IXPUG 2022

IMPROVING MPI+THREADS WITH MPIX_STREAM

KEN RAFFENETTI, HUI ZHOU, YANFEI GUO, RAJEEV THAKUR
Programming Models and Runtime Systesms Group
Mathematics and Computer Science Division
Argonne National Laboratory

September 28, 2022

AGENDA

- MPI+Threads Background
- MPIX_Stream Proposal
- Message Rate Experimental Results
- MPI Asynchronous Progress
- Communication/Computation Overlap Experimental Results
- Conclusion
- Q&A

MPI+THREADS

- MPI does not explicitly say what a thread is, but for the purposes of this talk, we can assume that a thread is typical operating system thread thread
- MPI-2.0 (1997*) added thread support levels
  - MPI_THREAD_SINGLE
  - MPI_THREAD_FUNNELED
  - MPI_THREAD_SERIALIZED
  - MPI_THREAD_MULTIPLE
- In typical implementations, the first 3 levels are mostly the same
  - > MPI_THREAD_SINGLE may affect how external libraries are called. I.e. may call thread-safe APIs if there is more than one thread being used by the application
- MPI_THREAD_MULTIPLE is a decade(s) long struggle in optimization

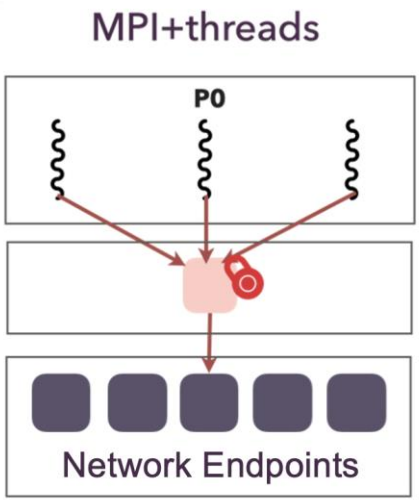
MPI+THREADS

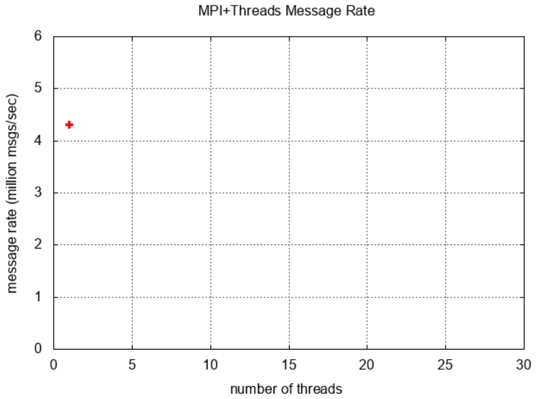
- MPI does not explicitly say what a thread is, but for the purposes of this talk, we can assume that a thread is typical operating system thread thread
- MPI-2.0 (1997*) added thread support levels
  - MPI_THREAD_SINGLE
  - MPI_THREAD_FUNNELED
  - MPI_THREAD_SERIALIZED
  - MPI_THREAD_MULTIPLE
- In typical implementations, the first 3 levels are mostly the same
  - > MPI_THREAD_SINGLE may affect how external libraries are called. I.e. may call thread-safe APIs if there is more than one thread being used by the application
- MPI_THREAD_MULTIPLE is a decade(s) long struggle in optimization

*Well before the multicore era of today!

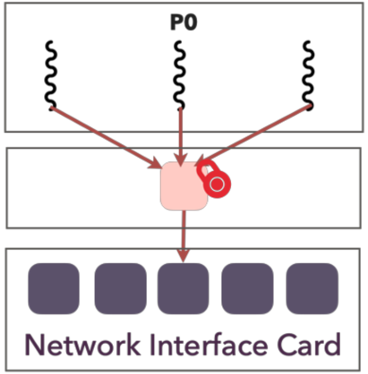
MPI+THREADS

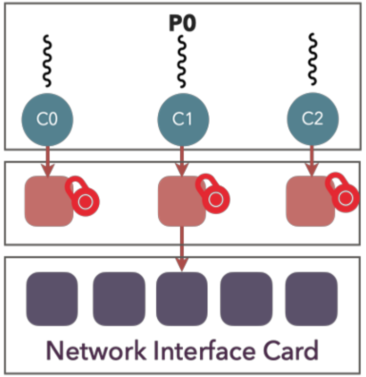
- Implementations for years were based on “global lock”
- Poor ☹ performance in small message rate test
- MPI does not recognize threads
- Threads contend for MPI resources
- Workarounds:
    - Avoid MPI_THREAD_MULTIPLE
    - Use MPI (Processes) Everywhere





VIRTUAL COMMUNICATION INTERFACE (VCI)





Multiple VCIs to preserve parallelism from
the application all the way to hardware

Rohit Zambre, Aparna Chandramowliswharan, and Pavan Balaji. 2020. How I learned to stop worrying about user-visible endpoints and love MPI. In Proceedings of the 34th ACM International Conference on Supercomputing (ICS '20).

VIRTUAL COMMUNICATION INTERFACE (VCI)

- Q: How can an application leverage multiple VCIs in MPICH?
- A: Many ways, but the most common is to map threads to communicators

```
for (int i = 0; i < num_threads; i++) {
    MPI_Comm_dup(MPI_COMM_WORLD, &comms[i]);
    pthread_create(&threads[i], NULL, worker_fn, &comms[i]);
}

void *worker_fn(void *comm_ptr)
{
    MPI_Comm comm = *(MPI_Comm *)comm_ptr;
    ...
}
```

VIRTUAL COMMUNICATION INTERFACE (VCI)

- Drawbacks to the implicitly mapped VCI approach

- Implicit allocation means the user has to verify, through documentation or experiments, that mapping threads to communicators will work with a given MPI implementation. May need to set extra environment variables, etc.
- MPI objects are still shareable between multiple threads (MPI_THREAD_MULTIPLE). The MPI library needs to protect them (locks), which can cost performance.

MPIX STREAM PROPOSAL

- `MPIX_Stream` identifies a serial execution context

```
int MPIX_Stream_create(MPI_Info info, MPIX_Stream *stream)
int MPIX_Stream_free(MPIX_Stream *stream)
```

- `info` can be `MPI_INFO_NULL`, identifies a generic thread context
- In the case of threads, it is the application's responsibility to ensure access to an MPIX_Stream is serialized. Essentially MPI_THREAD_SERIAL, but at the object-level, rather than all of MPI.
- Other use-cases, such as GPU stream awareness, not covered in this talk

Hui Zhou, Ken Raffenetti, Yanfei Guo, and Rajeev Thakur. 2022. MPIX Stream: An Explicit Solution to Hybrid MPI+X Programming. In Proceedings of the 29th European MPI Users' Group Meeting (EuroMPI/USA'22).

STREAM COMMUNICATOR

- Stream communicator is a communicator with local streams attached.

```
int MPIX_Stream_comm_create(MPI_Comm parent_comm,
        MPIX_Stream stream, MPI_Comm *stream_comm)
```

- MPIX streams are local, but communications are between pairs of them
- Otherwise, synchronization is unavoidable at receiver or sender.
- It okay for `stream` to be `MPIX_STREAM_NULL`.
- Conventional communicators are the same as stream communicators with `MPIX_STREAM_NULL` on every process.

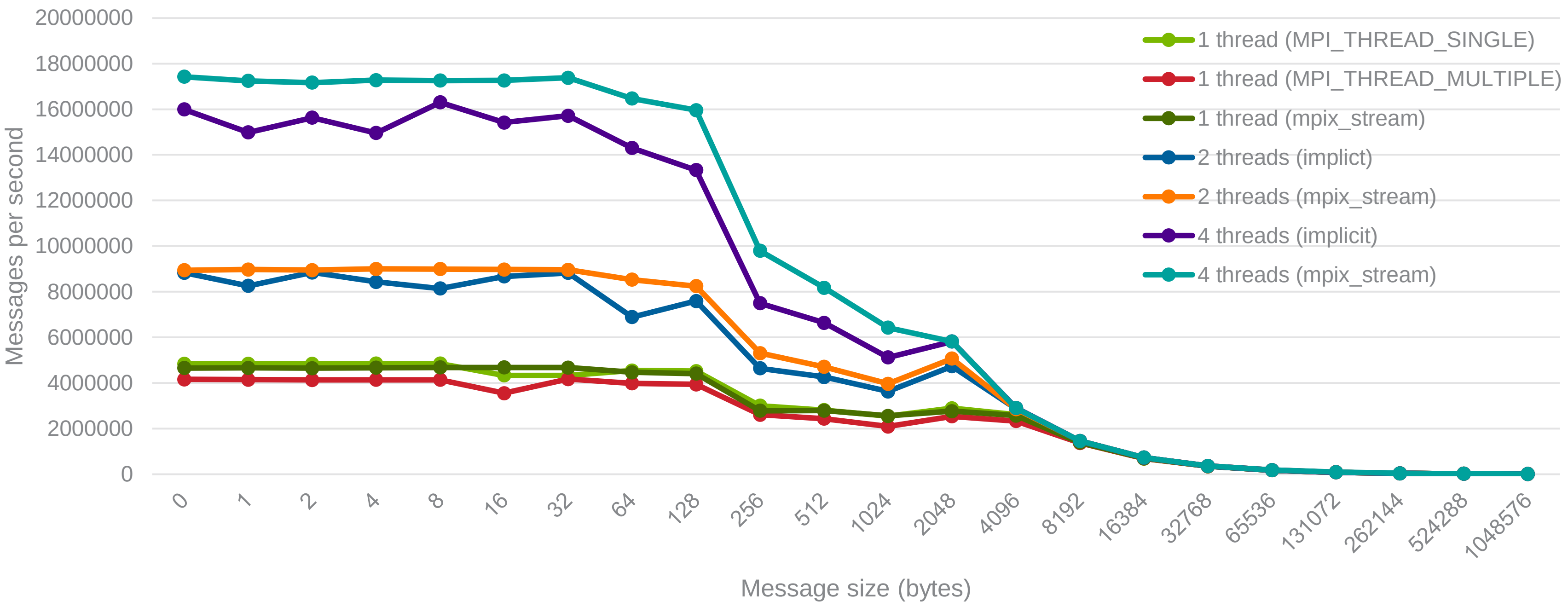
EXPERIMENTAL RESULTS

Benchmarks

- MT.ComB - Multi-Threaded (MT) Communication Benchmark
  - https://github.com/Mellanox/MT.ComB
  - Message rate tests for MPI point-to-point and one-sided APIs
  - Local modifications to use window-per-thread for RMA message rate
    - Communicator-per-thread already supported for point-to-point
  - Local modifications to support `MPIX_Stream` communicator

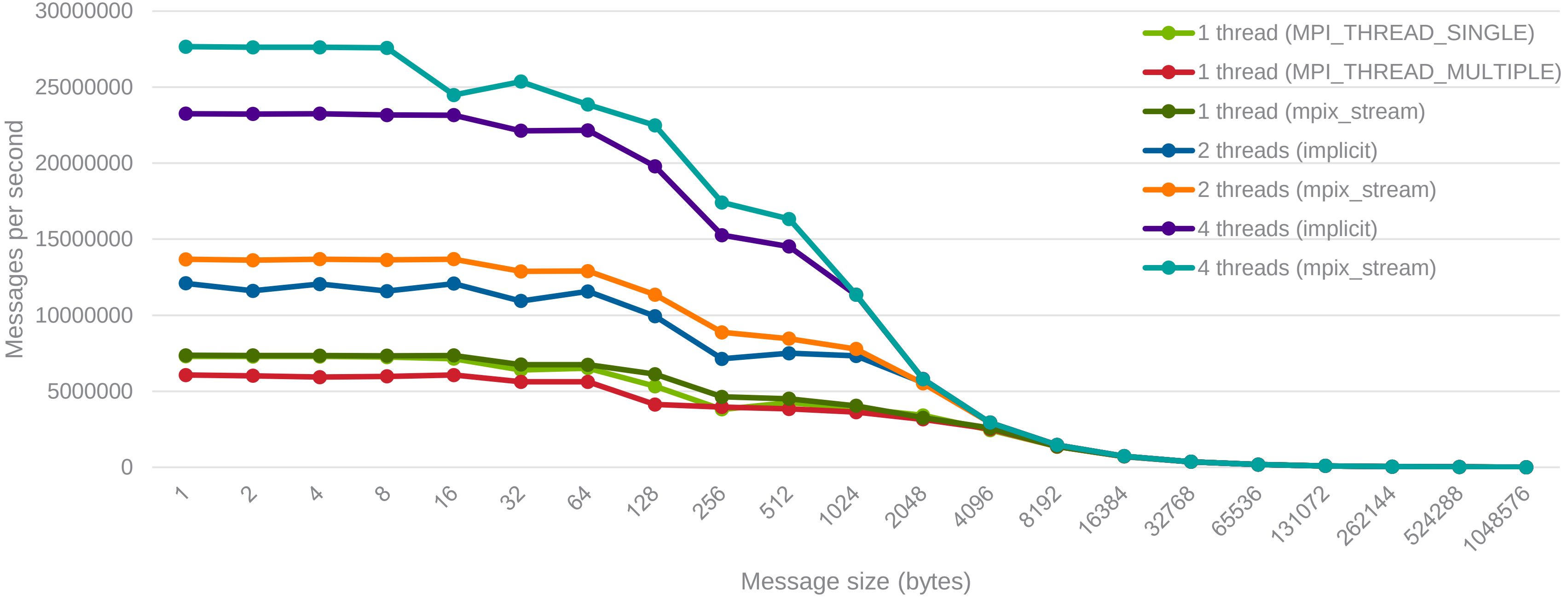
MT.COMB BENCHMARK (PT2PT MSG RATE)

Intel Xeon Platinum 8180M, ConnectX-6, nodes=2, ppn=1



MT.COMB BENCHMARK (RMA MSG RATE)

Intel Xeon Platinum 8180M, ConnectX-6, nodes=2, ppn=1



MPI PROGRESS MODEL

- Nonblocking MPI operations return immediately (`MPI_Isend`). An implementation is permitted to only make progress if/when the user calls `MPI_Test`/`MPI_Wait`.
- Overlapping computation with communication can be a challenge for applications. Does my MPI library make asynchronous progress? If not, how often should I call progress functions?
- MPI Standard progress requirements are esoteric and often unintuitive
- Progress threads are one approach
  - MPICH: `MPIR_CVAR_ASYNC_PROGRESS=1`
  - Spawns a thread during `MPI_Init`, makes progress on all communication resources until `MPI_Finalize`. Automatically raises the thread level to MPI_THREAD_MULTIPLE for the user.
  - Performs poorly in practice

MPIX STREAM FOR PROGRESS

- Progress communication on a stream instead of individual request(s)

```
int MPIX_Stream_progress(MPIX_Stream stream)
```

- Application can create/join progress threads as needed
- Coordinated stream progress needs no additional thread-safety from the implementation
- Progress thread does not need to be aware of outstanding requests

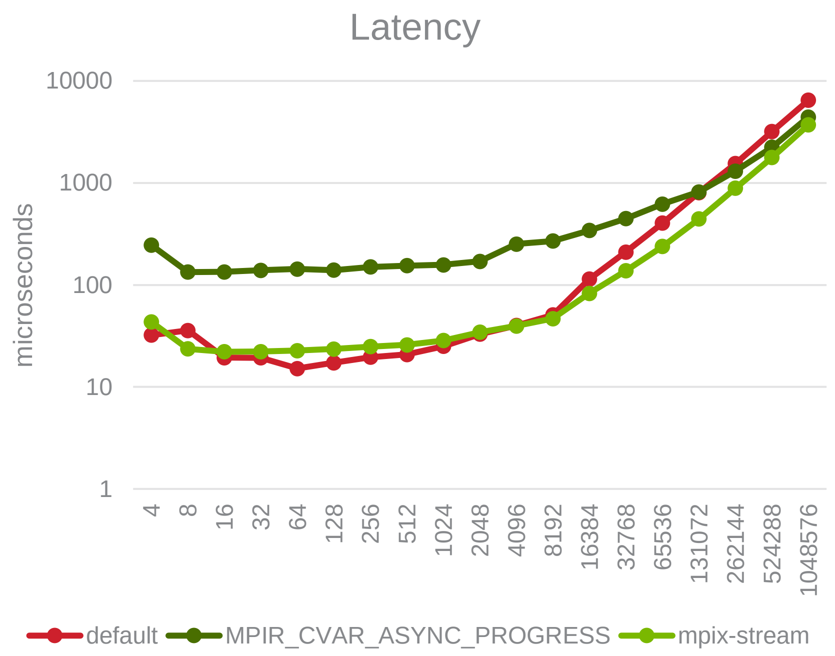
EXPERIMENTAL RESULTS

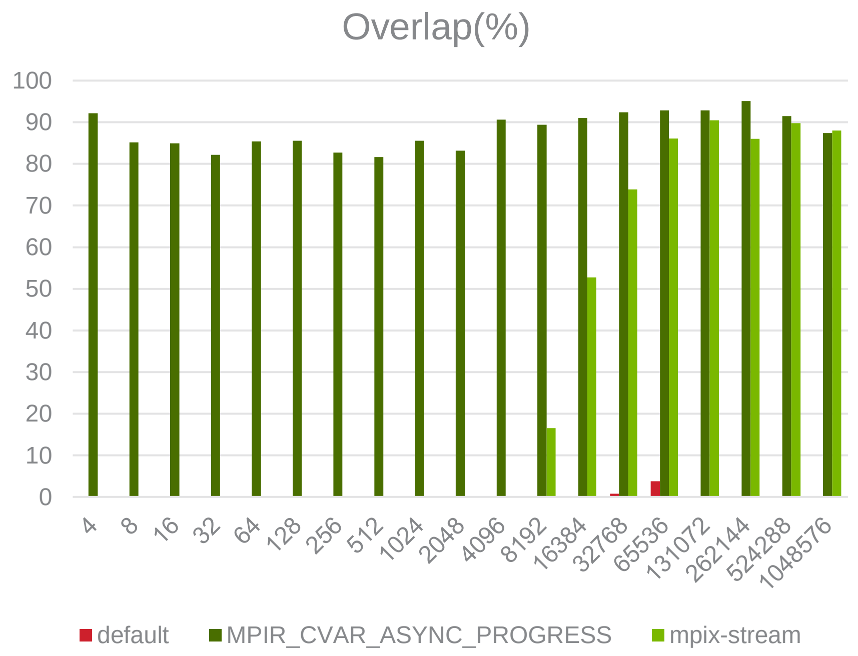
Benchmark

- OSU Microbenchmarks
  - https://mvapich.cse.ohio-state.edu/benchmarks/
  - Full suite of MPI microbenchmarks
  - Nonblocking collective benchmarks simulate communication/computation overlap by first measuring the base latency, then adding in dummy computation roughly equal in time to the communication latency.
    - Overlap percentage included in report
  - Local modifications to use MPIX_Stream-based progress thread

OSU MICROBENCHMARKS MPI_IALLREDUCE

Intel Xeon Platinum 8180M, Connect-X6, nodes=2, ppn=28





QUESTIONS?

- Website
  - www.mpich.org
- Mailing Lists
  - lists.mpich.org
- Github
  - http://github.com/pmodels/mpich
  - Try it!
  - Submit an issue or pull request!
- Email
  - raffenet@anl.gov