations in OpenMPI and MPICH
- Benchmarks
  - Psend vs Isend (single-/multithreaded)
  - Psend, Isend with completion tests
  - Psend vs RDMA

Motivation: Hybrid Programming



MPI: Partitioned Communication Operations



MPI-4.1: Nonblocking Communication Operations

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.

In particular, if the matching send is nonblocking, then the receive should complete even if no call is executed by the sender to complete the send.

MPI-4.1: Partitioned Communication

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

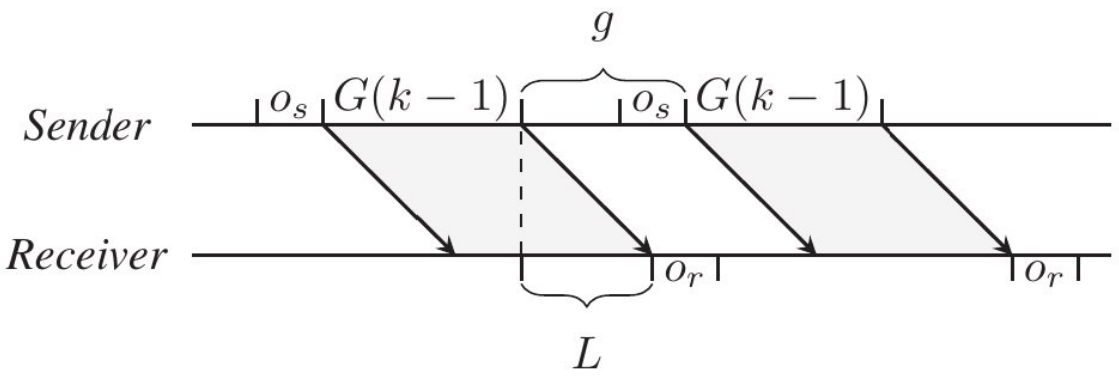
MPI is free to choose **how many transfers** to do within a partitioned communication send independent of how many partitions are reported as ready to MPI through MPI_PREADY calls.
Aggregation of partitions is permitted but not required.
Ordering of partitions is permitted but not required.

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. However, this modality of communication gives MPI implementations far more flexibility in data movement than nonpartitioned communications. (*End of rationale.*)

Possible optimizations – Reduce locking overhead

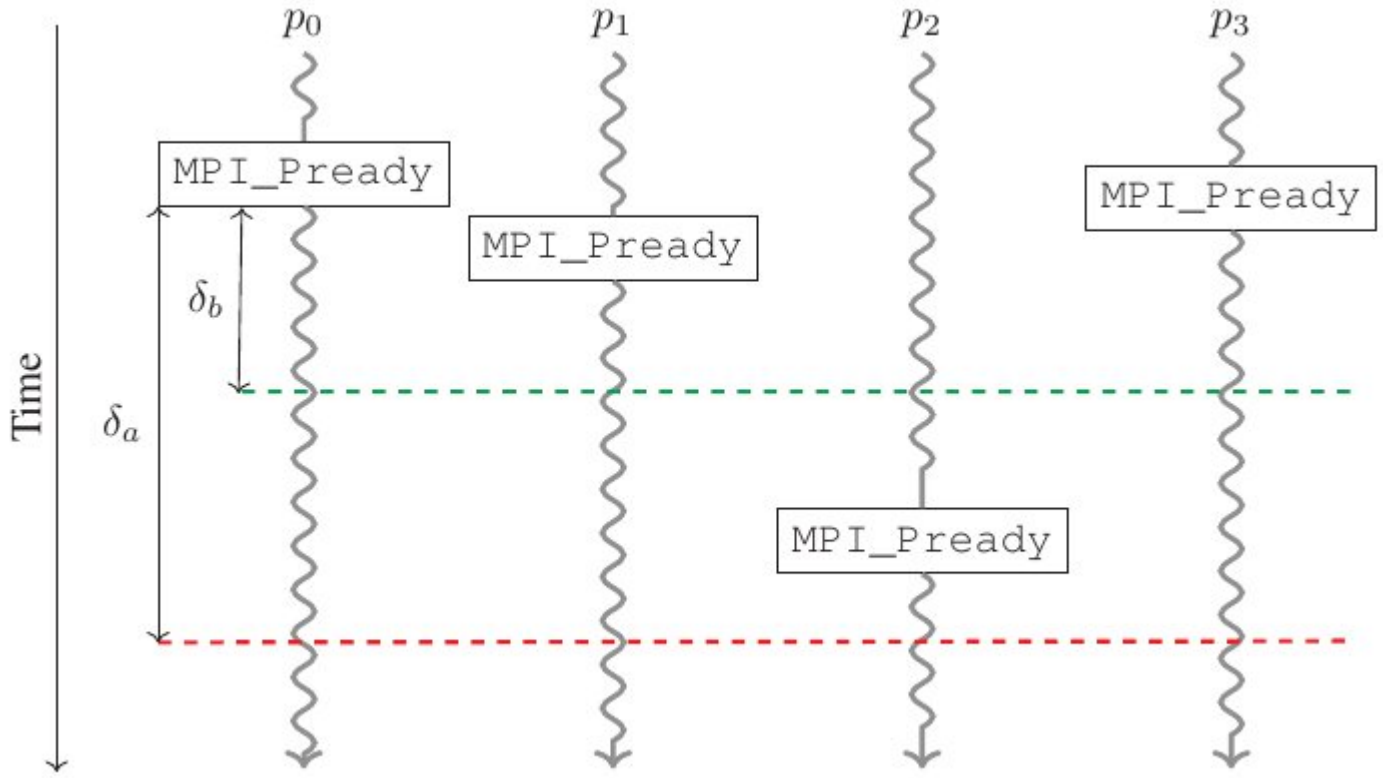
Possible optimizations – Early-bird effect

Possible optimizations – Message aggregation



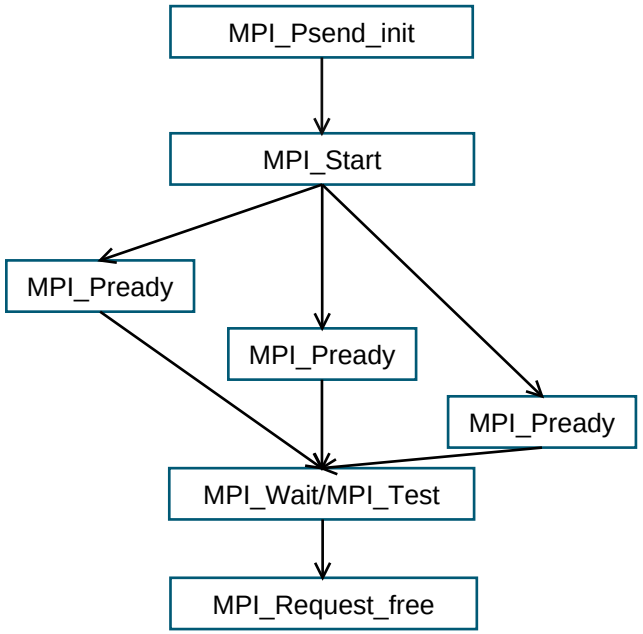
Y. H. Temuçin, S. Levy, W. Schonbein, R. E. Grant and A. Afsahi, "A Dynamic Network-Native MPI Partitioned Aggregation Over InfiniBand Verbs," https://doi.org/10.1109/CLUSTER52292.2023.00029

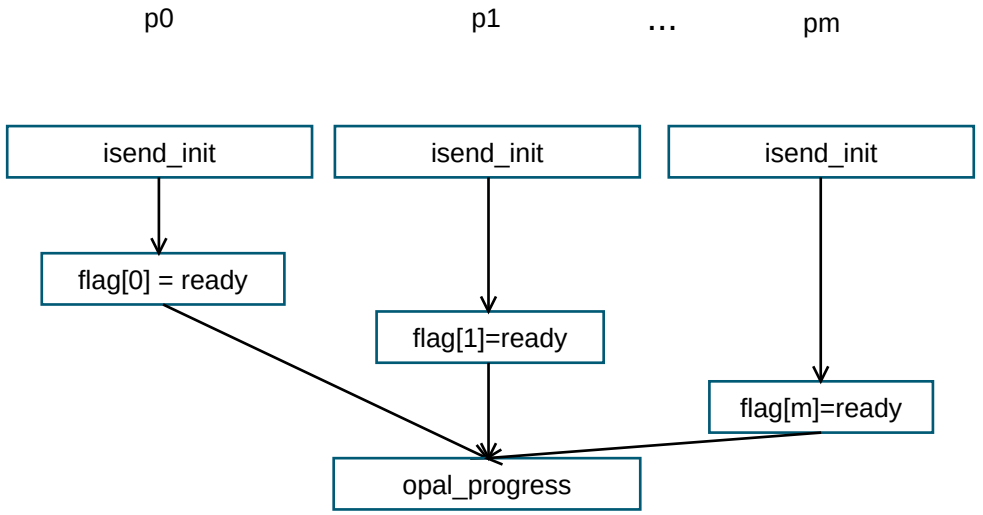
Possible optimizations – Message aggregation

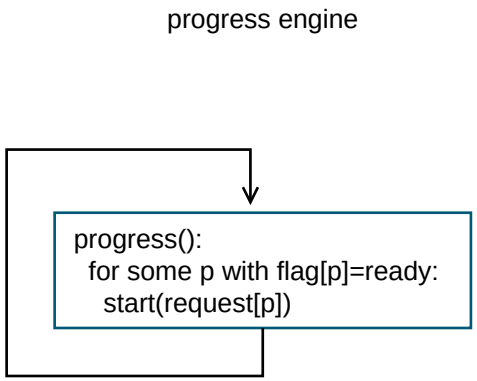


Y. H. Temuçin, S. Levy, W. Schonbein, R. E. Grant and A. Afsahi, "A Dynamic Network-Native MPI Partitioned Aggregation Over InfiniBand Verbs," https://doi.org/10.1109/CLUSTER52292.2023.00029

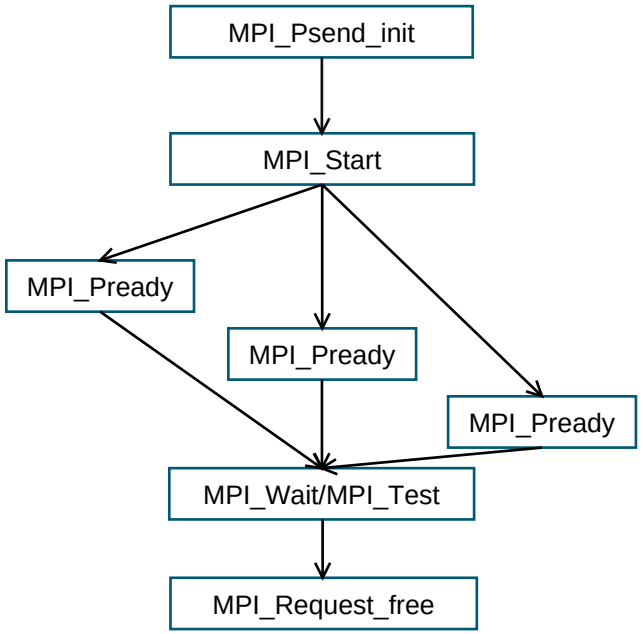
Current implementations: OpenMPI/5.0.2

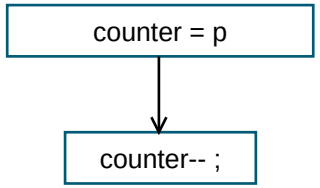


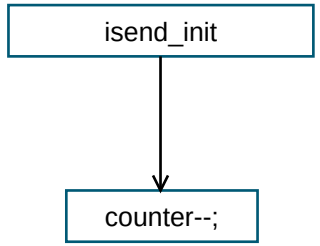


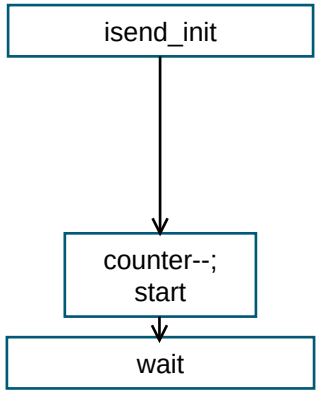


Current implementations: MPICH/4.2.0









Goals

- Measure effective bandwidth depending on message size
- Do different MPI-implementations perform optimizations (aggregation, reordering) on partitioned transfers?
- Is it possible to trigger early progress using completion tests?
- Do OpenMPIs Point-to-point Messaging Layer (PML) – components make use of hardware offloading?

Benchmarking scheme

```
1  // initialization
2
3  for i in 0 to num_iterations:
4   MPI_Barrier();  // synchronization
5
6   s_i = MPI_Wtime();
7
8   MPI_Start(request);
9
10  for p in 0 to partition_count in some order:
11   perform some MPI operation to send/receive partition p
12
13  MPI_Wait(request, send_status);
14
15  e_i = MPI_Wtime();
16
17 // cleanup
18
19 time = mean over (e_i - s_i)
```

Benchmarks

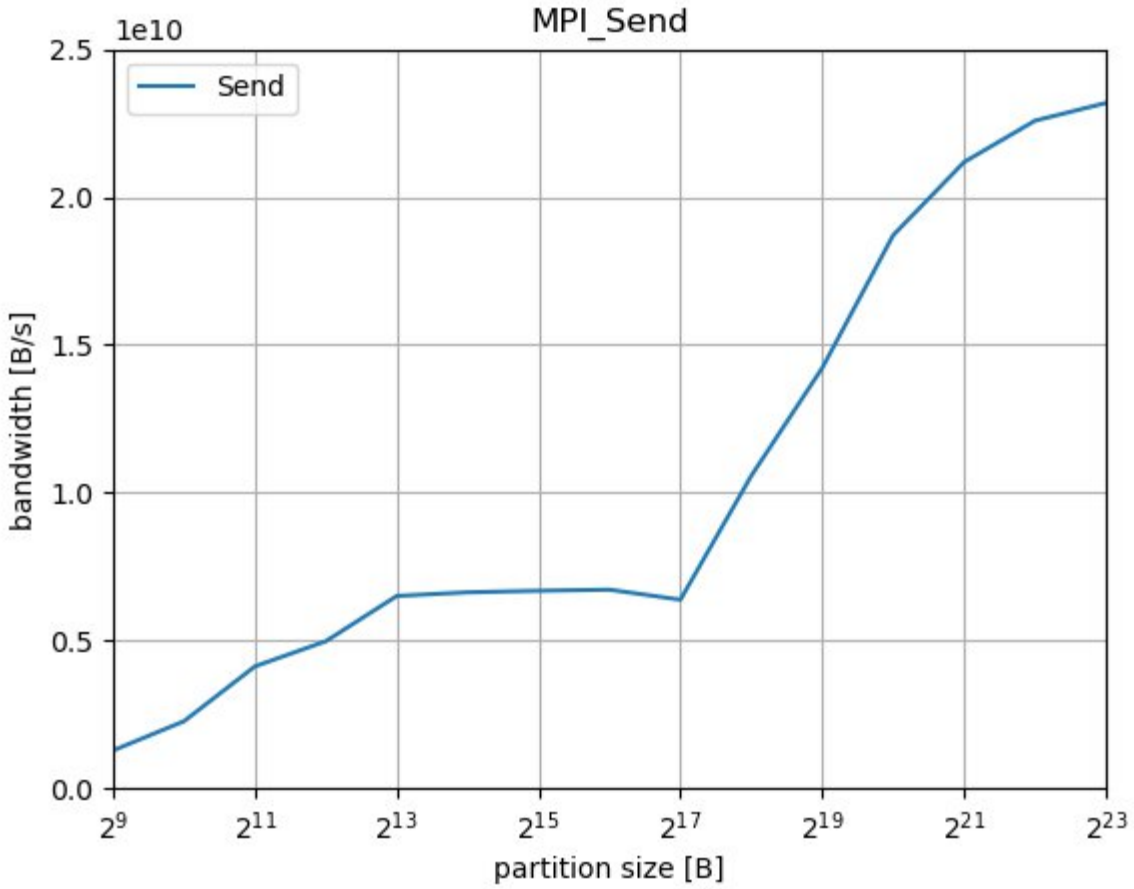
Mechanism	Initialization	Partition ready	Completion	Freeing
Send	-	MPI_Send (p)	-	-
SendPersistent	MPI_Send_init (p)	MPI_Start (p)	MPI_Waitall	MPI_Request_free (p)
Isend	-	MPI_Isend (p)	MPI_Waitall	MPI_Request_free (p)
Psend	MPI_Psend_init	MPI_Pready (p)	MPI_Wait	-
Win	MPI_Win_create	MPI_lock MPI_Put(p) MPI_unlock	-	MPI_Win_free

operations marked with (p) are performed for each partition

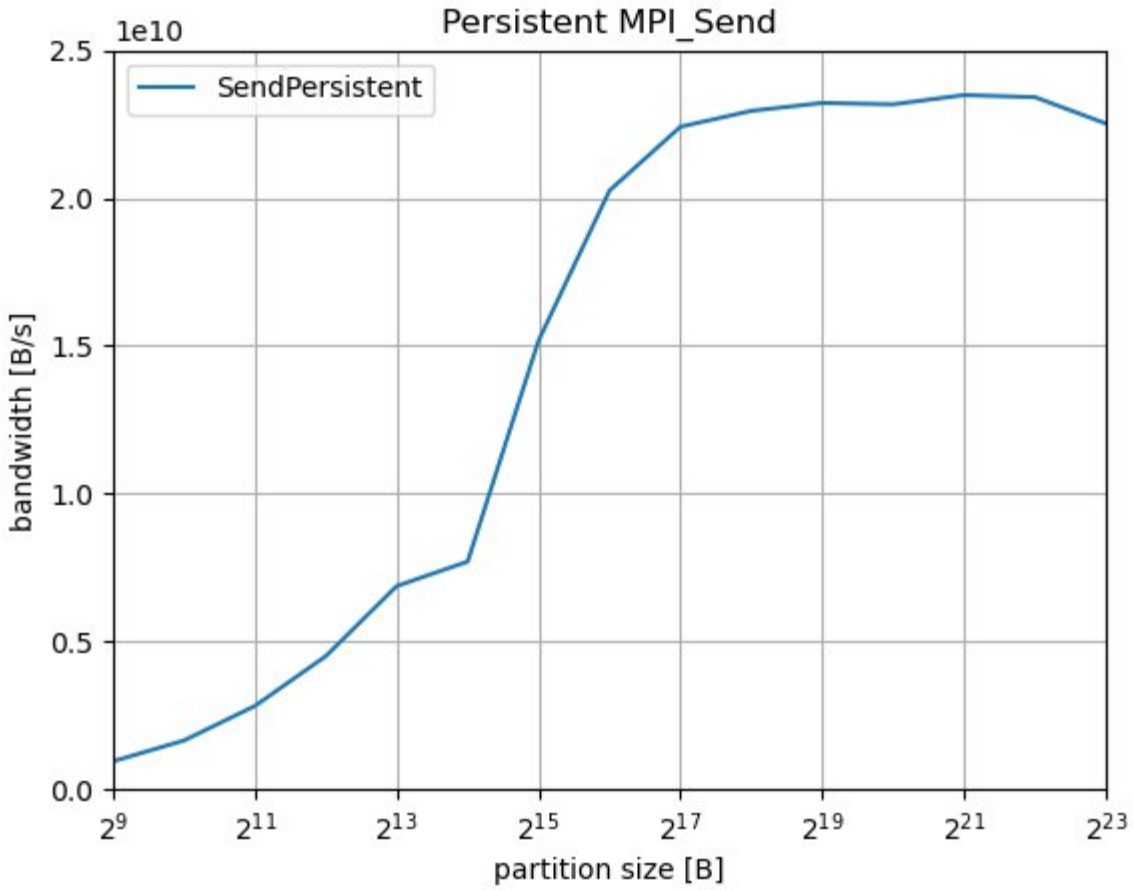
Experimental setup

- 2 nodes on HAWK with one mpi process each
  - Bandwidth:
- Fixed buffer size of 8MiB, partition sizes between 512B and 8MiB
- Partitions marked ready in left-to-right or randomized order
- Multithreading with OpenMP

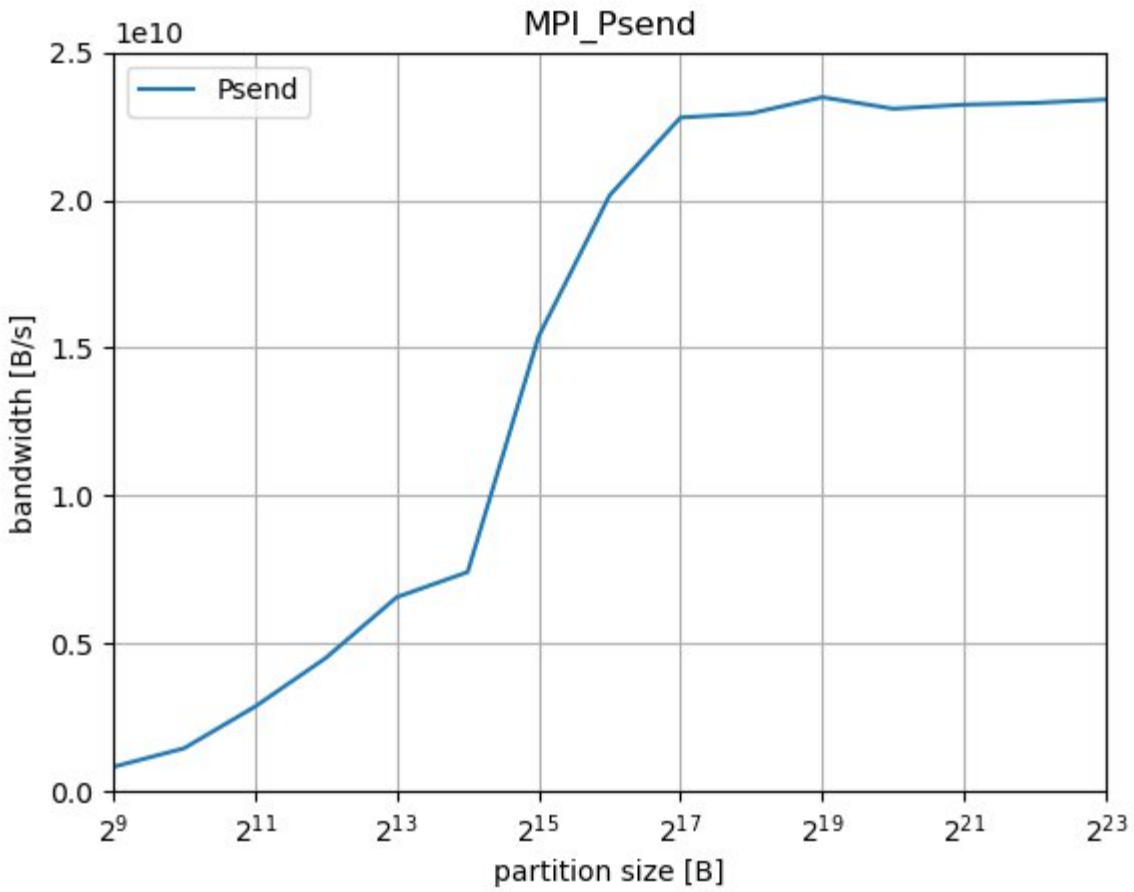
Baseline: Send->Recv



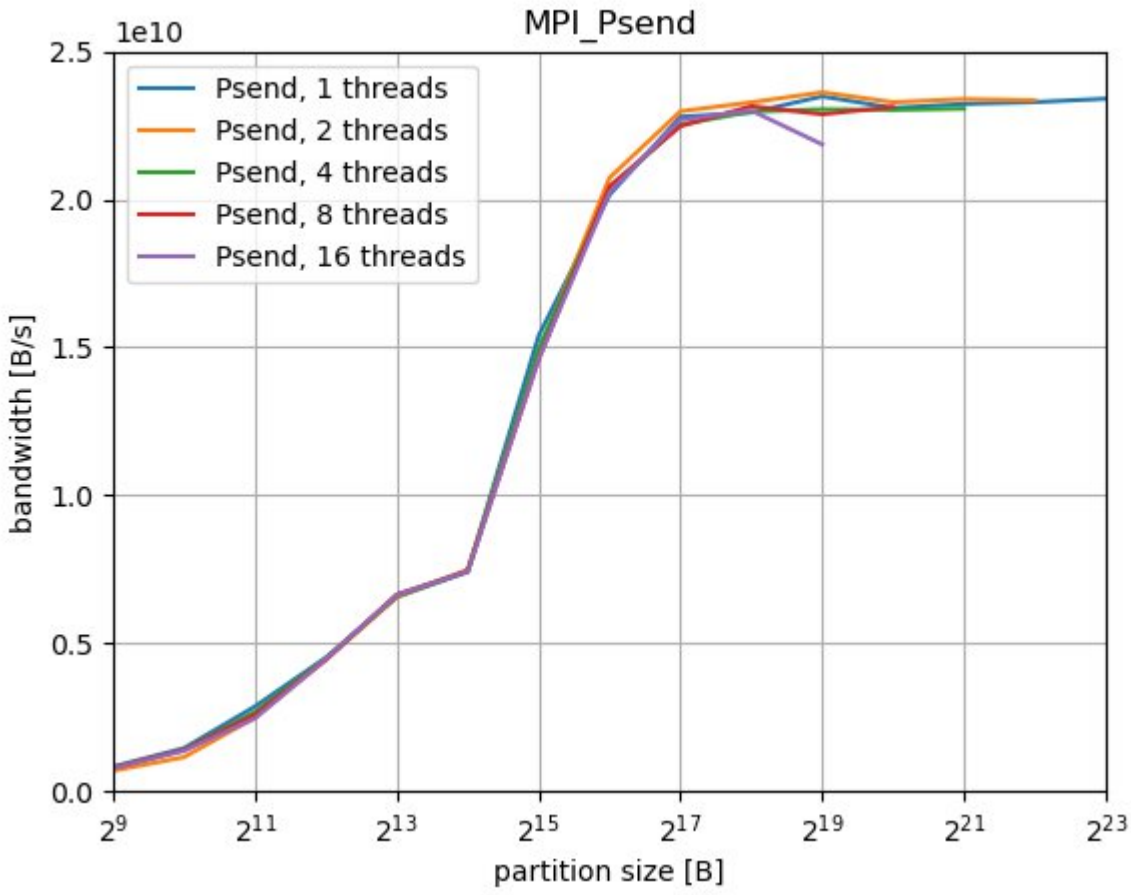
Baseline: Persistent Send

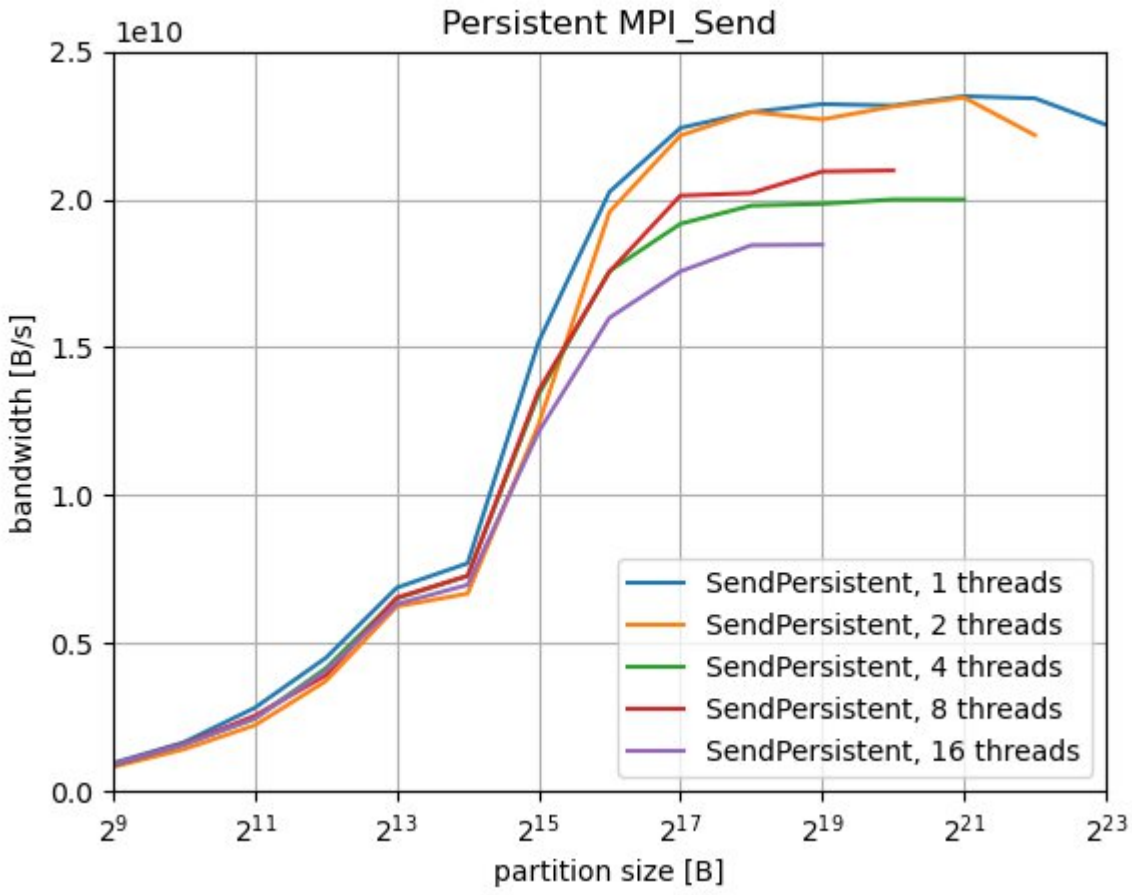


Results: Psend

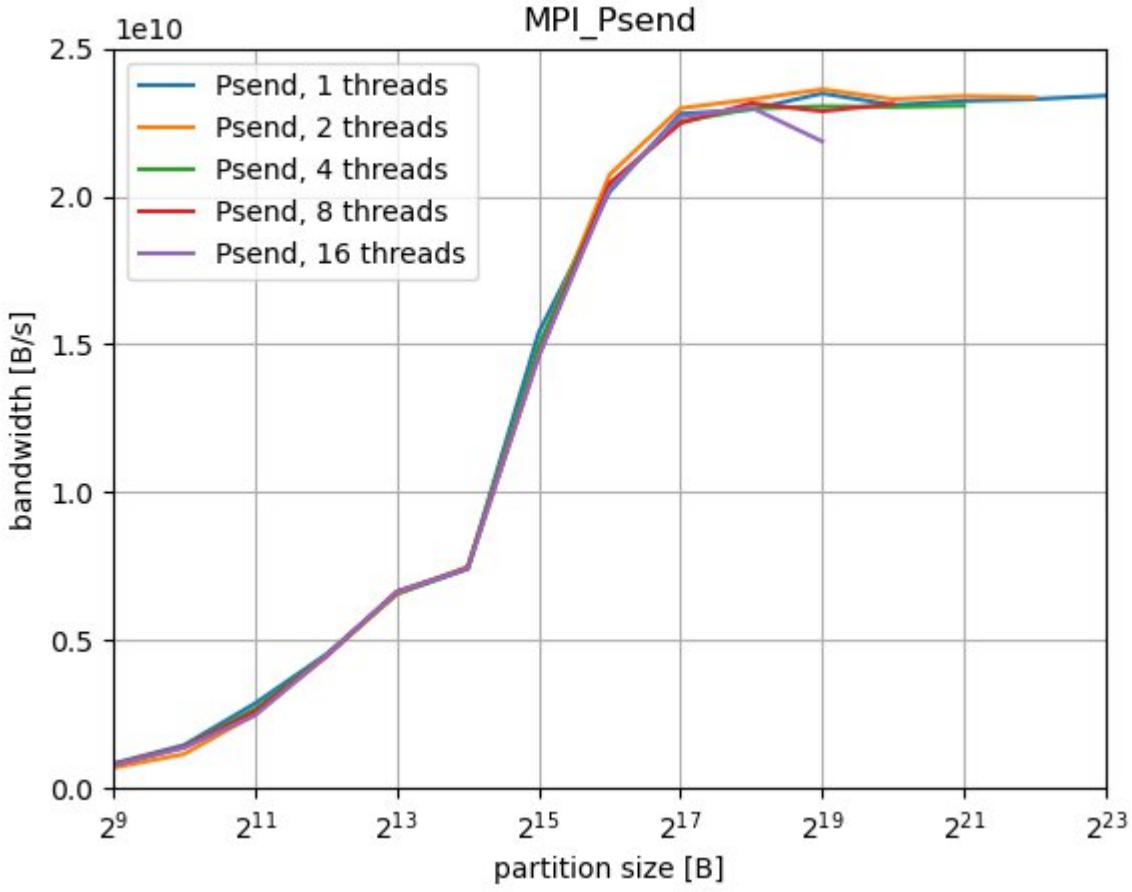


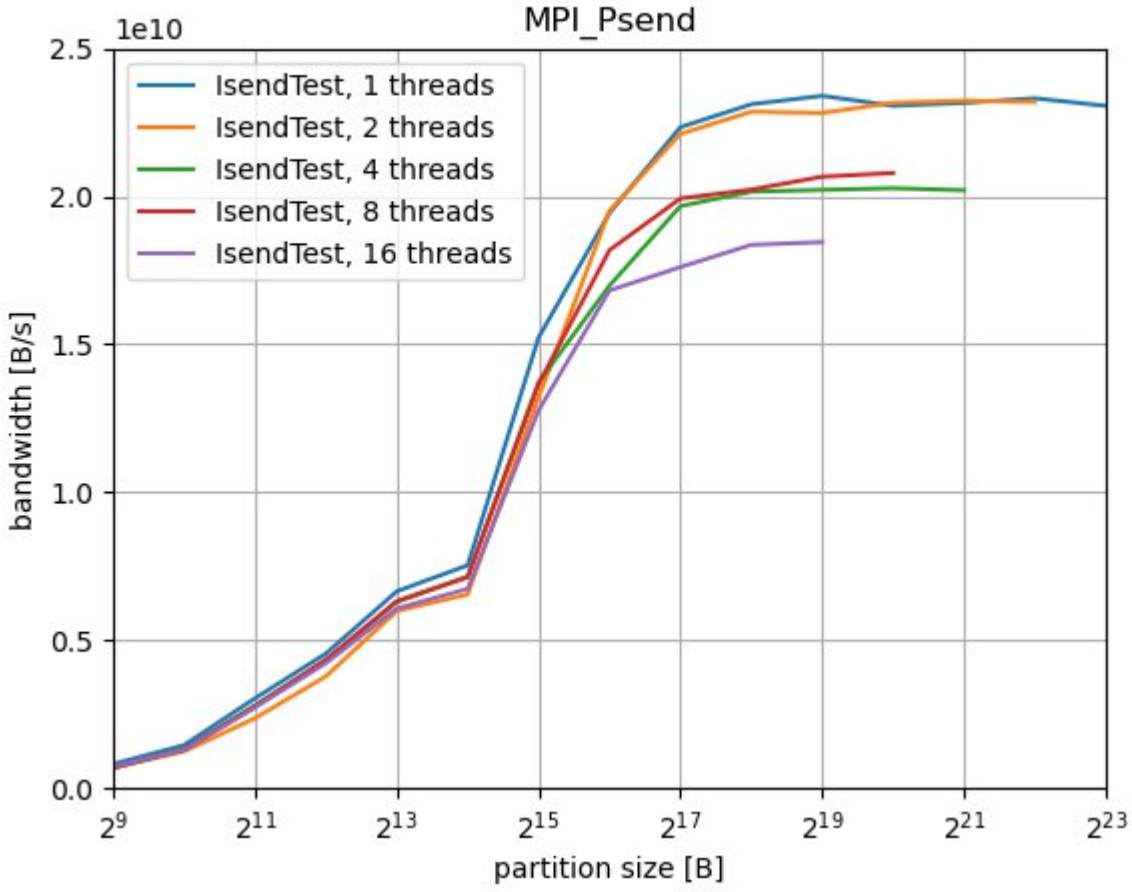
Results: Persistent Send vs Psend (multithreaded)



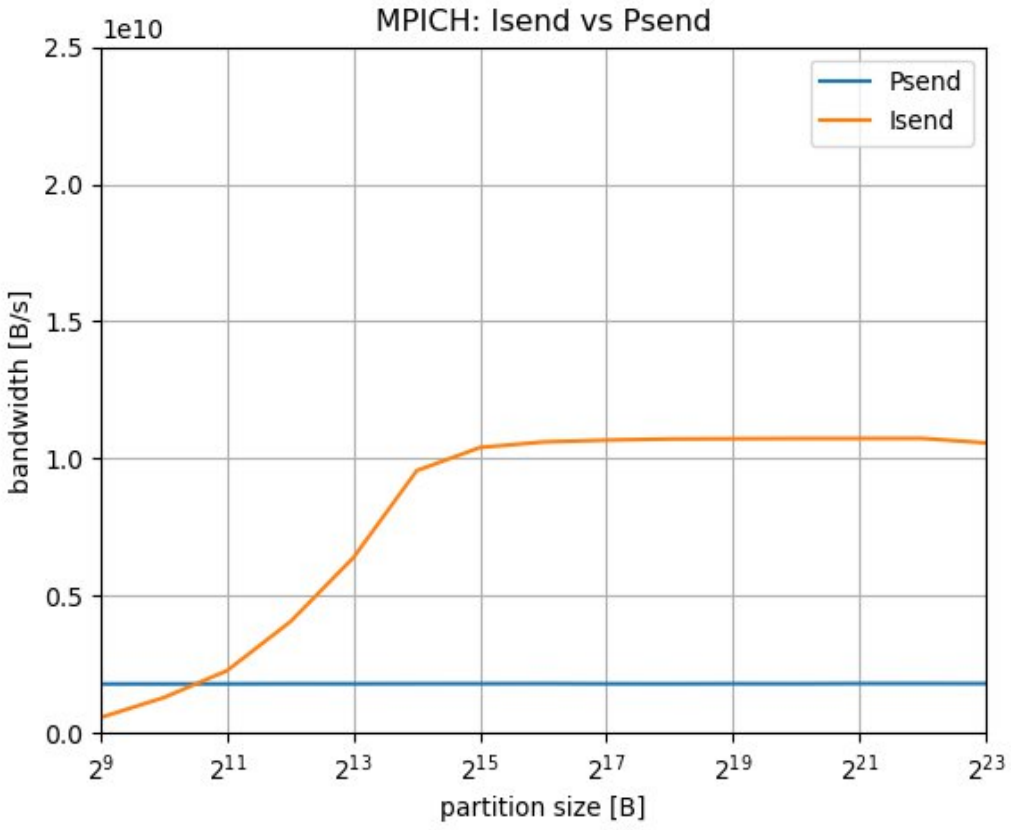


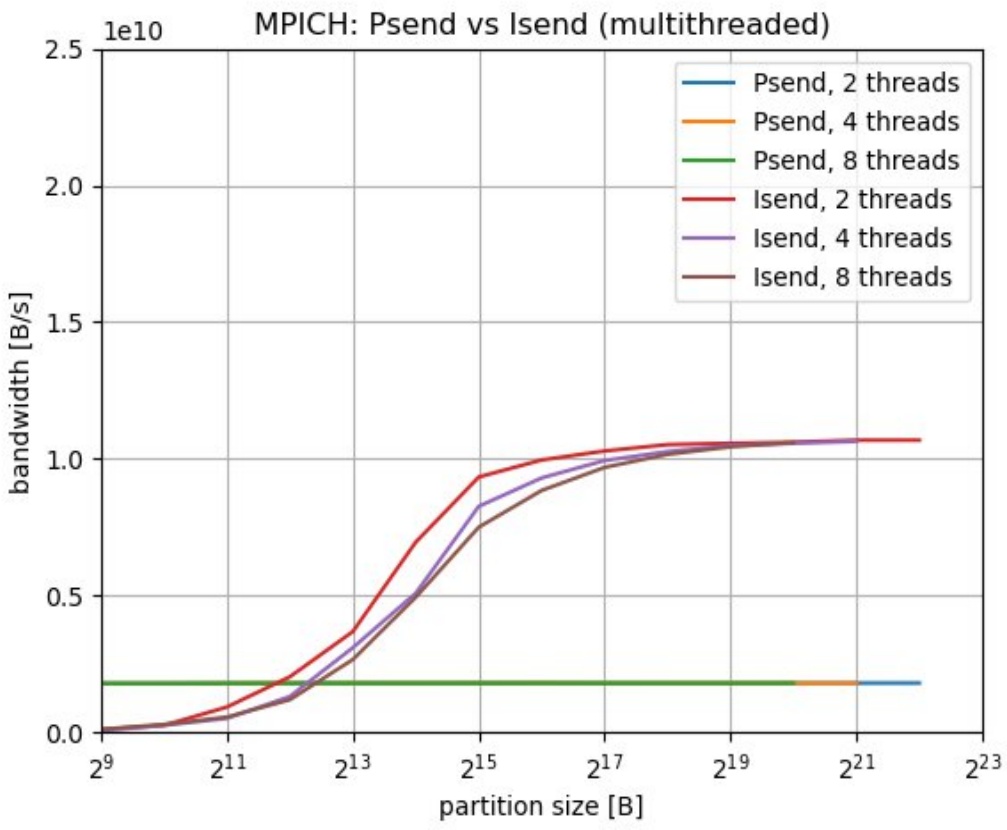
Results: Completion testing



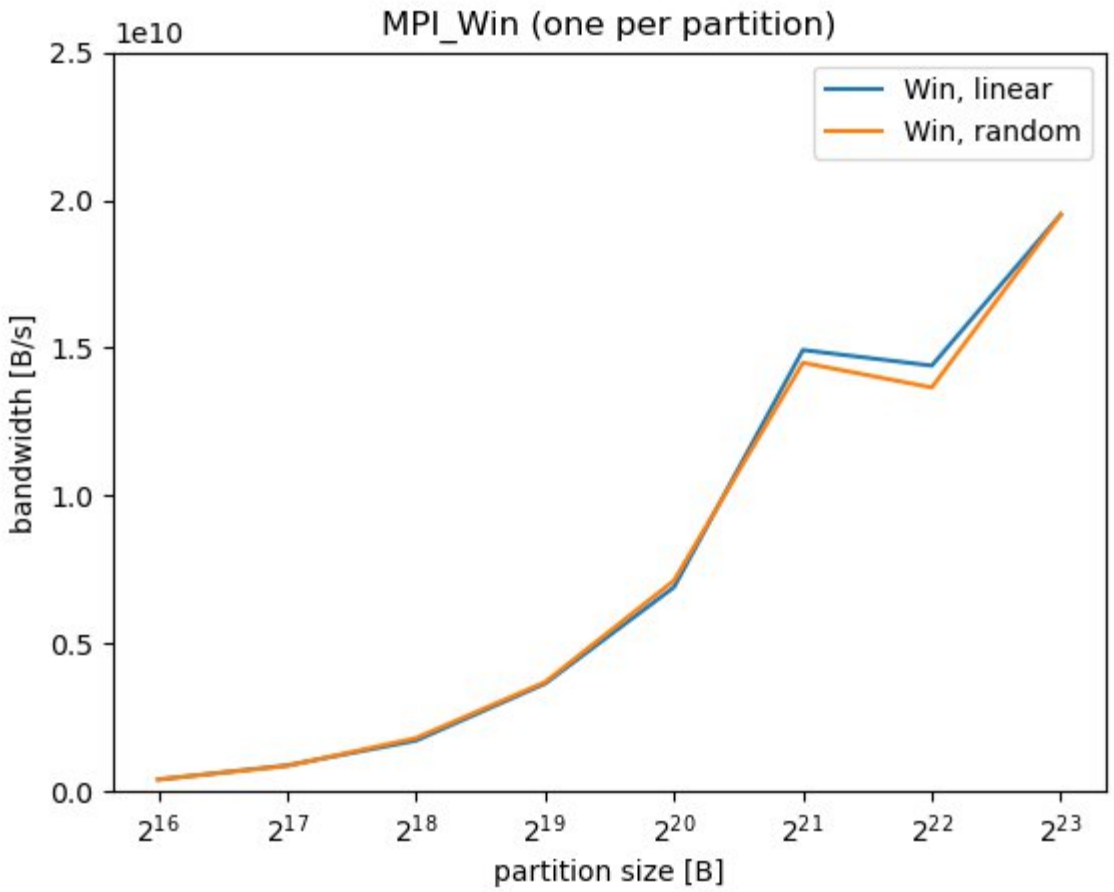


Results: Psend on MPICH-4.1.2

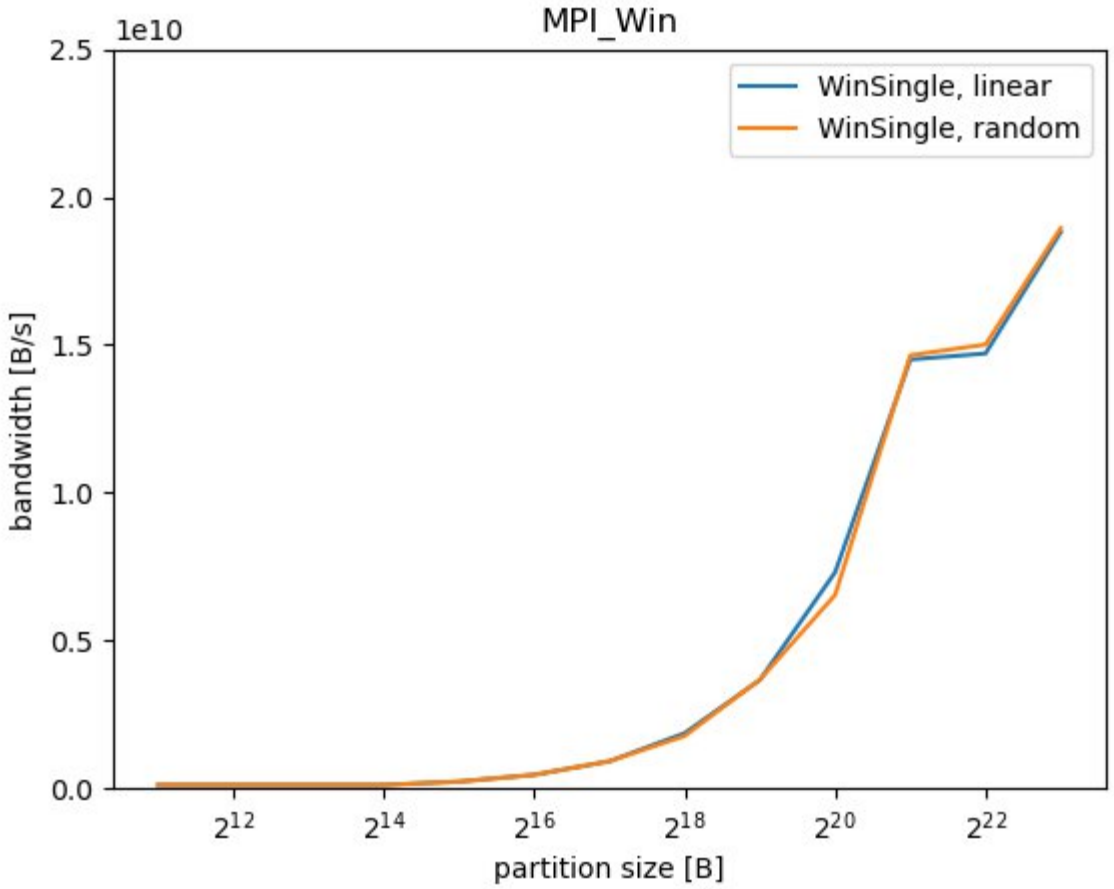




Results: MPI_Win per partition

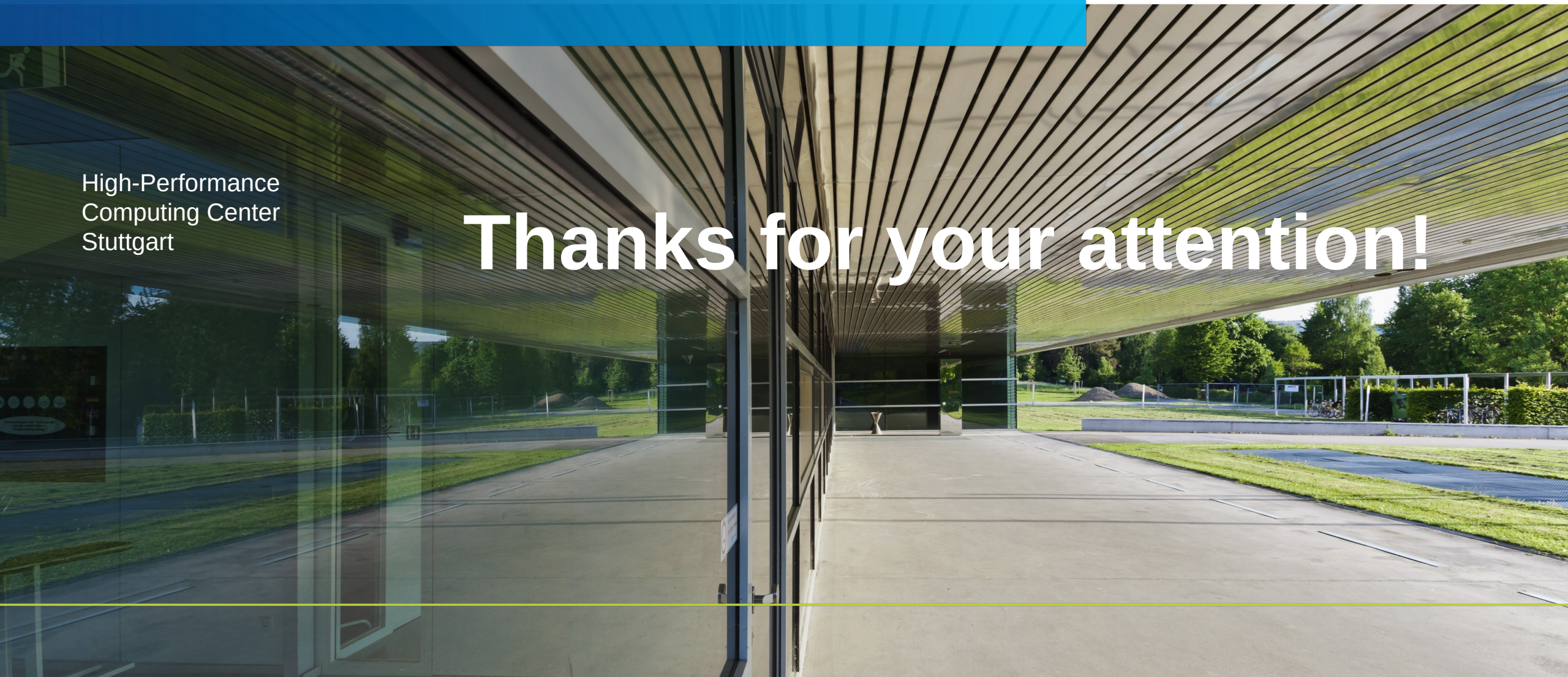


Results: MPI_Win for entire buffer, MPI_Put per partition



Conclusion

- OpenMPI
  - Partitioned Communication has similar performance as persistent Send when single-threaded
  - Partitioned Communication provides higher performance when using multiple threads
  - However, no further optimizations such as message aggregation
- Partitioned Communication in MPICH currently provides lower performance than other mechanisms



High-Performance
Computing Center
Stuttgart

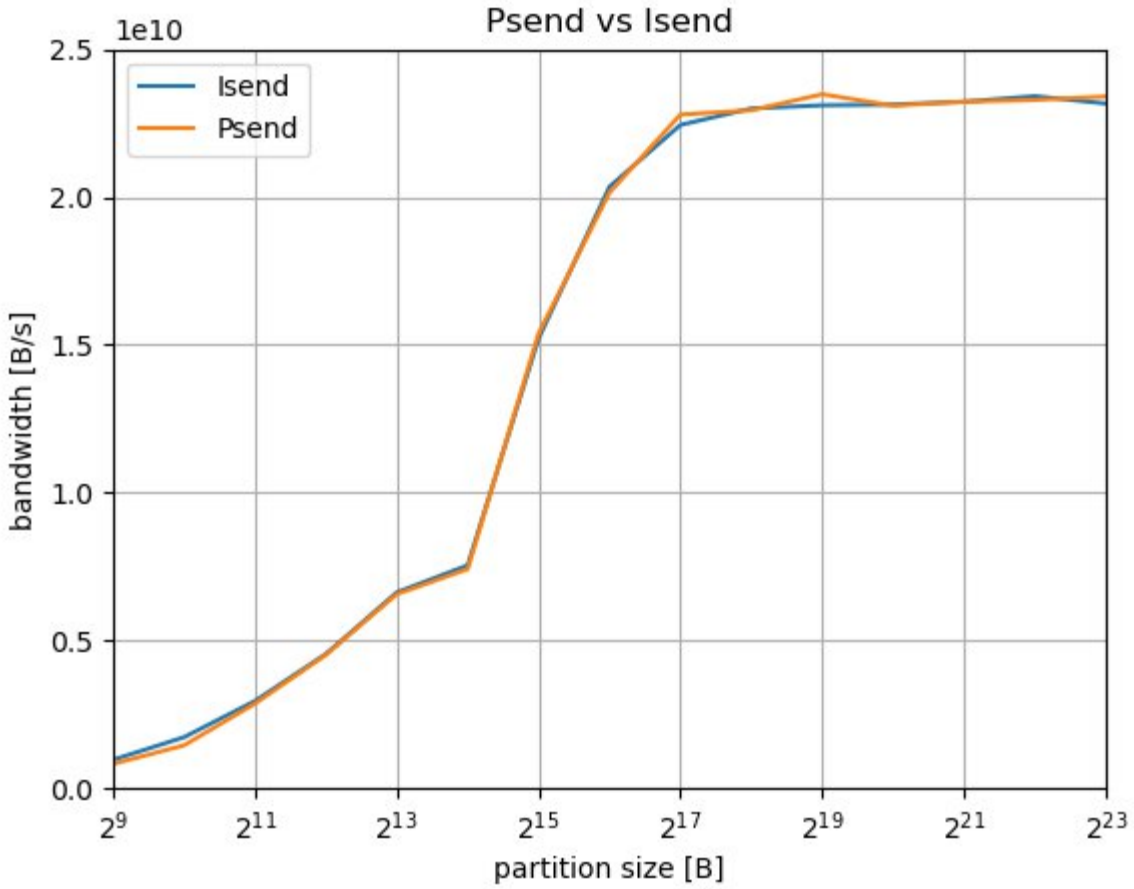
Thanks for your attention!

Appendix – OpenMPI progress

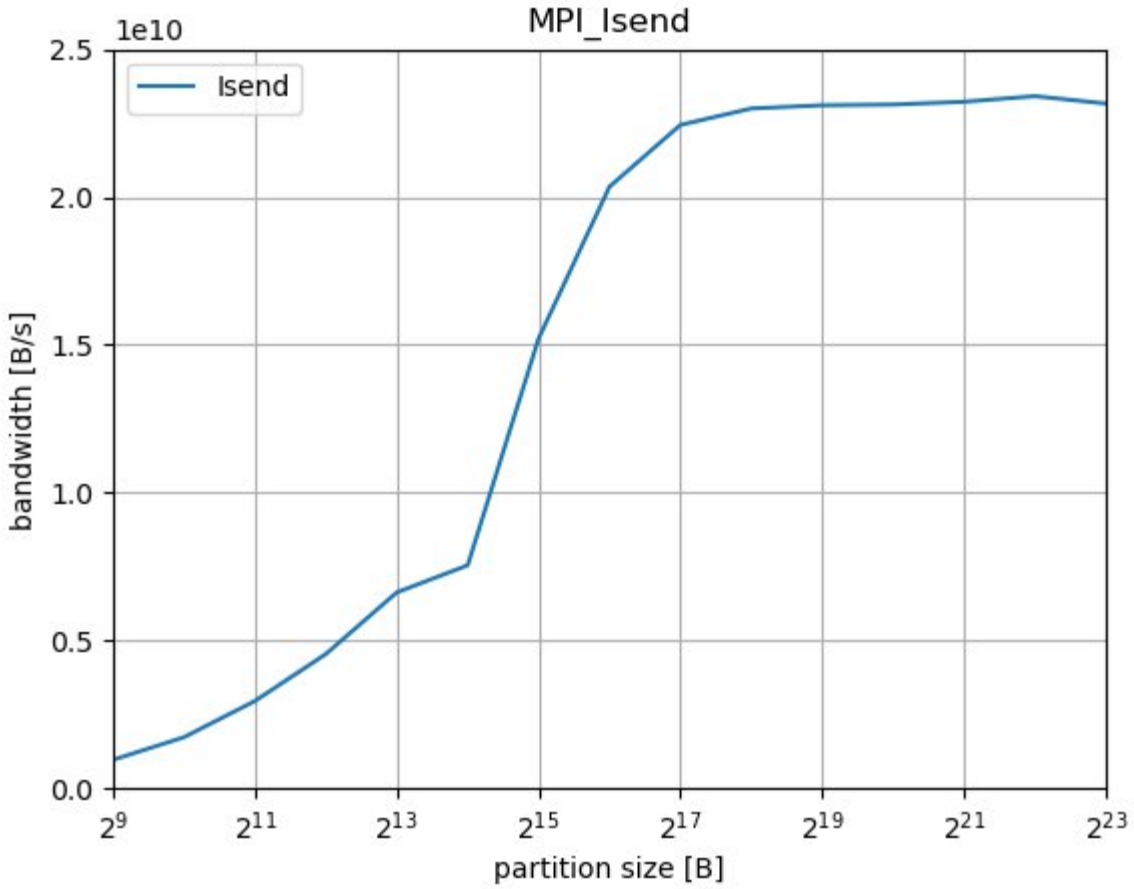
ompi > mca > part > persist > C part_persist.h

```
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           }
```

Appendix: Isend vs Psend



Appendix - Isend



Appendix: Isend vs Psend (multithreaded)