Should we enable MPI RMA calls from the GPU?

Presenter: Maria J. Garzaran, Principal Engineer

Future of MPI RMA Workshop (FoRMA'22)

Outline

- Should we support RMA operations (MPI_Put, MPI_Get,..) to execute from the GPU?
- Some experimental results with MPICH-OFI for basic RMA operations

Partition Communication: Example in MPI 4.0

```
if (myrank == 0) {
  MPI_Psend_init(message, partitions, count, xfer_type,
     dest, tag, info, MPI_COMM_WORLD, &request);
  MPI_Start(&request);

  #pragma omp parallel for shared(request) num_threads(NUM_THREADS)
  for (int i = 0; i < partitions; i++) {
      /* compute and fill partition #i, then mark ready: */
      MPI_Pready(i, request);
  }
  while (!flag) {
     /* Do useful work */
     MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
  }
  MPI_Request_free(&request);
}
```

```
if (myrank == 1) {
   MPI_Precv_init(message, partitions, count, xfer_type
      source, tag, info, MPI_COMM_WORLD, &request);
   MPI_Start(&request);

   while (!flag) {
    /* Do useful work */
      MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
    }
  MPI_Request_free(&request);
}
```

Partition Communication

- The Hybrid WG has been discussing extending Partitioned Communication to support GPU-initiated communication
- In practice, that means that:
  - Application is using two-sided semantics
  - The communication (issued from the GPU) is using one-sided semantics
- If so, shouldn't we also enable RMA to execute from the GPU?

Partition Communication using SYCL

```
if (myrank == 0) {
  MPI_Psend_init(message, partitions, count, xfer_type,
    dest, tag, info, MPI_COMM_WORLD, &request);
  MPI_Start(&request);

  mQueue.submit([&](sycl::handler &h) {
    h.parallel_for(partitions, [=](id<1> i) {
      MPI_Pready(i, request);
    });
  });

  mQueue.wait();
  while (!flag) {
    /* Do useful work */
    MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
  }
  MPI_Request_free(&request);
}
```

```
if (myrank == 1) {
   MPI_Precv_init(message, partitions, count, xfer_type
     source, tag, info, MPI_COMM_WORLD, &request);
    MPI_Start(&request);

  while (!flag) {
   /* Do useful work */
     MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
   }
  MPI_Request_free(&request);
}
```

Partition Communication using SYCL – GPU-initiated communication

```
MPI_request req[2];
MPI_Prequest preq;
MPI_Psend_init(.., &req[0]);
MPI_Precv_init(...,&req[1]);
MPI_Prequest_create(req[0], MPI_INFO_NULL, &preq);

while (...) {
   MPI_Startall (2, req);
   MPI_Pbuf_prepare_all (2, preq);

    mQueue.submit([&](sycl::handler &h) {
       h.parallel_for(partitions, [=](id<1> i) {
          MPI_Pready(i, preq);
       });
    });

    mQueue.wait();
    MPI_Waitall (2, req);
}
```

Creates a request that can be accessed from the device

Tag matching and Keys exchange only needed once

Guarantees buffer is ready on receiver side

Executes from the Device

This is really a put operation

MPI RMA: GPU-initiated communication

Regular CPU code

```
MPI_Win_create (buffer, partitions*sizeof(int),
sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win);

MPI_Win_fence(0, win);
if (myrank == 0) {
  for (int p=0;p<partitions;++p)
    MPI_Put (&data, …, win);
}

MPI_Win_fence(0, win);
MPI_Win_free(&win);
```

SYCL code

```
MPI_Win_create (buffer, n*sizeof(int), sizeof(int),
MPI_INFO_NULL, MPI_COMM_WORLD, &win);

MPI_Win_fence(0, win);
if (myrank == 0) {
   mQueue.submit([&](sycl::handler &h) {
      h.parallel_for(partitions, [=](id<1> i) {
        MPI_Put (&data,…, win);   Executes from the Device
      });
    });
   mQueue.wait();
}

MPI_Win_fence(0, win);
MPI_Win_free(&win);
```

MPI RMA: Memory Allocation in the Device

Memory allocation functions need to take in the info argument information to determine:

1. Type of memory allocation (system, host, device, or shared),
2. Kind (CUDA, Level Zero, HIP, …),
3. Device/Subdevice in the system where to allocate the data.

MPI_Win_allocate (size, disp_unit, info, comm, baseptr, win)

MPI_Win_allocate_shared (size, disp_unit, info, comm, baseptr, win)

MPI_Alloc_mem (size, info, baseptr)

RMA Implementation in MPICH-OFI

RMA Design and implementation in MPICH-OFI can be found in:

"Efficient implementation of MPI-3 RMA over openFabrics interfaces",

by Hajime Fujita, Chongxiao Cao, Sayantan Sur, Charles Archer, Erik Paulson, and Maria Garzaran, PARCO Volume 87, Issue C, Sep 2019.

Early Experimental Results

Experimental Setup

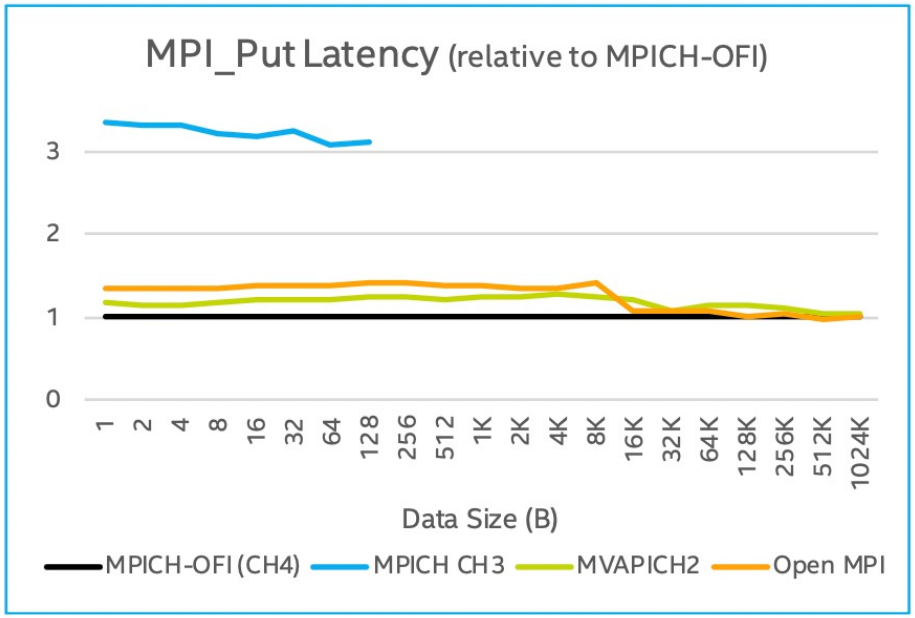
- CPU: Intel Xeon Gold6138
- Interconnect Intel Omni-Path Architecture
- OS: SUSE Linux Enterpirse Server 12 SP3
- Compiler: GCC 8.1.0 (-O3, LTO enabled)
- OFI libfabric: 1.6.1
- PSM2: 10.3.58
- IMB Benchmarks 2018 Update 1

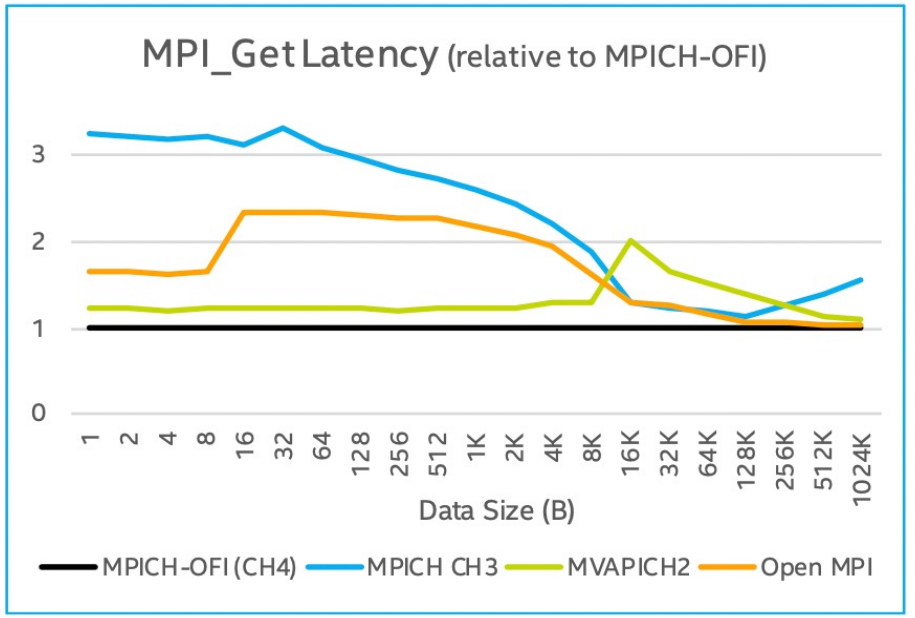
Comparison between:

- MPICH-OFI. MPICH 3.3b3
- MPICH CH3. MPICH 3.3b3 configured with CH3
- MVAPICH2 version 2.3. Configured to use PSM2 directly
- Open MPI version 3.1.1 with PSM2 (since it was faster than using OFI).

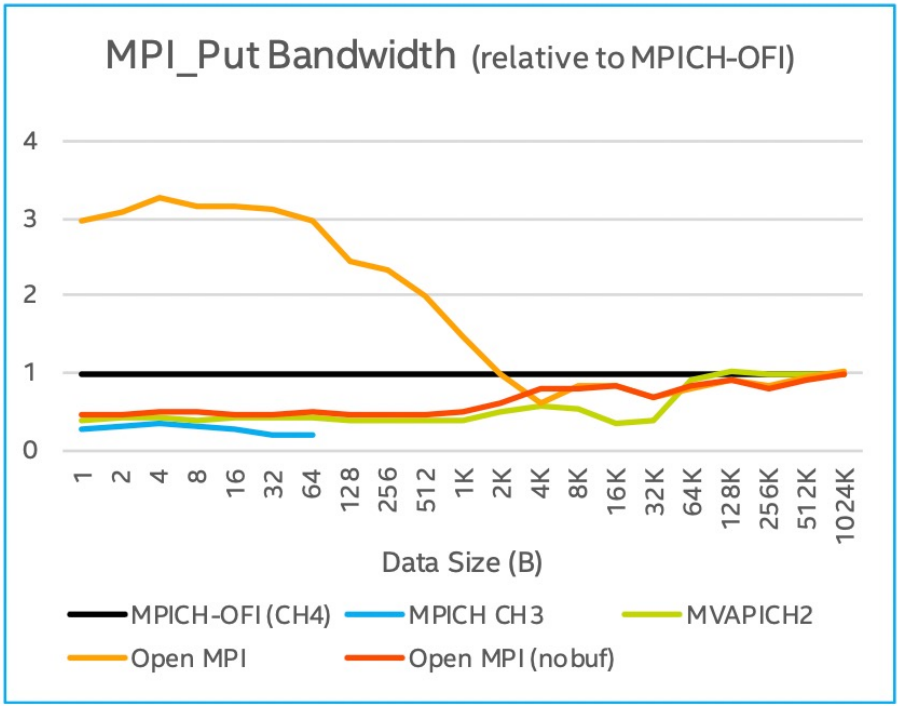
See all experimental setup details in the paper

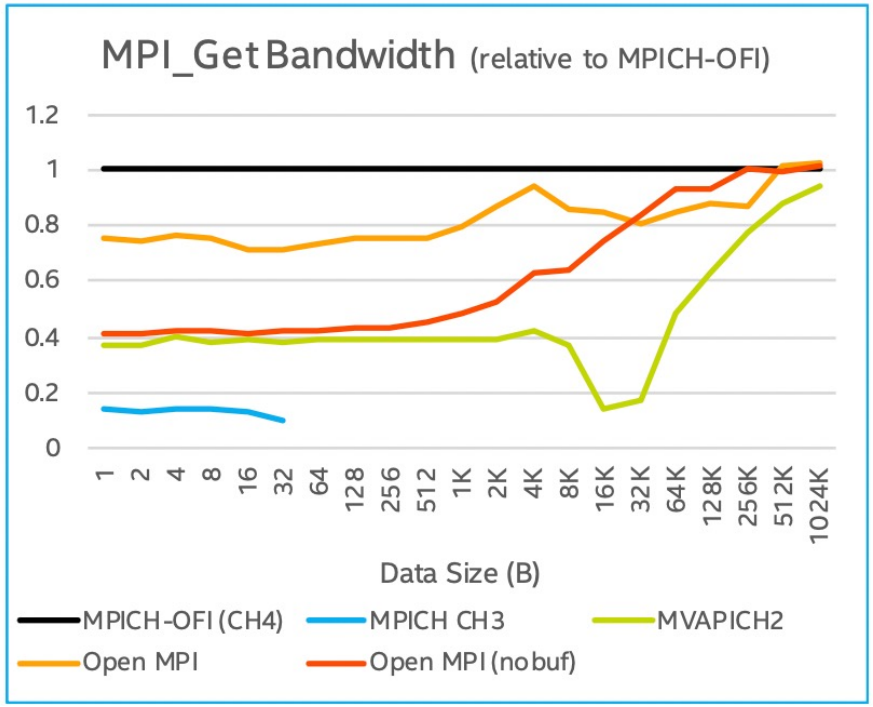
Latency


MPI_Put Latency (relative to MPICH-OFI)
3
2
1
0
1
2
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
256K
512K
1024K
Data Size (B)
MPICH-OFI (CH4)
MPICH CH3
MVAPICH2
Open MPI


MPI_Get Latency (relative to MPICH-OFI)
3
2
1
0
1
2
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
256K
512K
1024K
Data Size (B)
MPICH-OFI (CH4)
MPICH CH3
MVAPICH2
Open MPI

Bandwidth


MPI_Put Bandwidth (relative to MPICH-OFI)
4
3
2
1
0
1
2
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
256K
512K
1024K
Data Size (B)
MPICH-OFI (CH4)
MPICH CH3
MVAPICH2
Open MPI
Open MPI (nobuf)


MPI_Get Bandwidth (relative to MPICH-OFI)
1.2
1
0.8
0.6
0.4
0.2
0
1
2
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
256K
512K
1024K
Data Size (B)
MPICH-OFI (CH4)
MPICH CH3
MVAPICH2
Open MPI
Open MPI (nobuf)

Questions?

Notices & Disclaimers

Performance results are based on testing as of August 9th, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product or component can be absolutely secure.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the per- formance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Optimization Notice: Intels compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Intel technologies may require enabled hardware, software or service activation.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.