MPIX STREAM: AN EXPLICIT SOLUTION TO MPI+X

Hui Zhou, Ken Raffenetti, Yanfei Guo, and Rajeev Thakur

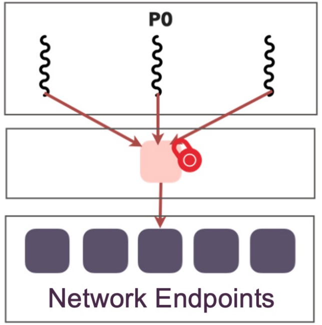
Sept. 26, 2022

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

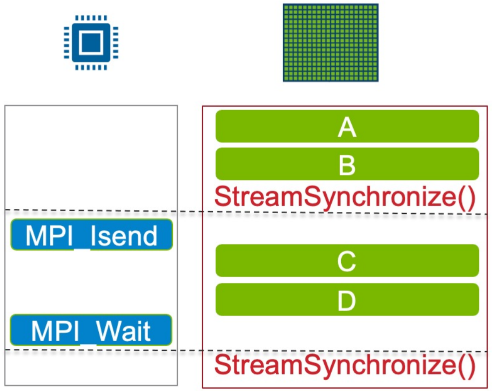
OUTLINE

- The Performance Challenges
  - MPI+Thread
  - MPI+GPU
- Background:
  - The Endpoints Proposal
  - Implicit Solutions for MPI+Thread
- The MPIX Stream Proposal
  - MPIX Stream
  - Stream Communicator
  - Stream Enqueue Operations
  - Multiplex Stream Communicator
- Performance
- Code Examples

MPI+Thread



MPI+GPU

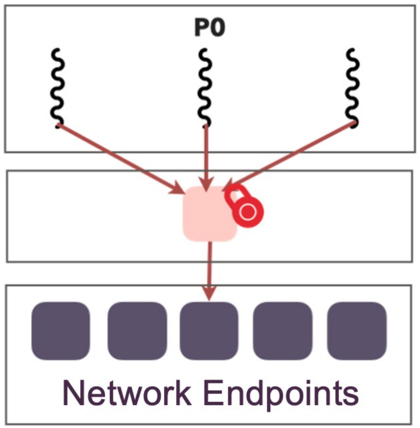


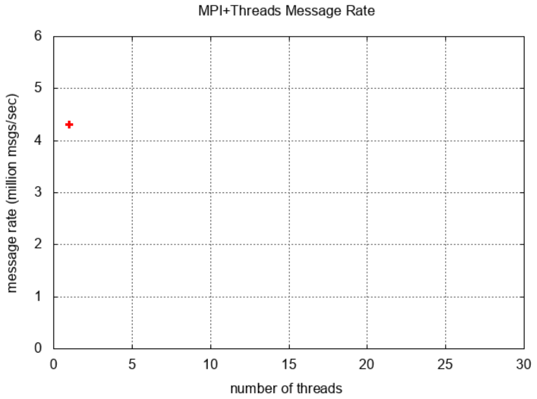
PERFORMANCE CHALLENGE:
MPI+THREAD

- Miserable ☹ performance in small-message rate with `MPI_THREAD_MULTIPLE`
- MPI does not recognize thread context
- Threads contend on global lock
- Work around for this long-running issue:
  - Avoid `MPI_THREAD_MULTIPLE`
  - Avoid MPI within parallel regions
  - Avoid small messge
- *The work-arounds are getting less acceptable with modern trends in HPC*

e.g. fat node, heterogeneous, data parallel, task parallel, event dispatching

MPI+Thread

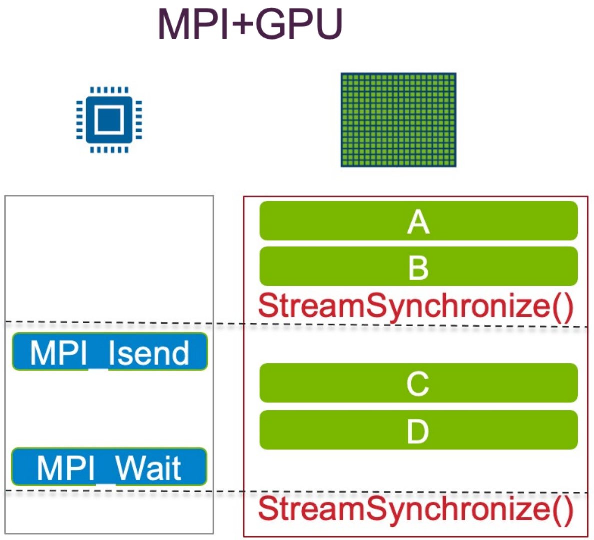




PERFORMANCE CHALLENGE:

MPI+GPU

- MPI does not recognize the offloading context
- MPI forces the use of heavy synchronization
- MPI prevents latency hiding of kernel launching
- Solution: asynchronously launch the MPI communication and use light weight synchronization e.g. triggered operation
- Require explicit input of GPU context into MPI



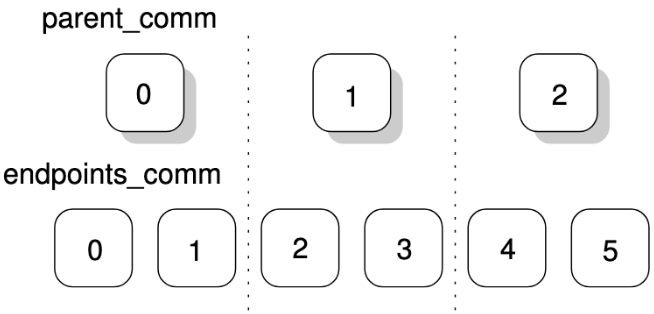
WHAT IS COMMON BEHIND MPI+X?

- The “X” brings new execution contexts beyond MPI processes
- MPI + Thread
  - Threads
    - Serial within, concurrent between
- MPI + GPU
  - Streams
    - Serial within, concurrent between
    - Asynchronous to CPU
  - Graphs
    - Asynchronous to CPU
- It’s essential to avoid extra synchronization
- *How can we get the performance when it depends on the knowledge of extra contexts, but we are not telling MPI about it?*

When a thread is executing one of these (MPI) routines, if another concurrently running thread also makes an MPI call, the outcome will be as if the calls executed in some order

THE ENDPOINTS PROPOSAL (2013)

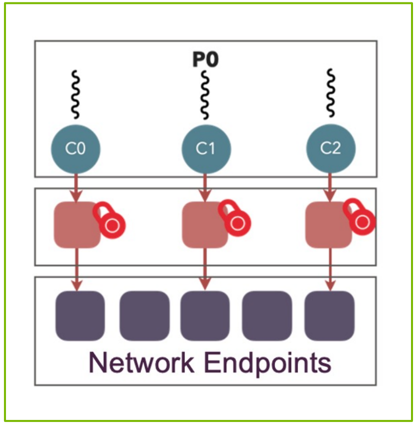
- Synoposis
  - Build endpoints-comm, where each thread is as if a *virtual process*



- The good
  - Very small API addition
  - Explicit addressing of thread context
- However, the devil is in the details
  - MPI is still not acknowledging threads
  - Inflating threads to *virtual processes* has implications to MPI semantics at large, e.g. comm dup, comm group, inter comm
  - Threads may not be as persistent as process

IMPLICIT SOLUTION TO MPI+THREAD

- Currently adopted by major MPI implementations
- MPI already provides sufficient means to express parallelism
  - Communicators
  - Source/Destination ranks
  - Tags
- We can achieve perfect network endpoints mapping if user match their threads with e.g. communicators
- But, communicators (or ranks and tags) are the wrong semantics for execution context
- Can't match the performance of MPI_THREAD_SINGLE (but we should)



PROPOSAL: MPIX STREAM

- `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 CPU context (i.e. thread)
- It can be specialized into offloading context, e.g. for `cudaStream_t`

```
MPI_Info_create(&info);
MPI_Info_set(info, "type", "cudaStream_t");
MPIX_Info_set_hex(info, "value", &stream, sizeof(stream));

MPIX_Stream_create(info, &mpi_stream);
```

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)
```

- The stream communicator specifies both the local and remote network endpoints
- Otherwise, synchronizations are unavoidable at receiver or sender.
- It's backward compatible
  – Conventional communicators are the same as stream communicators with `MPIX_STREAM_NULL` on every process.

USING STREAM COMMUNICATOR

- Existing MPI functions will work with stream communicator, but remember -
  - Illegal to operate on an MPIX_Stream concurrently
  - Use locks if necessary

In a sense, we are shifting the burden of thread synchronization from implementation to application. We argue that it is less complex and more effective on the application side.

GPU ENQUEUE OPERATIONS

Showing Point-to-Point Communications

- Alias APIs for async launching MPI communications

```
int MPIX_Send_enqueue(buf, count, datatype, dest, tag, comm)
int MPIX_Recv_enqueue(buf, count, datatype, source, tag,
                     comm, status)
int MPIX_Isend_enqueue(buf, count, datatype, dest, tag,
                     comm, request)
int MPIX_Irecv_enqueue(buf, count, datatype, source, tag,
                     comm, request)
int MPIX_Wait_enqueue(request, status)
int MPIX_Waitall_enqueue(count, array_of_requests,
                     array_of_statuses)
```

COMPONENTS/CONCEPTS

- Execution Context
- Queue / Graph
- Launching graph onto execution context and synchronization in implementation
- (try not to reinvent the MPI operations)

WHY DO WE NEED NONBLOCKING ENQUEUE?

Two different types of async

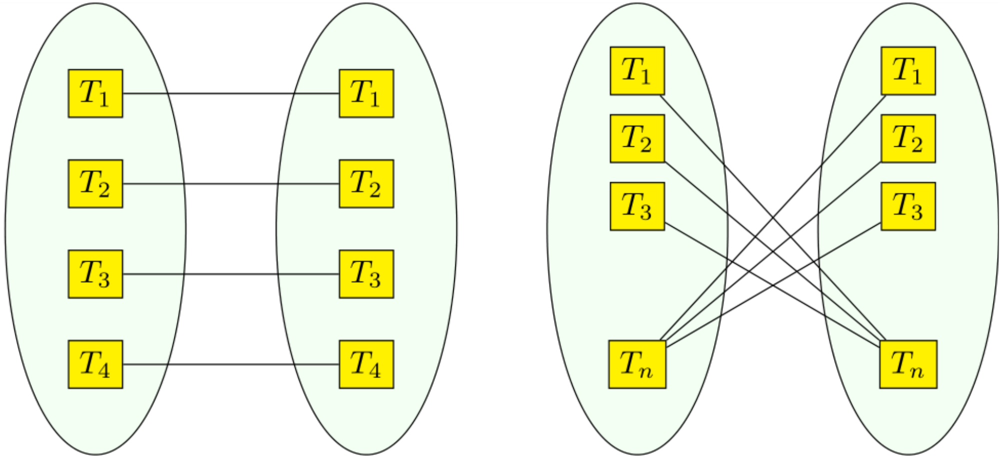
- MPI Nonblocking operations
  - Asynchronous regarding data buffer
- GPU Async kernel launching
  - Asynchronous regarding launching on the GPU
- Two async models are orthogonal

MPIX_Isend_enqueue

MPIX_Wait_enqueue

MULTIPLEX STREAM COMMUNICATORS

1-to-1 pattern



N-to-1 pattern

```
int MPIX_Stream_comm_create_multiple(MPI_Comm parent_comm,
                    int count, MPIX_Stream local_streams[],
                    MPI_Comm *stream_comm)
```

USING MULTIPLEX STREAM COMMUNICATOR

- A set of point-to-point APIs that use rank+index to explicitly address local and remote streams

```
int MPIX_Stream_send(buf, count, datatype, dest, tag, comm,
                     int src_idx, int dst_idx)
int MPIX_Stream_recv(buf, count, datatype, source, tag, comm,
                     int src_idx, int dst_idx, status)
int MPIX_Stream_isend(buf, count, datatype, dest, tag, comm,
                     int src_idx, int dst_idx, request)
int MPIX_Stream_irecv(buf, count, datatype, source, tag, comm,
                     int src_idx, int dst_idx, request)
```

MPI+THREAD PERFORMANCE

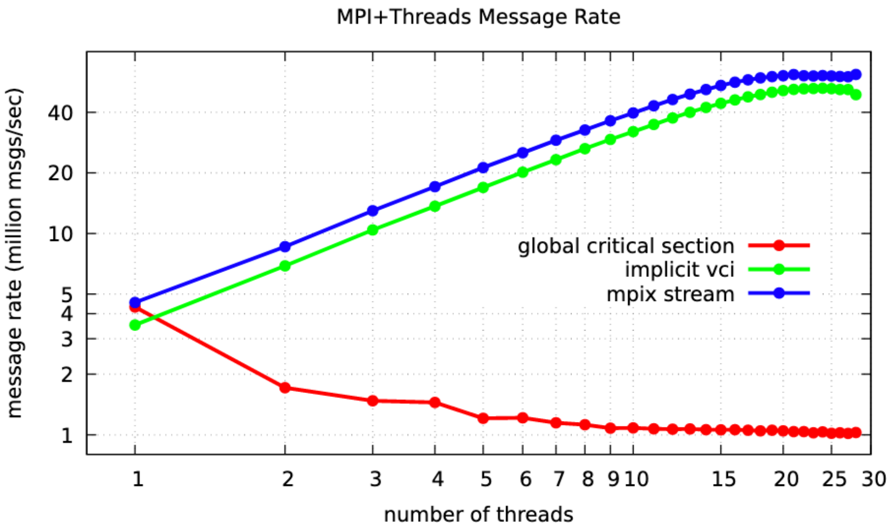
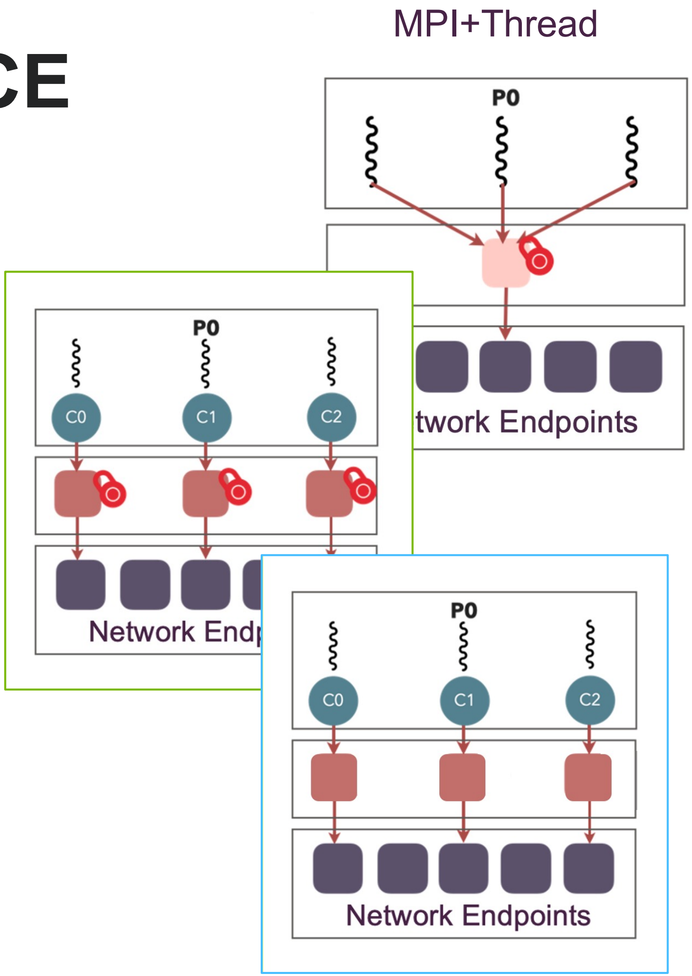
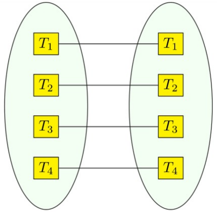


Figure 3: Multithread message rate on 8-byte messages using MPI_Isend/MPI_Irecv.



EXAMPLE CODE: MPI + OPENMP




```
MPIX_Stream streams[NT];
MPI_Comm comms[NT];
for (int i = 0; i < NT; i++) {
    MPIX_Stream_create(MPI_INFO_NULL, &streams
        [i]);
    MPIX_Stream_comm_create(MPI_COMM_WORLD,
        streams[i], &comms[i]);
}

        ...

for (int i = 0; i < NT; i++) {
    MPIX_comm_free(&comms[i]);
    MPIX_Stream_free(&streams[i]);
}
```

```
#pragma omp parallel num_threads (NT)
{
    int id = omp_get_thread_num () ;
    char buf [100];
    int tag = 0;

    if ( rank == 0) {
        MPI_Send (buf, 100, MPI_CHAR, 1,
tag, comms[id]);

    } else if ( rank == 1) {
        MPI_Recv ( buf, 100, MPI_CHAR, 0,
tag, comms[id], MPI_STATUS_IGNORE);

    }
}
```

EXAMPLE CODE: MPI + CUDA

~~StreamSynchronize~~

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "type", "cudaStream_t");
MPIX_Info_set_hex(info, "value", &stream,
    sizeof(stream));

MPIX_Stream mpi_stream;
MPIX_Stream_create(info, &mpi_stream);

MPI_Info_free(&info);

MPI_Comm stream_comm;
MPIX_Stream_comm_create(MPI_COMM_WORLD,
    mpi_stream, &stream_comm);

        ...

MPI_Comm_free(&stream_comm);
MPIX_Stream_free(&mpi_stream);

cudaStreamDestroy(stream);
```

```
/* Rank 0 sends x data to Rank 1 , Rank 1
   performs a * x + y and checks result */

if ( rank == 0) {

    MPIX_Send_enqueue(x, N, MPI_FLOAT, 1, 0,
stream_comm);

} else if (rank == 1) {

    cudaMemcpyAsync(d_y, y, N * sizeof(float),
cudaMemcpyHostToDevice, stream);
    MPIX_Recv_enqueue(d_x, N, MPI_FLOAT, 0, 0,
stream_comm, MPI_STATUS_IGNORE);
    saxpy<<<(N+255)/256, 256, 0, stream>>>(N,
a, d_x, d_y);
    cudaMemcpyAsync(y, d_y, N * sizeof(float),
cudaMemcpyDeviceToHost, stream);

}
```

SUMMARY

- New MPI object – MPIX_Stream – to represent serial execution context
- Stream communicator to encapsulate communication pairs
- Works with existing MPI functions with the addition of new semantics
- Enables explicit optimizations from applications

ACKNOWLEDGEMENT

Thanks for very useful feedback and discussions from –

- MPI Forum Hybrid working group
- Intel MPI team
- HPE Cray MPI team

Q & A