Using MPI Effectively on Theta

SDL Workshop – Oct 3rd 2018

Sudheer Chunduri
sudheer@anl.gov

www.anl.gov

Acknowledgements
Krishna Kandalla, Cray
Outline

- Cray XC40 (Theta) Network Software Stack
- Introduction to MPI on Theta (Cray MPICH)
  - MPI 3.0 feature support in Cray MPICH
- MPI Tuning Parameters
  - KNL specific
  - Network specific
- Recommendations for optimizing MPI performance
Cray XC Network Software Stack

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

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

**uGNI** and **DMAPP** provide low-level communication services to user-space software
Brief Introduction to Cray MPICH

- Cray MPI compliant with MPI 3.1
  - Cray MPI uses MPICH3 distribution from Argonne
  - Merge to ANL MPICH 3.2 – latest release MPT 7.7.1

- I/O, collectives, P2P, and one-sided all optimized for XC architecture
  - SMP aware collectives
  - High performance single-copy on-node communication via xpmem (not necessary to program for shared memory)
  - HW collectives to optimize small message collectives at scale (MPI-3)
  - Non-Blocking Collectives (MPI-3)
  - Highly optimized “Thread-Hot” MPI-3 one-sided (RMA) communication (MPI-3)
  - Dynamic Process Management Support (MPI-3)
  - Optimized and Tuned MPI I/O

- Highly tunable through environment variables
  - Defaults should generally be best, but some cases benefit from fine tuning

- Integrated within the Cray Programming Environment
  - Compiler drivers manage compile flags and linking automatically
  - Profiling through Cray Perftools
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 ( <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

- Cray MPT 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 Neighborhood Collectives

- **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`. 
Rank Reordering

- **MPICH_RANK_REORDER_METHOD**
  - Vary rank placement to optimize communication (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
    ```
Rank Reordering

- **MPICH_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 MPICH_RANK_REORDER_METHOD=4`
    - `export MPICH_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>On-Node Bytes/PE%</th>
<th>MPICH_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, calls
- Load imbalance across the ranks

---

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

- MPI message sizes are reported

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

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

- Convenience functions (MPI-1)
  - Create a graph and query it, nothing else
  - Useful especially for Cartesian topologies
  - Query neighbors in n-dimensional space
  - Graph topology: each rank specifies full graph

- Scalable Graph topology (MPI-2.2)
  - Graph topology: each rank specifies its neighbors or arbitrary subset of the graph

- Neighborhood collectives (MPI-3.0)
  - Adding communication functions defined on graph topologies (neighborhood of distance one)
Neighborhood Collectives

- New functions MPI_Neighbor_allgather, MPI_Neighbor_alltoall, and their variants define collective operations among a process and its neighbors
  - Allgather: One item to all neighbors
  - Alltoall: Personalized item to each neighbor

- Neighborhood collectives add communication functions to process topologies
  - Neighbors are defined by an MPI Cartesian or graph virtual process topology that must be previously set

- There functions are useful, for example, in stencil computations that require nearest-neighbor Exchanges
- Enables “Build your own collective” functionality in MPI
  - Neighborhood collectives are a simplified version – data types for communication patterns
- They also represent sparse all-to-many communication concisely, which is essential when running on many thousands of processes
  - Do not require passing long vector arguments as in MPI_Alltoallv
Hugepages to Optimize MPI

- Use HUGEPAGES
  - While this is not an MPI env variable, 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)

- See 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

- See 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
  - Locks needed (within the MPI library) only if the number of threads exceed the number of network resources
  - Dynamic mapping between threads and network resources
    - Helps mitigate load imbalance and skew between threads
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 MPI support for MCDRAM on KNL (use cases)

- When the entire data set fits within MCDRAM, on KNL nodes in flat mode:
  ```
  aprun -Nx -ny numactl --membind=1 ./a.out
  ```
  - Easiest way to utilize hugepages on MCDRAM
  - craype-hugepage module is honored.
  - Allocations (malloc, memalign) on MCDRAM will be backed by hugepages
  - However, all memory allocated on MCDRAM (including MPI’s internal memory)
  - Memory available per node limited to % of MCDRAM configured as FLAT memory

- MPICH_INTERNAL_MEM_AFFINITY=DDR
  - forces shared-memory and mail-box memory (internal memory regions allocated by the MPI library) to DDR
  - Alternate solutions needed to utilize hugepage memory on MCDRAM, when the data set per node exceeds 16G
  - Necessary to identify performance critical buffers
  - Replace memory allocation calls with MPI_Alloc_mem() or MPI_Win_allocate()
  - Use Cray MPI env. vars to control page size, memory policy and memory affinity for allocations
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)
      o 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
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 several 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`
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:
- Cray XC series Network: [https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf](https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf)
- MPI 3.1 Standard: [https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf](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](https://www.alcf.anl.gov/files/Chunduri_MPI_Theta.pdf) (May 18 workshop - slightly basic version than this talk)
- MPI benchmarking on Theta: [https://cuq.org/proceedings/cuq2018_proceedings/includes/files/pap131s2-file1.pdf](https://cuq.org/proceedings/cuq2018_proceedings/includes/files/pap131s2-file1.pdf)
- Run-to-run Variability: [https://dl.acm.org/citation.cfm?id=3126908.3126926](https://dl.acm.org/citation.cfm?id=3126908.3126926)