Hewlett Packard
Enterprise

Getting the Most out of HPE Cray MPI

Tim Mattox, HPC Performance Engineer, HPE Frontier Center of Excellence (COE)

February 16, 2023

MPI Best Practices (from a former MPI developer… me)

- Post non-blocking receives before their matching sends
- Check the MPI error/return codes
  - Things can go wrong (and will someday)
- Don’t abuse the MPI layer… It isn’t magic
  - Finite number of Communicators
  - Finite number of Tags (MPI_TAG_UB could be as low as 32,767)
  - Finite number of pending receives/sends
  - Give MPI a chance to make progress (details on another slide)
- Avoid *unnecessary* use of MPI_ANY_SOURCE
  - But use it if you need to, rather than posting a bunch of “just in case” receives
- Don’t roll-your-own MPI Collectives
  - File a bug if performance is not what you expect (unless you are an HPL developer :-)
  - Why? Roll-your-own won’t take advantage of hardware collective acceleration
  - Also, have a look at the “new” non-blocking collectives in MPI-3

How to post non-blocking receives before their matching sends

I put the i-receives before the sends, right?

```
for (i = 0; i < dimensions; ++i) {
  MPI_Irecv;
  MPI_Isend;
}
MPI_Waitall;
```

Well, not exactly…

You need to do it across ranks, like this:

```
for (i = 0; i < dimensions; ++i) {
  MPI_Irecv;
}
// Do some useful compute here
// Don't do MPI_Barrier here
for (i = 0; i < dimensions; ++i) {
  // Okay place for some compute†
  MPI_Isend;
  // Better place for some compute†
}
// Best place to overlap compute†
MPI_Waitall;
```

†Note: MPI buffers passed into `MPI_Isend`, etc. are owned by MPI until *after* the `MPI_Wait`, etc. Don't touch! That includes reading!

How to give MPI a chance to make progress?

Cassini NICs do **both** tag-matching and progress the rendezvous protocol in hardware!

However, if profiling shows a non-trivial amount of time in `MPI_Wait`, etc., try one of these two methods:

Make calls into MPI every so often

- During "computation overlap" code, especially if there are pending non-blocking collectives
- `MPI_Testsome` is preferred over `MPI_Testany`
- Many MPI calls will guarantee MPI progress
  - `MPI_Test` (and all its variants)
  - `MPI_Wait` (and all its variants)
  - `MPI_Request_get_status` might be a better alternative in MPI 4.1, officially†

†Note: The MPI Forum passed initial vote for it having same text as `MPI_Test` with regards to "progress guarantees".

Set `MPICH_ASYNC_PROGRESS=1`

- This will spawn an MPI progress thread
- That thread will need a CPU core‡
- Forces thread-safety to **MPI_THREAD_MULTIPLE**
- Might cause some MPI performance overhead

Avoid this if you can. It is rarely actually needed. With Cassini NICs, the only common case that would need this is when using non-blocking collectives and your code doesn't have MPI calls for a long time.

‡Note: It is worth a try if you have spare cores

HPE Cray MPI Documentation

- `man intro_mpi`
  - Most useful MPI environment variables are documented here
  - Pointers to other useful man pages
- `man fi_cxi` (The CXI Fabric Provider for libfabrics)
  - Cassini (CXI) is the name of the NIC in the Slingshot-11 network
  - This man page documents environment variables for libfabrics (FI) that are specific to Cassini NICs
- `module show cray-mpich`
  - Change log and bugs fixed by the currently loaded version
- OLCF System User Guide for Frontier
  - https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#gpu-aware-mpi (scroll up a tiny bit to start)
  - MPI information is sprinkled throughout the examples

Please *ignore* RXM on Sligshot-11 systems

Building/Compiling with HPE Cray MPI

- Load needed modules
  - `module load cray-mpich` (you might want to try a newer than the default version)
  - `module load craype-accel-amd-gfx90a` (not strictly necessary, but you might want it)
- Tell your build system how to link to the MPI, in two primary ways
  - Use the HPE Cray Wrapper Compilers: `cc/CC/ftn` with either of these two PrgEnv modules:
    `-module load PrgEnv-cray` and if you want a non-default ROCm: `module load amd-mixed/X.Y.Z`
    `-module load PrgEnv-amd` and if you want a non-default ROCm: `module swap amd amd/X.Y.Z`
  - Or specify these compiler and linker flags for your non-wrapped compiler of choice
    - Compiler flags: `-I${MPICH_DIR}/include`
    - Linker flags: `-L${MPICH_DIR}/lib -lmpi`
    - Extra Linker flags to handle GPU resident message buffers:

      `${PE_MPICH_GTL_DIR_amd_gfx90a} ${PE_MPICH_GTL_LIBS_amd_gfx90a}`

HPE Cray MPI: Commonly Used Environment Variables

- `MPICH_ENV_DISPLAY=1`
  - Set this so your program will log exactly how MPI was configured
- `MPICH_VERSION_DISPLAY=1`
  - Helps to track how issues change with different MPI versions

- `MPICH_GPU_SUPPORT_ENABLED=1`
  - Allows GPU resident buffers to be used in MPI calls. Probably already using this one
- `MPICH_GPU_MANAGED_MEMORY_SUPPORT_ENABLED=1`
  - Allows MPI to use memory addresses that can be migrated (i.e. cuda/hipMallocManaged memory)
- `MPICH_OFI_NIC_POLICY=GPU`
  - Selects the NIC that is closest to the GPU (GCD) being used†
  - Use `MPICH_OFI_NIC_POLICY=NUMA` for non-GPU codes (also usually works for GPU codes)

†NOTE: For best performance, each Frontier node needs at least 4 MPI ranks (1 per NIC), usually 8 MPI ranks (1 per GCD). If you have fewer than 4 MPI ranks per node, not all the NICs will be used. HPE Cray MPI does not currently (and isn't planning on) load balancing a single MPI rank's traffic across multiple Cassini NICs.

HPE Cray MPI: Performance/Tuning Environment Variables

Generally, the defaults are good, but sometimes you can tune things for your application

- `FI_MR_CACHE_MONITOR=memhooks`
  - This is the first thing I try for unexplained performance issues or MPI problems (hangs or significant performance loss at scale). It forces a different memory registration cache monitor that tends to work better, in my experience.
- `FI_CXI_RX_MATCH_MODE=software`
  - Use this for flow control or queue exhaustion problems. Seeing a lot of these at scale
- If using a version *older* than `cray-mpich/8.1.23` you might try increasing these parameters:
  - `FI_CXI_REQ_BUF_SIZE`, `FI_CXI_REQ_BUF_MIN_POSTED`, `FI_CXI_DEFAULT_CQ_SIZE`
  - These variables change the amount of buffer space available in software matching mode.
  - Increasing these can OOM the job, so try to be judicious.
- `MPICH_GPU_IPC_THRESHOLD=<value>`
  - This is a performance tunable parameter for controlling when GPU DMA engines are used for intra-node GPU-GPU transfers. Messages smaller than this threshold won't spend time setting up the DMA engine. If your code does *not* reuse GPU-GPU buffers, it might be worth increasing this threshold from the 1024-byte default. However, if your code *does* reuse GPU-GPU buffers, the setup time can be amortized. Thus, try to reuse buffers.

HPE Cray MPI: Debugging† Environment Variables

- `MPICH_SINGLE_HOST_ENABLED=0`
  - Force traffic through the NIC when you only have one node
- `MPICH_SMP_SINGLE_COPY_MODE=NONE`
  - This disables all intranode optimizations and forces all in-node communication through slow interprocess communicator paths on CPU memory. This is very useful to turn on if you think there's a bug in a fast on-node communication pathway.
- `MPICH_GPU_IPC_ENABLED=0`
  - Disabling SMP operations above will also disable IPC. To keep CPU side intranode optimizations in place but force GPU intranode through CPU staging buffers, turn on GPU IPC.

†NOTE: Don’t use these in production runs! They are for debugging purposes only.

Low Level Fabric Interface (FI) Debugging/Logging Environment Variables

- `FI_LOG_LEVEL=warn` & `FI_LOG_PROV=cxi`
  - This will cause the Cassini provider to log messages. Can be very useful to debug issues.
  - Turn up to debug (`FI_LOG_LEVEL=debug`) and you'll get a firehose, so be careful.
  - When using, ignore these messages, they are “normal” on Frontier:

```
libfabric:8012:1658440432:core:core:cuda_hmem_dl_init():287<warn> Failed to find cudaMemcpy
libfabric:8012:1658440432:core:core:ofi_hmem_init():227<warn> Failed to initialize hmem iface FI_HMEM_CUDA: No data available
libfabric:13281:1658440433:core:core:fi_getinfo_():1072<warn> fi_getinfo: provider ofi_rxm returned -61 (No data available)
libfabric:13281:1658440433:core:core:fi_getinfo_():1072<warn> fi_getinfo: provider ofi_rxd returned -61 (No data available)
libfabric:4110:1659388214:core:core:ofi_shm_map():153<warn> shm_open failed
```

HPE Cray MPI Takeaways

- Profile your application at relevant scale & problem size
- If you see or suspect MPI performance problems
  - Make sure you followed the MPI Best Practices I mentioned†
  - See if any of the environment variables I mentioned are relevant
  - Look for other relevant environment variables in `man intro_mpi`
  - Re-profile to see if the problem has changed/improved/moved
  - See if there is a newer cray-mpich/X.Y.Z version available
  - Ask at the Weekly Crusher (Frontier) COE Office Hours (currently held on Mondays from 2-3pm EST): https://www.olcf.ornl.gov/crusher-office-hours

†My copy of the MPI-3 specification from 2012 is 822 pages long. This talk only scratched the surface of what could be recommended.

Thank you

Tim Mattox
HPC Performance Engineer, HPE Frontier Center of Excellence (COE)
timothy.mattox@hpe.com

MPI Best Practices, Explanations

Why post non-blocking receives before their matching sends?

- If the MPI_Irecv is already posted when a message arrives:
  - The payload can go directly into the destination buffer without needing to make a temporary copy
  - Even for larger messages using the rendezvous protocol, the RDMA-read of the payload can start immediately
- Otherwise, the message is put into an “Unexpected Message” queue
  - For small messages using the eager protocol:
    - The payload is copied into a temporary buffer
    - The payload is copied again when the matching receive is posted
  - For larger messages using the rendezvous protocol:
    - The (bulk of the) payload waits at the sender until the matching receive is posted
    - This waiting delays getting those bytes onto the wire (effectively averaging in “zero” bandwidth during this delay)
  - A non-empty “Unexpected Queue” must be searched for a match any time an MPI receive call is made.

MPI Best Practices, Explanations

Why check the MPI error/return codes? MPI_ERRORS_ARE_FATAL is the default setting, right?

- Yes, MPI_ERRORS_ARE_FATAL is the default setting, but you might prefer something else
- Things can go wrong (and will someday, especially on new leading-edge systems such as Frontier)
- You can get better debug information from your application by switching to MPI_ERRORS_RETURN
  - Backtraces don't always tell you enough about why or where things died.
  - Having your application stop and print a non-generic error message can greatly help debug things

MPI Best Practices, Explanations

Why not abuse the MPI layer? It seems like magic to me!

- Only use a reasonable number of MPI Communicators
  - Many communicator creation calls are non-local, causing book-keeping network traffic
  - Each communicator has more than just a “set membership” data structure (e.g. group) to track
    - See Chapters 6 and 7 of the MPI Standard for all the things communicators can keep track of
- Most MPI implementations reserve some values of the 32-bit tag field for internal use.
  - Slingshot-11 supports 33,554,431 as the maximum tag value.
  - Other underlying interconnects that HPE Cray MPI supports have different maximum tag values
  - Use `MPI_Comm_get_attr(comm, MPI_TAG_UB, &val, &flag)` to find the upper bound
- Only post a reasonable number of pending receives/sends
  - When a message arrives, the the pending-receives data structure must be searched for the correct match
    - Singshot-11 hardware matching can only track a limited number before having to fall back to software matching
    - The search time grows relative to the number of pending receives, especially when using software matching
  - The Slingshot-11 NIC can only directly track/progress a limited number of outstanding sends
  - The MPI layer may not know as well as the application which messages can progress efficiently

MPI Best Practices, Explanations

Why avoid MPI_ANY_SOURCE? Isn't that easier to match?

- Matching with MPI_ANY_SOURCE is harder
- Specifying a specific source rank allows for O(~1) matching using a hash of the message envelope
- The receive with MPI_ANY_SOURCE can't be matched using that O(~1) hash function
  - Instead, a message must first fail to match with the hash
  - and then check for a match in the queue of receives that were marked as MPI_ANY_SOURCE
- Thus, it is best to avoid *unnecessary* use of MPI_ANY_SOURCE
  - But use it if you need to, rather than posting a bunch of "just in case" receives.

What about slurm options that affect MPI?

- The Frontier and Crusher systems are running in Low Noise Mode (LNM)
  - The Linux OS is generally restricted to run its tasks on core 0
  - Similarly, interrupts are mapped to the 1st core of each CCD, leaving 7 per CCD that should be less noisy
  - In testing we discovered that sometimes GCD helper tasks get scheduled on the 2nd core of a CCD
  - Avoid all these potentially noisy cores with the –S16 option to slurm `sbatch` or `salloc` commands
    - This lowers the available hardware cores to just 48 out of 64 per node, so this is not the best choice for CPU OpenMP
- Some example srun options (I also add `--cpu-bind=`*something* as well, details are for another talk):
  - For `--core-spec=16` (a.k.a `-S16`) allocations (my personal recommendation):

    ```
    srun --cpus-per-gpu=6 --gpus-per-task=1 --gpu-bind=closest ...
    ```
  - For `--core-spec=8` (a.k.a `-S8`) allocations (the slurm default on Frontier & Crusher):

    ```
    srun --cpus-per-gpu=7 --gpus-per-task=1 --gpu-bind=closest ...
    ```
  - For `--core-spec=0` (a.k.a `-S0`) allocations:

    ```
    srun --cpus-per-gpu=8 --gpus-per-task=1 --gpu-bind=closest ...
    ```
- Verify the mapping is as you wanted using something like the hello_jobstep program:
  - https://code.ornl.gov/olcf/hello_jobstep