Optimizing MPI Performance on Theta

ALCF Computational Performance Workshop – May 5th 2020

Sudheer Chunduri
sudheer@anl.gov

www.anl.gov

Acknowledgements
Krishna Kandalla, Cray
Outline

- Cray XC network software stack and MPI software stack
- Non-blocking collectives
- Topology mapping optimizations
- Few key performance tuning knobs
- MPI+X optimizations
- MPI support for MCDRAM on KNL
- Cray XC routing optimizations
- Noise mitigation at the node level
Cray XC Network Software Stack

**uGNI** - Generic Network Interface  
(message passing based)

**DMAPP** - Distributed Shared Memory  
Application APIs (shared memory)

**uGNI** and **DMAPP** provide low-level communication services to user-space software.
Cray MPI Software Stack (CH3 device)
MPI-3 Nonblocking Collectives

- Enables **overlap of communication/computation** similar to nonblocking (send/recv) communication
- Non-blocking variants of all collectives: MPI_Ibcast (\(<\text{bcast args}\), MPI_Request *req);

**Semantics**
- Function returns no matter what
- Usual completion calls (wait, test)
- Out-of-order completion

**Semantic advantages**
- Enables asynchronous progression (software pipelining)
- Decouple data transfer and synchronization (Noise Resiliency)
- Allow overlapping communicators
- Multiple outstanding operations at any time

```c
MPI_Comm comm;
int array1[100], array2[100];
int root=0;
MPI_Request req;
...
MPI_Ibcast(array1, 100, MPI_INT, root, comm, &req);
compute(array2, 100);
MPI_Wait(&req, MPI_STATUS_IGNORE);
```
MPI-3 Nonblocking Collectives Support

- Includes many optimizations for MPI-3 nonblocking Collectives
- Not ON by default. User must set the following env. Variables:
  ```
  export MPICH_NEMESIS_ASYNC_PROGRESS=[SC|MC|ML] (network interface DMA engine enables asynchronous progress)
  export MPICH_MAX_THREAD_SAFETY=multiple
  ```
- Special optimizations for Small message MPI_Iallreduce, based on Aries HW Collective Engine:
  Users must link against DMAPP
  ```
  -Wl,--whole-archive,-ldmapp,--no-whole-archive (static linking)
  -ldmapp (dynamic linking)
  ```
  ```
  export MPICH_NEMESIS_ASYNC_PROGRESS=[SC|MC|ML]
  export MPICH_MAX_THREAD_SAFETY=multiple
  export MPICH_USE_DMAPP_COLL=1
  ```
Topology Mapping and Rank Reordering

- **Topology mapping**
  - Minimize communication costs through interconnect topology aware *task mapping*
  - Could *potentially* help reduce congestion
  - Node placement for the job could be a factor (no explicit control available to request a specific placement)

- **Application communication pattern**
  - MPI process topologies expose this in a portable way
  - Network topology agnostic

- **Rank reordering**
  - Can override the default mapping scheme
  - The default policy for *aprun* launcher is SMP-style placement
  - To display the MPI rank placement information,
    - set `MPICH_RANK_REORDER_DISPLAY`.
MPI Rank Reordering

- **MPICH_RANK_REORDER_METHOD**
  - Vary rank placement to optimize communication (ex: maximize on-node communication between MPI ranks)
  - Use CrayPat with “-g mpi” to produce a specific `MPICH_RANK_ORDER` file to maximize intra-node communication
  - Or, use perf_tools `grid_order` command with your application's grid dimensions to layout MPI ranks in alignment with data grid

- To use:
  - name your custom rank order file: `MPICH_RANK_ORDER`
  - This approach is physical system topology agnostic
    
    ```
    export MPICH_RANK_REORDER_METHOD=3
    ```
MPI Rank Reordering

- **MPI_RANK_REORDER_METHOD (cont.)**
  - A topology and placement aware reordering method is also available
  - Optimizes rank ordering for Cartesian decompositions using the layout of nodes in the job
  - To use:
    - export `MPI_RANK_REORDER_METHOD=4`
    - export `MPI_RANK_REORDER_OPTS="-ndims=3 -dims=16,16,8"`

**MPI Grid Detection:**

There appears to be point-to-point MPI communication in a 96 X 8 grid pattern. The 52% of the total execution time spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node. The effect of several rank orders is estimated below.

A file named `MPICH_RANK_ORDER.Grid` was generated along with this report and contains usage instructions and the Custom rank order from the following table.

<table>
<thead>
<tr>
<th>Rank Order</th>
<th>On-Node Bytes/PE</th>
<th>Bytes/PE% of Total</th>
<th>MPI_RANK_REORDER_METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>2.385e+09</td>
<td>95.55%</td>
<td>3</td>
</tr>
<tr>
<td>SMP</td>
<td>1.880e+09</td>
<td>75.30%</td>
<td>1</td>
</tr>
<tr>
<td>Fold</td>
<td>1.373e+06</td>
<td>0.06%</td>
<td>2</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>0.000e+00</td>
<td>0.00%</td>
<td>0</td>
</tr>
</tbody>
</table>
Profiling with CrayPat

- Application built with "pat_build -g mpi"
- pat_report generates the CrayPat report
- Note the MPI call times and number of calls
- Load imbalance across the ranks

Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>667.935156</td>
<td>--</td>
<td>--</td>
<td>49,955,946.2</td>
<td>Total</td>
</tr>
<tr>
<td>40.0%</td>
<td>267.180169</td>
<td>--</td>
<td>--</td>
<td>49,798,359.2</td>
<td>MPI</td>
</tr>
<tr>
<td>24.0%</td>
<td>160.400193</td>
<td>28.907525</td>
<td>15.3%</td>
<td>2,606,756.0</td>
<td>MPI_Wait</td>
</tr>
<tr>
<td>6.4%</td>
<td>42.897564</td>
<td>0.526996</td>
<td>1.2%</td>
<td>157,477.0</td>
<td>MPI_Allreduce</td>
</tr>
<tr>
<td>4.8%</td>
<td>31.749303</td>
<td>3.923541</td>
<td>11.0%</td>
<td>42,853,974.0</td>
<td>MPI_Comm_rank</td>
</tr>
<tr>
<td>3.5%</td>
<td>23.303805</td>
<td>1.774076</td>
<td>7.1%</td>
<td>1,303,378.0</td>
<td>MPI_Isend</td>
</tr>
<tr>
<td>1.1%</td>
<td>7.658009</td>
<td>0.637044</td>
<td>7.7%</td>
<td>1,303,378.0</td>
<td>MPI_Irecv</td>
</tr>
<tr>
<td>39.1%</td>
<td>260.882504</td>
<td>--</td>
<td>--</td>
<td>2.0</td>
<td>USER</td>
</tr>
<tr>
<td>39.1%</td>
<td>260.882424</td>
<td>17.270557</td>
<td>6.2%</td>
<td>1.0</td>
<td>main</td>
</tr>
<tr>
<td>20.9%</td>
<td>139.872482</td>
<td>--</td>
<td>--</td>
<td>157,585.0</td>
<td>MPI_SYNC</td>
</tr>
<tr>
<td>20.4%</td>
<td>136.485384</td>
<td>36.223589</td>
<td>26.5%</td>
<td>157,477.0</td>
<td>MPI_Allreduce(sync)</td>
</tr>
</tbody>
</table>
### Profiling with CrayPat

MPI message sizes are reported

The message size distributions can help characterize an application as

- Latency sensitive
- Bandwidth sensitive

---

<table>
<thead>
<tr>
<th>Category</th>
<th>Total</th>
<th>MPI Msg Bytes</th>
<th>100.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MPI Msg Bytes</td>
<td>18,052,938,280.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI Msg Count</td>
<td>1,460,959.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MsgSz &lt;16 Count</td>
<td>157,529.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16&lt;= MsgSz &lt;256 Count</td>
<td>65.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256&lt;= MsgSz &lt;4KiB Count</td>
<td>2,815.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4KiB&lt;= MsgSz &lt;64KiB Count</td>
<td>1,300,511.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64KiB&lt;= MsgSz &lt;1MiB Count</td>
<td>39.0 msgs</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Category</th>
<th>Total</th>
<th>MPI Msg Bytes</th>
<th>100.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MPI Msg Bytes</td>
<td>18,051,670,432.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI Msg Count</td>
<td>1,303,378.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MsgSz &lt;16 Count</td>
<td>16.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16&lt;= MsgSz &lt;256 Count</td>
<td>0.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256&lt;= MsgSz &lt;4KiB Count</td>
<td>2,812.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4KiB&lt;= MsgSz &lt;64KiB Count</td>
<td>1,300,511.0 msgs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64KiB&lt;= MsgSz &lt;1MiB Count</td>
<td>39.0 msgs</td>
</tr>
</tbody>
</table>
Hugepages to Optimize MPI

- Use HUGEPAGES
  - Linking and running with hugepages can offer a significant performance improvement for many MPI communication sequences
    - including MPI collectives and basic `MPI_Send/MPI_Recv` calls
  - Most important for applications calling `MPI_Alltoall[v]` or performing point to point operations with a similarly well-connected pattern and large data footprint

- To use HUGEPAGES:
  - `module load craype-hugepages8M` (many sizes supported)
  - `<< re-link your app >>`
  - `module load craype-hugepages8M`
  - `<< run your app >>`
Key Environment Variables for XC

- **Use** `MPICH_USE_DMAPP_COLL` for hardware supported collectives
  - Most of MPI's optimizations are enabled by default, but not the DMAPP-optimized features, because...
  - Using DMAPP may have some disadvantages
    - May reduce resources MPICH has available (share with DMAPP)
    - Requires more memory (DMAPP internals)
    - DMAPP does not handle transient network errors
  - These are highly-optimized algorithms which may result in significant performance gains, but user has to request them

- Supported DMAPP-optimized functions:
  - `MPI_Allreduce` (4-8 bytes)
  - `MPI_Bcast` (4 or 8 bytes)
  - `MPI_Barrier`

- **To use** (link with `libdmapp`):
  - Collective use: `export MPICH_USE_DMAPP_COLL=1`
Key Environment Variables for XC

- **MPICH GNI environment variables**
  - To optimize inter-node traffic using the Aries interconnect, the following are the most significant env variables to play with *(avoid significant deviations from the default if possible)*:
    - `MPICH_GNI_MAX_VSHORT_MSG_SIZE`
      - Controls max message size for E0 mailbox path (Default: varies)
    - `MPICH_GNI_MAX_EAGER_MSG_SIZE`
      - Controls max message size for E1 Eager Path (Default: 8K bytes)
    - `MPICH_GNI_NUM_BUFS`
      - Controls number of 32KB internal buffers for E1 path (Default: 64)
    - `MPICH_GNI_NDREG_MAXSIZE`
      - Controls max message size for R0 Rendezvous Path (Default: 4MB)
    - `MPICH_GNI_RDMA_THRESHOLD`
      - Controls threshold for switching to BTE from FMA (Default: 1K bytes)

- Refer the MPI man page for further details
Key Environment Variables for XC

- Specific Collective Algorithm Tuning
  - Different algorithms may be used for different message sizes in collectives (e.g.)
    - Algorithm A might be used for Alltoall for messages < 1K.
    - Algorithm B might be used for messages >= 1K.
  - To optimize a collective, you can modify the cutoff points when different algorithms are used. This may improve performance. A few important ones are:
    - MPICH_ALLGATHER_VSHORT_MSG
    - MPICH_ALLGATHERV_VSHORT_MSG
    - MPICH_GATHERV_SHORT_MSG
    - MPICH_SCATTERV_SHORT_MSG
    - MPICH_GNI_A2A_BLK_SIZE
    - MPICH_GNI_A2A_BTE_THRESHOLD
  - Refer the MPI man page for further details
MPI+X Hybrid Programming Optimizations

- MPI Thread Multiple Support for
  - Point to point operations & Collectives (optimized global lock)
  - MPI-RMA (thread hot)

- All supported in default library
  (Non-default Fine-Grained Multi-Threading library is no longer needed)

- Users must set the following env. variable:
  - `export MPICH_MAX_THREAD_SAFETY=multiple`

- Global lock optimization ON by default (N/A for MPI-RMA)
  - `export MPICH_OPT_THREAD_SYNC=0` falls back to `pthread_mutex()`

- “Thread hot” optimizations for MPI-3 RMA:
  - Contention free progress and completion
  - High bandwidth and high message rate
  - Independent progress – thread(s) flush outstanding traffic, other threads make uninterrupted progress
Cray MPI support for MCDRAM on KNL

- Cray MPI offers allocation + hugepage support for MCDRAM on KNL
  - Must use: MPI_Alloc_mem() or MPI_Win_Allocate()
  - Dependencies: memkind, NUMA libraries and dynamic linking. module load cray-memkind

- Feature controlled with env variables
  - Users select: Affinity, Policy and PageSize
  - MPICH_ALLOC_MEM_AFFINITY = DDR or MCDRAM
    - DDR = allocate memory on DDR (default)
    - MCDRAM = allocate memory on MCDRAM
  - MPICH_ALLOC_MEM_POLICY = M/ P/ I
    - M = Mandatory: fatal error if allocation fails
    - P = Preferred: fall back to using DDR memory (default)
    - I = Interleaved: Set memory affinity to interleave across MCDRAM NUMA nodes (For SNC* cases)
  - MPICH_ALLOC_MEM_PG_SZ
    - 4K, 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M, 512M (default 4K)
Cray XC Routing

- Aries provides three basic routing modes
  - Deterministic (minimal)
  - Hashed deterministic (minimal, non-minimal), hash on “address”
  - Adaptive
    - 0 – No bias (default)
    - 1 – Increasing bias towards minimal (as packet travels)
      - Used for MPI all-to-all
    - 2 – Straight minimal bias (non-increasing)
    - 3 – Strong minimal bias (non-increasing)

- Non-adaptive modes are more susceptible to congestion unless the traffic is very uniform and well-behaved

- `MPICH_GNI_ROUTING_MODE` environment variable
  - Set to one of `ADAPTIVE_[0123]`, `MIN_HASH`, `NMIN_HASH`, `IN_ORDER`

- `MPICH_GNI_A2A_ROUTING_MODE` also available

Cray XC group:
Minimal path: 2 hops
Non-minimal path: 4 hops

module load adaptive-routing-a3
module unload adaptive-routing-a3
module help adaptive-routing-a3
Core Specialization

- Offloads some kernel and MPI work to unused Hyper-Thread(s)
- Good for large jobs and latency sensitive MPI collectives
- Highest numbered unused thread on node is chosen
  - Usually the highest numbered HT on the highest numbered physical core
- Examples
  - `aprun -r 1 ...`
  - `aprun -r N ...` # use extra threads
- Cannot oversubscribe, OS will catch
  - Illegal: `aprun -r1 -n 256 -N 256 -j 4 a.out`
  - Legal: `aprun -r1 -n 255 -N 255 -j 4 a.out`
  - Legal: `aprun -r8 -n 248 -N 248 -j 4 a.out`

8-Byte Allreduce on Theta
64 processes per node
(run in production – other jobs are running)
Summary

- Optimizations were done in Cray MPI to improve pt2pt and collective latency on KNL
- Further tuning is possible through the environment variables
- Topology & routing-based optimizations, huge-page and hybrid programming optimizations could be explored
- MPI 3.0 nonblocking and neighborhood collectives are optimized
- Necessary to use -r1 (core spec) to reduce performance variability due to OS noise

References:

- MPI 3.1 Standard: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
- Cray MPI for KNL: https://www.alcf.anl.gov/files/Chunduri_MPI_Theta.pdf (May 18 workshop - slightly basic version than this talk)
- MPI benchmarking on Theta: https://cug.org/proceedings/cug2018_proceedings/includes/files/pap131s2-file1.pdf
- Advanced MPI Programming Tutorial at SC17, November 2017 (https://www.mcs.anl.gov/~thakur/sc17-mpi-tutorial/)
- Run-to-run Variability: https://dl.acm.org/citation.cfm?id=3126908.3126926
- LDMS: https://github.com/ovis-hpc/ovis/tree/master/ldms
Hands-on Session

- Few sample codes are provided here at https://xgitlab.cels.anl.gov/alcf/training/tree/mpi/ProgrammingModels/MPI/Theta
- These are also accessible at /projects/Comp_Perf_Workshop/examples/training/ProgrammingModels/MPI/Theta

- The nonblocking_coll and nonblocking_p2p have sample run scripts to submit jobs
- Feel free to experiment with using the environment variables with your own application codes
- Some examples to try out
  - Potential performance optimization (Huge pages, Routing mode change, Hardware collective offload, Core specialization etc.)
  - Functionality specific (nonblocking collectives, MPI+X etc.)