Cray MPI for KNL

Gene Wagenbreth
Performance Engineer
gwagenbret@cray.com

Cray MPI for KNL

Heidi Poxon

Sr. Principal Engineer

Cray Programming Environment

Agenda

- Introduction to Cray MPI
- Rank Re-Order
- Load Balance
- “Jitter”
- KNL Optimizations - Threading
- Specific KNL optimizations: DMAPP, MCDRAM
- Optimizations for Hybrid (MPI/OpenMP) applications

Introduction to Cray MPI

- **Fortran, C, C++**
- **Integrated within the Cray Programming Environment**
  - Compiler drivers manage compile flags and linking automatically
  - Profiling through Cray performance tools
- **I/O, collectives, P2P, and one-sided all optimized for Cray system architecture**
  - SMP-aware collectives
  - High performance single-copy on-node communication via xpmem (not necessary to program for shared memory)
- **Highly tunable through environment variables**
  - Defaults should generally be best, but some cases benefit from fine tuning

Cray MPI Documentation

- **Primary user resource for tuning and feature documentation is the man page**
  - man intro_mpi

  OR

  - man MPI

- **Standard function documentation available as well**
  - E.g., man mpi_isend

MPICH_RANK_REORDER_METHOD

- Keep transfer on node (grid nearest neighbor)
- Vary your rank placement to optimize communication
- Can be a quick, low-hassle way to improve performance
- Use CrayPAT to produce a specific MPICH_RANK_ORDER file to maximize intra-node communication
- Or, use grid_order utility 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
  - export MPICH_RANK_REORDER_METHOD=3
- R09 (new) for off node reordering

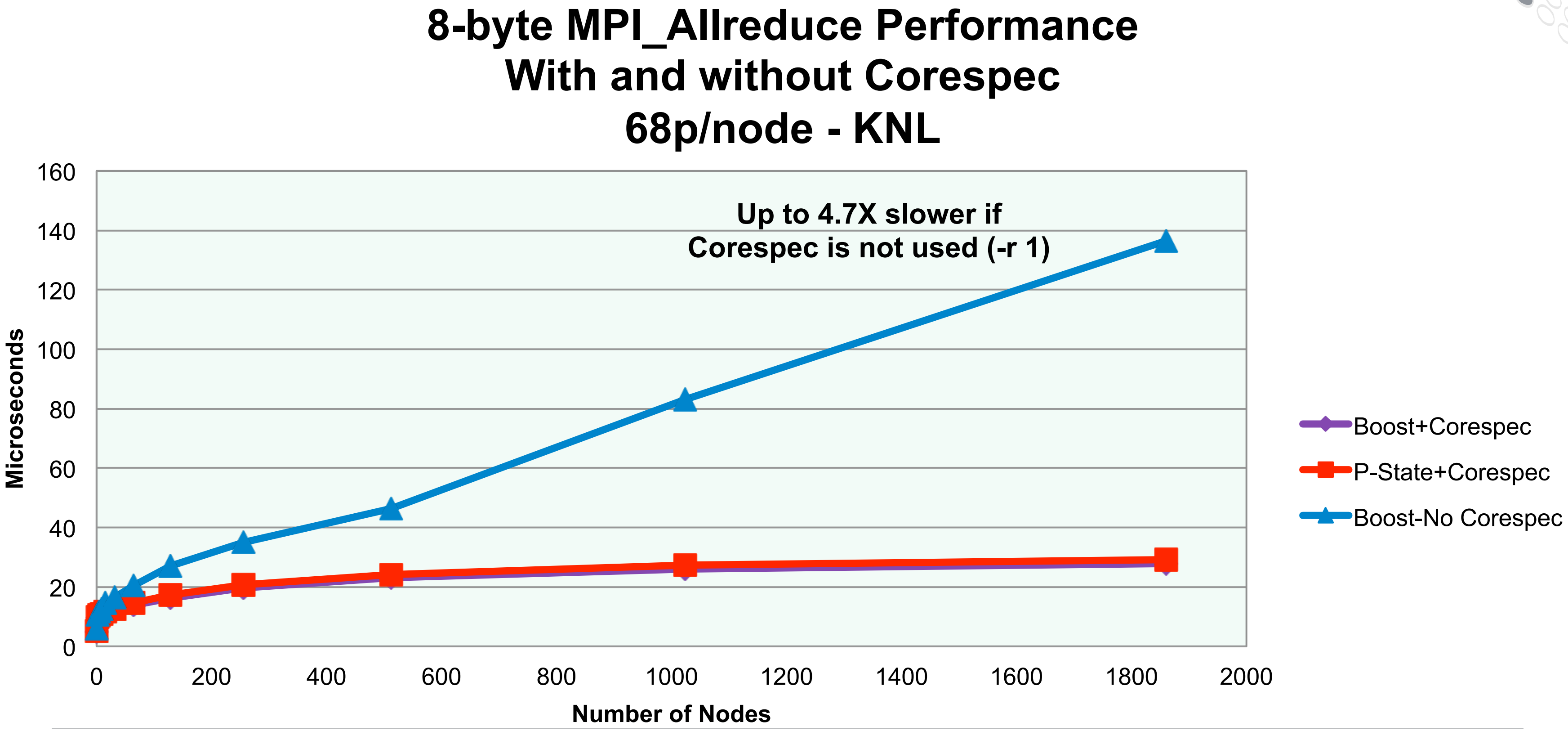
Load Balance

- Very important with large MPI task counts
- Time shows up in MPI Barriers
- Grid refinement example
- Difficult to diagnose
- Difficult to fix
- CrayPat or other tools

OS Noise Plays a Role – How Big?

- **Studied performance with and without corespec (-r 1)**
- **MPI Latency: Collectives, 1 rank per node**
  - 5.5X slower when not using corespec
- **MPI Latency: Collectives, 2 to 68 ranks per node**
  - 4.7X slower when not using corespec
- **MCDRAM direct mapped**

With and Without Corespec

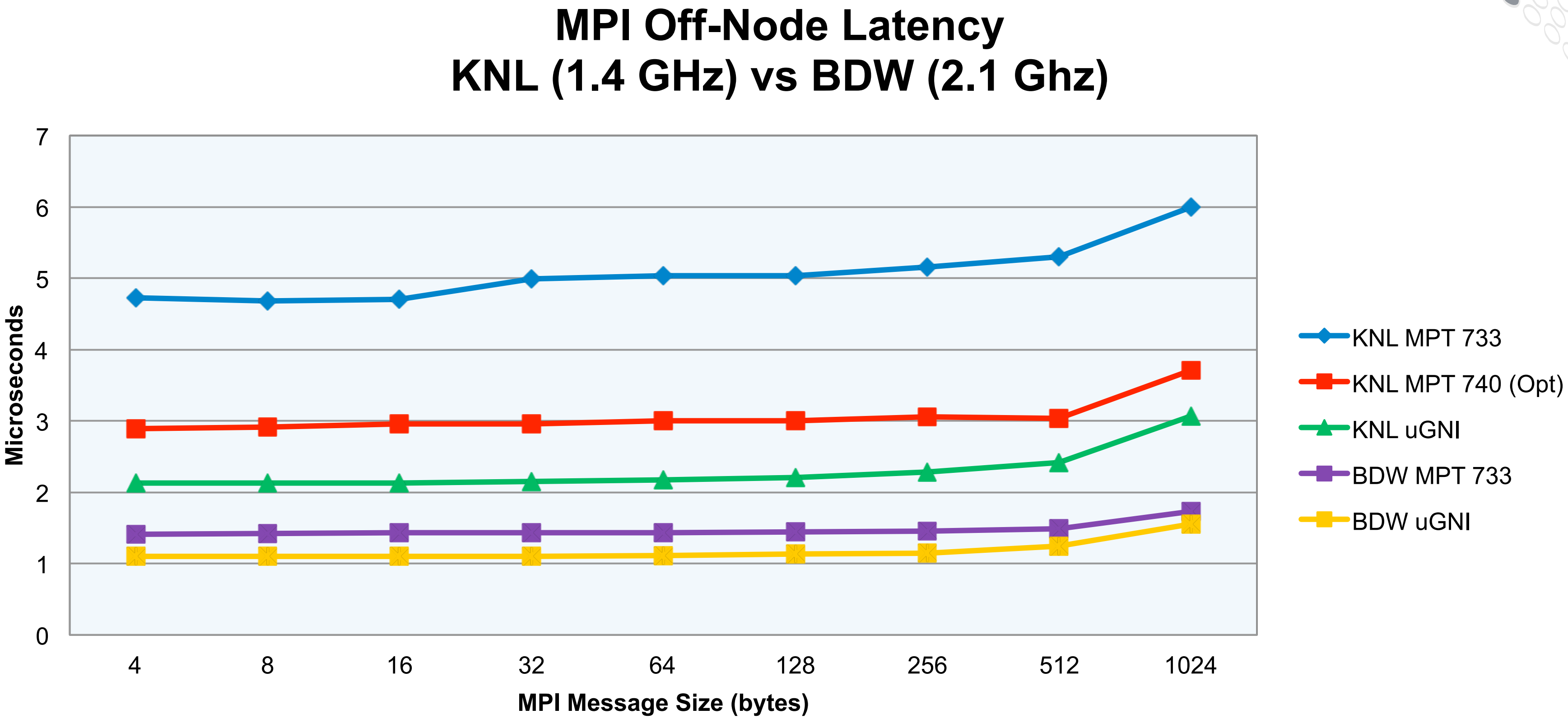


Latency studies on KNL with Cray MPI

- **MPI is typically all scalar code**
  - Lots of branches
  - Lots of small functions and function calls using pointers
  - With smaller Branch Target Buffer (BTB), KNL does not handle this type of scalar code as well as the Xeon processor (even if you adjust Xeon to slower KNL CPU frequency)

- **Optimizing the “critical path”**
  - Inline small functions
  - Use higher compiler optimization
  - Selective hand-tuning, which avoids taking branches in critical path
  - Disable FMA sharing when not needed
  - Use KNL-optimized memcpy

MPI Off-node Latency on KNL



COMPUTE | STORE | ANALYZE

Multi-threaded MPI Support and Optimizations

Thread Hot Communication in Cray MPI

- Design Objectives
  - Contention Free progress and completion
  - High bandwidth and high message rate
  - Independent progress – thread(s) flush outstanding traffic, other threads make uninterrupted progress
  - Dynamic mapping between threads and network resources
  - Locks needed only if the number of threads exceed the number of network resources
- MPI-3 RMA
  - Epoch calls (`Win_complete, Win_fence`) are thread-safe, but not intended to be thread hot
  - All other RMA calls (including request-based operations) are thread hot
  - Multiple threads doing Passive Synchronization operations likely to perform best
- MPI Pt2pt
  - `MPI_Send/MPI_Recv, MPI_Isend/MPI_Irecv, MPI_Wait/MPI_Waitall` will be thread hot.
  - Supports use cases where multiple threads issue Isend/Irecv ops, but master thread alone does Waitall
- MPI_Alltoall
  - Multiple threads can issue, progress and complete Alltoall operations concurrently. Each thread has a separate `MPI_Comm` handle.
  - The Allgather exchange (mem address, hndls) is protected by the big lock (room for optimization)

Multi-threading Optimizations in Cray MPI

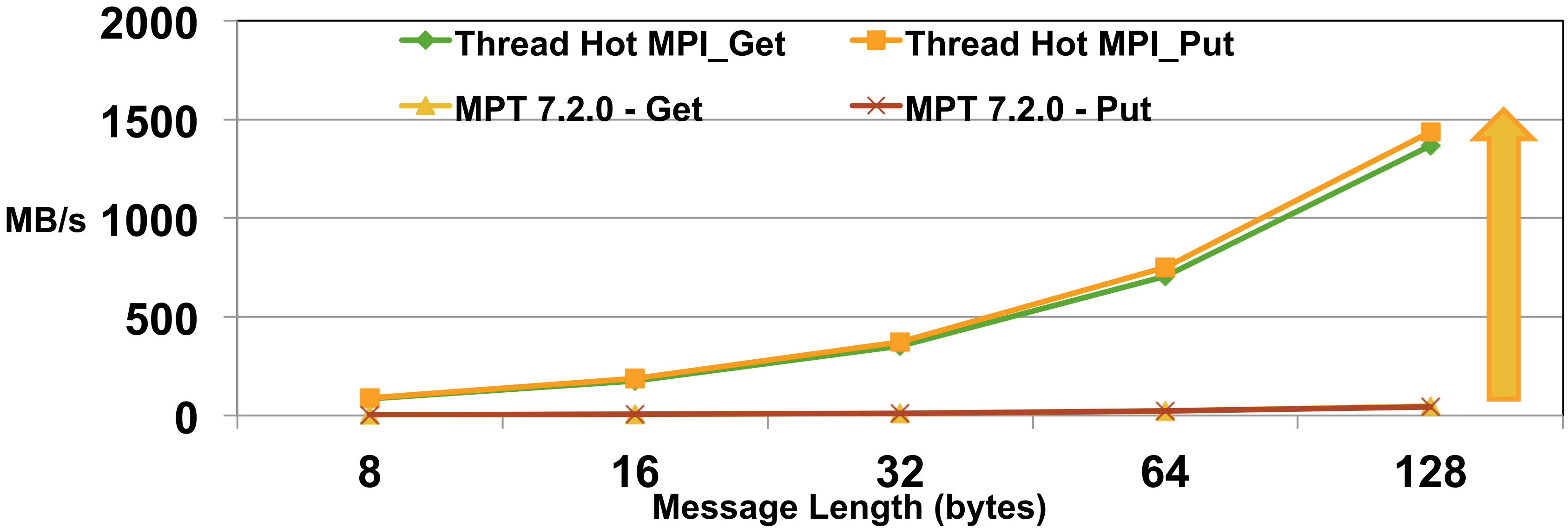
- **Easy way to hit the ground running on a KNL – MPI only mode**
  - Works quite well in our experience
  - Scaling to more than 2-8 threads most likely requires a different application design approach
  - MPI on hyperthreads may not perform well

- **"Bottom-Up" OpenMP development approach**

- **"Top-Down" SPMD model**
  - Increases the scope of code executed by OpenMP, allows for better load balancing and overall compute scaling on KNL
    - Allows multiple threads to call MPI concurrently
    - In this model, performance is limited by the level of support offered by MPI for multi-threaded communication
    - MPI implementations must offer "Thread-Hot" communication capabilities to improve communication performance for highly threaded use cases on KNL

MPI-3 RMA Communication Bandwidth

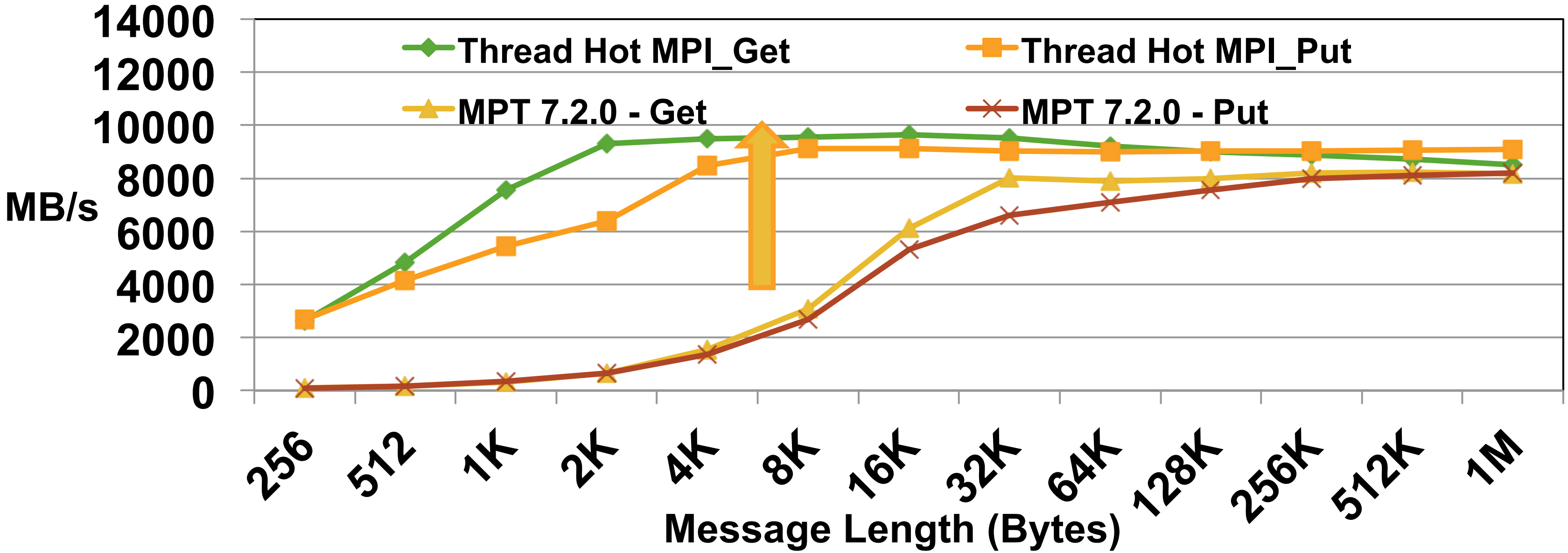
1 MPI process per node, 32 threads, Haswell, small messages



Thread Hot Cray MPI significantly outperforms the default (global-lock) implementation with the multi-threaded RMA benchmark for small payloads

MPI-3 RMA Communication Bandwidth

1 MPI process per node, 32 threads, Haswell, large messages



Thread Hot Cray MPI outperforms the default (global-lock) implementation with the multi-threaded RMA benchmark by about 4X for small and medium sized payloads

SPMD Top Down OpenMP – MPI Hybrid

- Typical OpenMP is done on a perloop basis
  - Hopefully outer loop of loop nest
- MPI is outside OpenMP regions
  - Single threaded
- Hybrid approach
  - Large OpenMP regions – maybe entire program
  - Multiple simultaneous MPI transfers inside threads
  - Works best with MPI-3 one sided transfers
  - Coarray Fortran or C
- Starting from scratch or total rewrite
- Peter Mendygral - Wombat

KNL Optimizations

Key Environment Variables and Tips for XC

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**

- **To use HUGEPAGES:**
  - `module load craype-hugepages8M` (many sizes supported)
  - << *compile your app* >>
  - `module load craype-hugepages8M`
  - << *run your app* >>

Using DMAPP

- **DMAPP optimizations not enabled by default because…**
  - Using DMAPP may have some disadvantages
    - May reduce resources MPICH has available (shared with DMAPP)
    - Requires more memory (for 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
  - MPI_Put / MPI_Get / MPI_Accumulate
- **To use, link with `libdmapp` and set the following environment variable**
  - Collective use: export MPICH_USE_DMAPP_COLL=1
  - RMA one-sided use: export MPICH_RMA_OVER_DMAPP=1

MPICH GNI Environment Variables

Used to optimize inter-node traffic using the Aries interconnect, the following are the most significant variables to try (*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

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, which may improve performance**

- **MPICH_ALLTOALL_SHORT_MSG**
- **MPICH_ALLGATHER_VSHORT_MSG**
- **MPICH_ALLGATHERV_VSHORT_MSG**
- **MPICH_GATHERV_SHORT_MSG**
- **MPICH_SCATTERV_SHORT_MSG**

See the MPI man page for further details

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
  - To use: $ `module load cray-memkind`

- **Preliminary release will expose this feature via environment variables**
  - Users select: Affinity, Policy and PageSize
  - MPICH_ALLOC_MEM_AFFINITY = DDR or MCDRAM
    - DDR = allocate memory on DDR (default)
    - MCDRAM = allocate memory on MDCRAM
  - 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)

Using MPI_Alloc_mem()

- **Not restricted to be used only for communication buffers, or MPI's internal buffers**

- **Can also be used to allocate application's data buffers**

- **Cray MPI does not register the memory returned by Alloc_mem()**

- **Cray MPI also does not "touch" memory allocated via Alloc_mem()**
  - NUMA affinity resolved when the memory pages are first touched by the process/threads
  - It is not ideal from a NUMA perspective to have the master thread alone touch the entire buffer right after allocation

- **MPI_Alloc_mem returns page-aligned memory for all page sizes**

Typical MCDRAM Use Cases

- **When the entire data set fits within MCDRAM on a Quad/Flat system:**

```
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

- **Alternate solutions needed to utilize hugepage memory on MCDRAM, when the data set per node exceeds 16 Gbytes**
  - 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

Using MCDRAM, Dataset size > 16GB

- **Quad/Flat mode, without numactl options:**
  - `malloc()`, `memalign()` will use DDR first
  - Can access MCDRAM via `hbw_*` or compiler directives
  - `craype-hugepages` module honored *only* on DDR
  - `hbw_malloc()` will return memory backed by basepages
  - Memkind can be used to get 2M hugepages on MCDRAM (but not larger)

- **Users need to identify critical buffers and use `MPI_Alloc_mem()` to allocate hugepages with larger page sizes, and set affinity to MCDRAM**

- **Use following environment variables:**

  `MPICH_ALLOC_MEM_AFFINITY=M` (or MCDRAM)

  `MPICH_ALLOC_MEM_PG_SZ = 16M` (16M hugepages)

  `MPICH_ALLOC_MEM_POLICY = P` (or Preferred)

Using MCDRAM, Dataset size > 16GB (continued)

- **Quad/Flat mode, with `numactl --membind=1`**
  - `malloc()` and `memalign()` will use MCDRAM
  - Hugepage allocations via the `craype-hugepages` module now possible on MCDRAM
  - But, MCDRAM space is limited. Scaling issues

- **Users can identify buffers *not* critical to application performance and use `MPI_Alloc_mem()` to set affinity to DDR**

- **Use following env. vars:**

  `MPICH_ALLOC_MEM_AFFINITY=D` (or DDR)

  `MPICH_ALLOC_MEM_PG_SZ` = <as needed, defaults to 4KB base pages>

  `MPICH_ALLOC_MEM_POLICY = P` (or Preferred)

Summary and Recommendations

- Optimizations in Cray MPI to improve pt2pt and collective latency on KNL
- Enhancements in Cray MPI to enable users to best utilize the MCDRAM technology on KNL (hugepages)
- New solutions in Cray MPI to offer Thread-Hot capabilities on Intel Xeon and Intel KNL architectures
- MPI-only works quite well on KNL

Summary and Recommendations (continued)

- Necessary to use –r1 to reduce performance variability
- Node rank reordering
- Load Balance
- Using hugepages on MCDRAM can improve large message communication performance
- Multi-threaded MPI is an important tool on KNL for hybrid applications
- Asynchronous communication can hide/overlap communication overheads on KNL
- Collectives implemented with user pt2pt is strongly discouraged
  - Especially for alltoall, bcast, and gather
  - Very unlikely pt2pt will perform better
  - If they do, please file a bug with Cray

CRAY

COMPUTE | STORE | ANALYZE

Thank You!

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.