Cray MPI for KNL

Peter Mendygral
pjm@cray.com

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, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM, REVEAL. 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.

XC Topology and Aries

- I will not be covering details of the XC topology and Aries interconnect

- Please refer to the following document or feel free to talk with me at any time the rest of the workshop

http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

Agenda

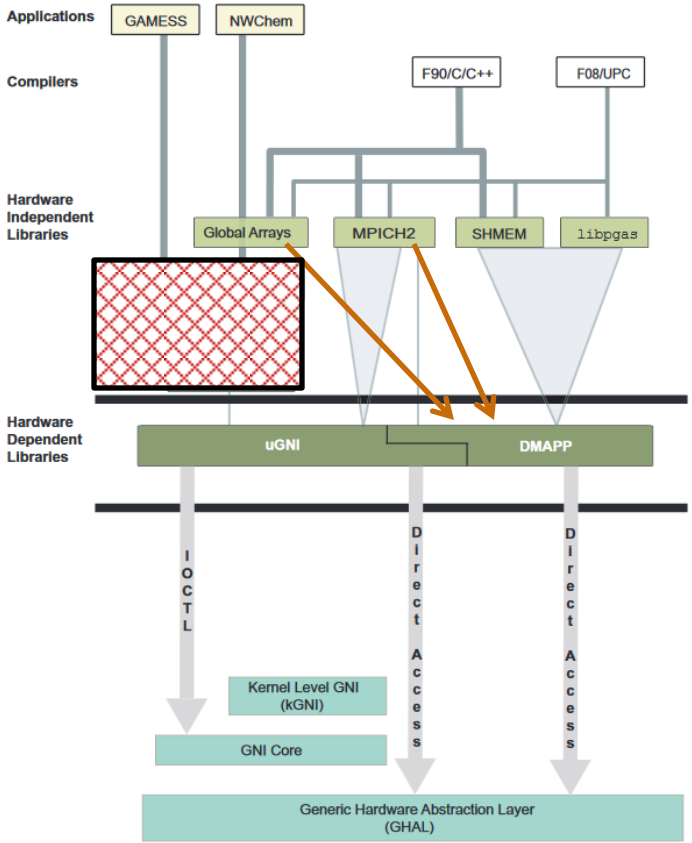
- **Brief Introduction to Cray MPICH**
- **Specific KNL optimizations including MCDRAM**
- **Optimizations for Hybrid (MPI/OpenMP) applications**
  - Application study of astrophysics code Wombat
- **Q&A (feel free to ask questions along the way)**

Introduction to Cray MPICH

Brief Introduction to Cray MPICH

- **Cray MPI compliant with MPI 3.1**
  - Merge to ANL MPICH 3.2rc1 – in MPT 7.3.0 (Dec 2015)
- **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)
- **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

Figure 1. GNI and DMAPP Software Layers



Cray MPI Resources

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

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

Key Environment Variables for XC

- **MPICH_RANK_REORDER_METHOD**
  - 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 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
    - export MPICH_RANK_REORDER_METHOD=3

Key Environment Variables for XC

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

Key Environment Variables for XC

- **MPICH_USE_DMAPP_COLL / MPICH_RMA_OVER_DMAPP**
  - 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
    - MPI_Put / MPI_Get / MPI_Accumulate
  - To use (link with libdmapp):
    - Collective use: export MPICH_USE_DMAPP_COLL=1
    - RMA one-sided use: export MPICH_RMA_OVER_DMAPP=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

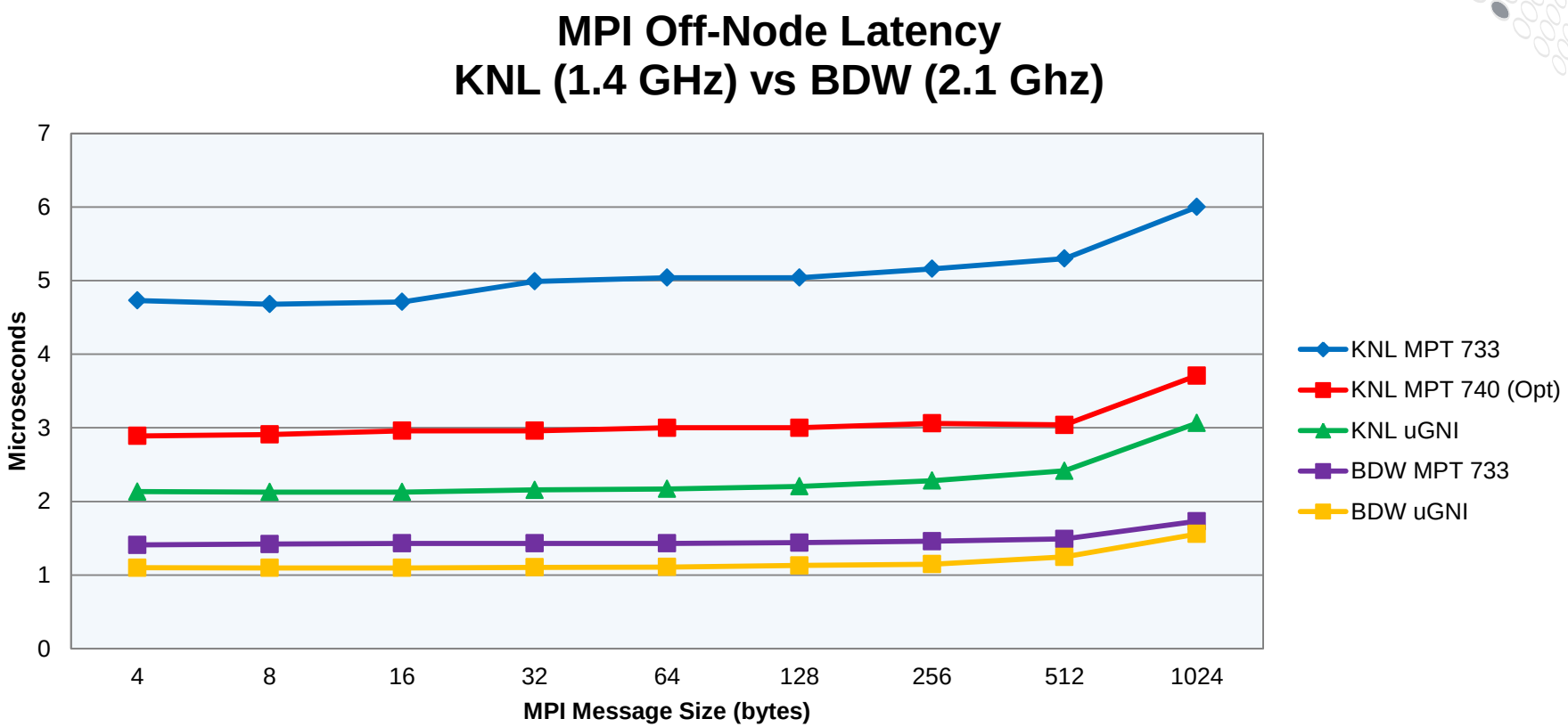
- **Specific Collective Algorithm Tuning**
  - Different algorithms may be used for different message sizes in collectives (e.g.)
    - Algorithm A might be used for Alltoall for messages < 1K.
    - Algorithm B might be used for messages >= 1K.
  - To optimize a collective, you can modify the cutoff points when different algorithms are used. This may improve performance.
  - 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

KNL Optimizations

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 when adjusting to the slower CPU frequency)
- **Optimizing the “critical path”**
  - More inlining of small functions
  - Using higher compiler optimization
  - Hand-optimizations (avoid taking branches in critical path)
  - Disable FMA sharing when not needed
  - Provide a KNL-optimized memcpy

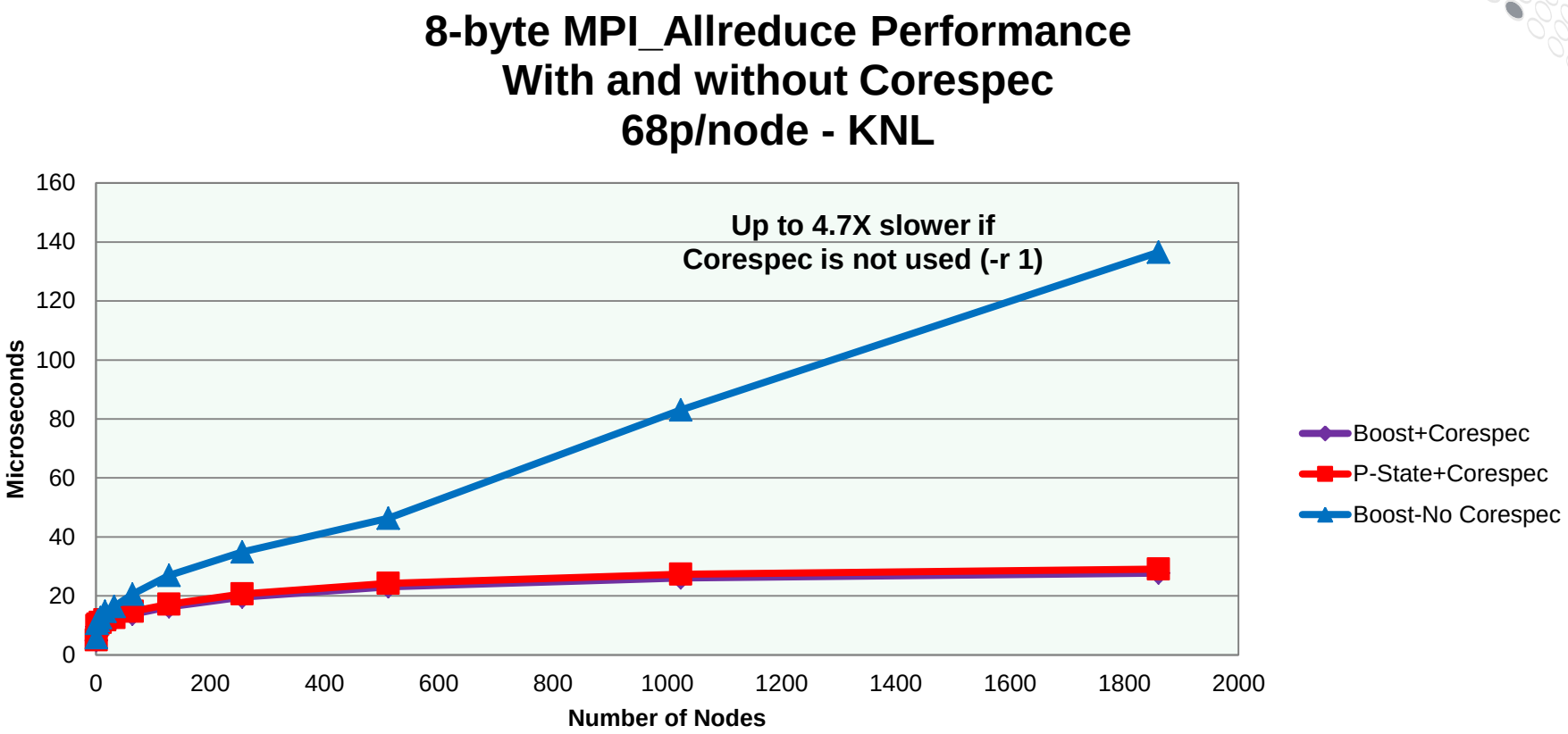
MPI Off-node Latency on KNL



OS Noise Plays a Role – How Big?

- **Studied performance with and without corespec (-r 1)**
- **MPI Latency: Collectives 1P / node**
  - Results show 5.5X slower when not using corespec
- **MPI Latency: Collectives 2-68 P / node**
  - Results show 4.7X slower when not using corespec

With and Without Corespec



Cray MPI support for MCDRAM on KNL

- **Cray MPI will offer 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
- **Preliminary release will expose feature via 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 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)
- **Follow-on release will offer Info Key Support for MPI_Alloc_mem and MPI_Win_allocate**
  - Allows user to specify characteristics via Info keys for each call

Cray MPI support for MCDRAM on KNL

- 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.
  (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

Cray MPI support for MCDRAM on KNL Typical 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 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 MPI support for MCDRAM on KNL: Typical Use cases (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 env. vars:**

  MPICH_ALLOC_MEM_AFFINITY=M (or MCDRAM)

  MPICH_ALLOC_MEM_PG_SZ = 16M (16M hugepages)

  MPICH_ALLOC_MEM_POLICY = P (or Preferred)

Cray MPI support for MCDRAM on KNL: Typical Use cases (Datset size > 16GB)

- **Quad/Flat mode, with numactl --membind=1**
  - Malloc(),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)

Using MPI_Alloc_mem/MPI_Free_mem with a 3DFFT Kernel (Fortran)

```
..
lenr    = nx * ny * mynz
lenc    = MAX((nxd2p1 * ny * mynz),
                (nxd2p1 * nz * myny))
lenm    = MAX((nxd2p1 * ny * mynz),
                (nxd2p1 * nz * myny))
rc_grid = FFTW_ALLOC_REAL(lenr * nvars)
cc_grid = FFTW_ALLOC_COMPLEX(lenc * nvars)
mc_grid = FFTW_ALLOC_COMPLEX(lenm)

CALL C_F_POINTER(rc_grid, rgrid,
                        (/nx, ny, mynz, nvars/))
CALL C_F_POINTER(cc_grid, cgrid,
                        (/lenc, INT(nvars, C_SIZE_T)/))
CALL C_F_POINTER(mc_grid, mgrid, (/lenm/))

..
```

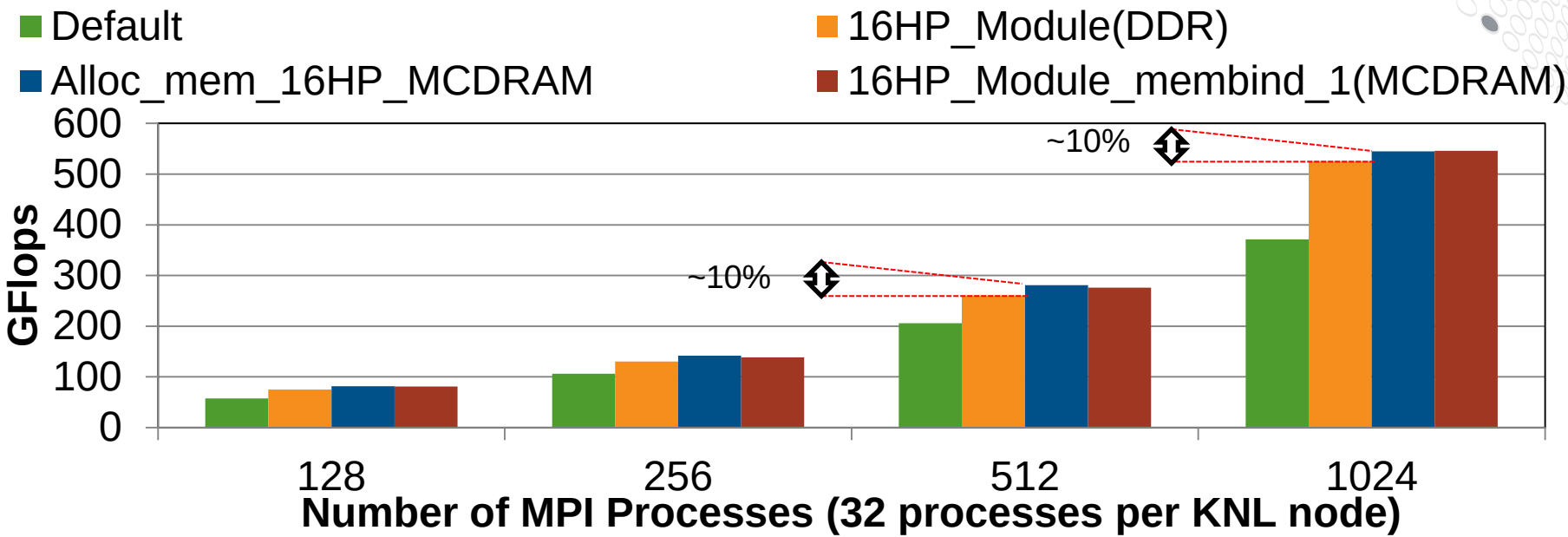
```
lenr    = nx * ny * mynz
lenc    = MAX((nxd2p1 * ny * mynz),
                (nxd2p1 * nz * myny))
lenm    = MAX((nxd2p1 * ny * mynz),
                (nxd2p1 * nz * myny))
info    = MPI_INFO_NULL
CALL MPI_ALLOC_MEM(lenr * nvars * 8_8, info,
                                rc_grid, ierr)
CALL MPI_ALLOC_MEM(lenc * nvars * 16_8, info,
                                cc_grid, ierr)
CALL MPI_ALLOC_MEM(lenm * 16_8, info,
                                mc_grid, ierr)

CALL C_F_POINTER(rc_grid, rgrid,
                        (/nx, ny, mynz, nvars/))
CALL C_F_POINTER(cc_grid, cgrid,
                        (/lenc, INT(nvars, C_SIZE_T)/))
CALL C_F_POINTER(mc_grid, mgrid, (/lenm/))
..
```

Using MPI_Alloc_mem/MPI_Free_mem with a 3DFFT Kernel (Fortran)

```
CALL FFTW_FREE(rc_grid)
CALL FFTW_FREE(cc_grid)
CALL FFTW_FREE(mc_grid)
```

```
CALL MPI_FREE_MEM(rgrid, ierr)
CALL MPI_FREE_MEM(cgrid, ierr)
CALL MPI_FREE_MEM(mgrid, ierr)
```

MCDRAM Experiments with a 3DFFT Kernel



3DFFT Weak scaling (Data Grid: 1024, 1024, 1024)

MPI_Alloc_mem with hugepages offers same performance as using membind=1

Hugepages on MCDRAM performs better than DDR with the same hugepage size

Using MPI_Alloc_mem can help cases where the entire data set does not fit within MCDRAM

Cray MPI support for MCDRAM on KNL: On-going effort

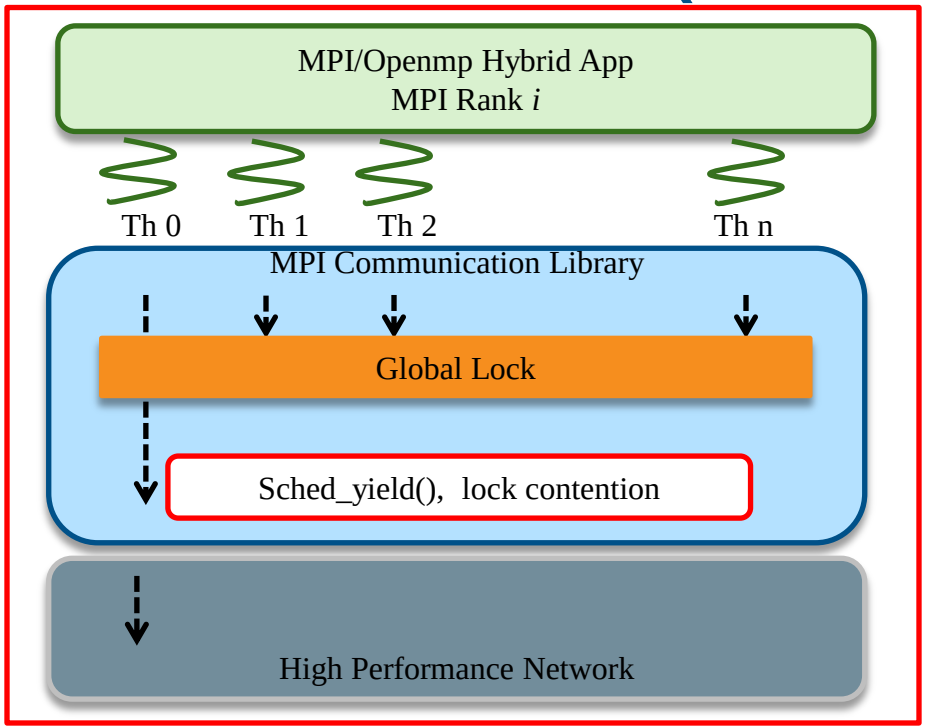
- Info key support for MPI_Alloc_mem and MPI_Win_allocate()
- Env. vars affect the entire job, info keys can offer fine-grained controls for each memory allocation
- Info keys can be used to allocate 64M Hugepages on MCDRAM for one buffer, and 64M Hugepages on DDR for a different buffer in the same job.
- Env. vars are still respected if info keys are not set
- Portable across MPI implementations. Possible to maintain info key names and format consistent between Cray MPI and Intel's MPI implementations. A different MPI impl. could choose to silently ignore the proposed info keys

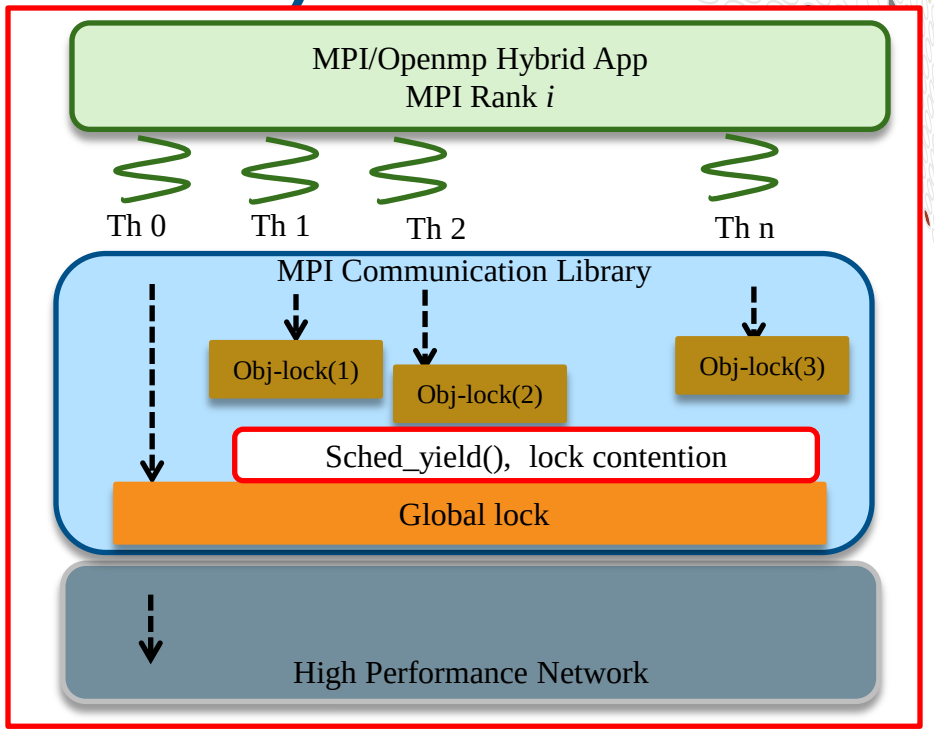
Multi-threaded MPI Support and Optimizations

CORAL NRE MS13 Milestones

Schedule	Deliverables
December 2015	**MS1:** Design Document for the Thread-Scalable MPI NRE.
Q3 2016	**MS2: Demonstrate the benefits of an optimized Cray MPICH implementation on the Theta System with Intel KNL and Aries interconnect.**
Q2 2017	**MS3:** Implement a thread-scalable prototype of Cray MPICH with the CH4 interface and the uGNI-based OFI provider layer. Demonstrate the functional characteristics of this implementation.
Q4 2017	**MS4:** Implement a thread-scalable prototype of ANL MPICH with the CH4 interface and the STL2-based OFI provider layer. This prototype is intended for an experimental system with Intel KNH and Intel STL2.
Q4 2018	**MS5:** Finalize and deliver the Thread-Scalable MPI implementation and source for the Aurora System.

Multi-Threaded MPI (State-Of-The-Art)

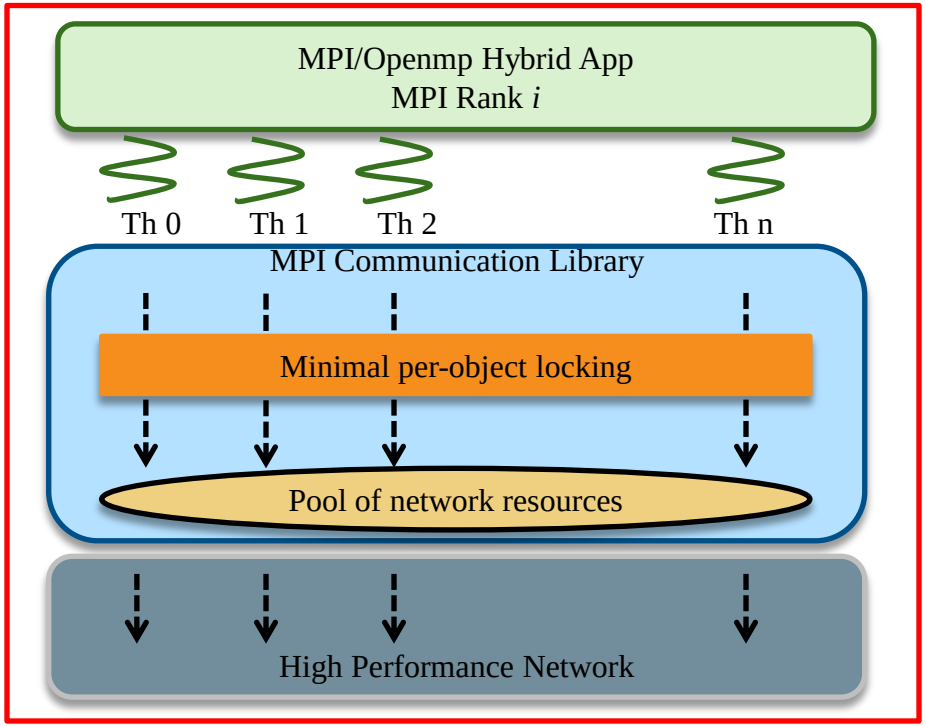




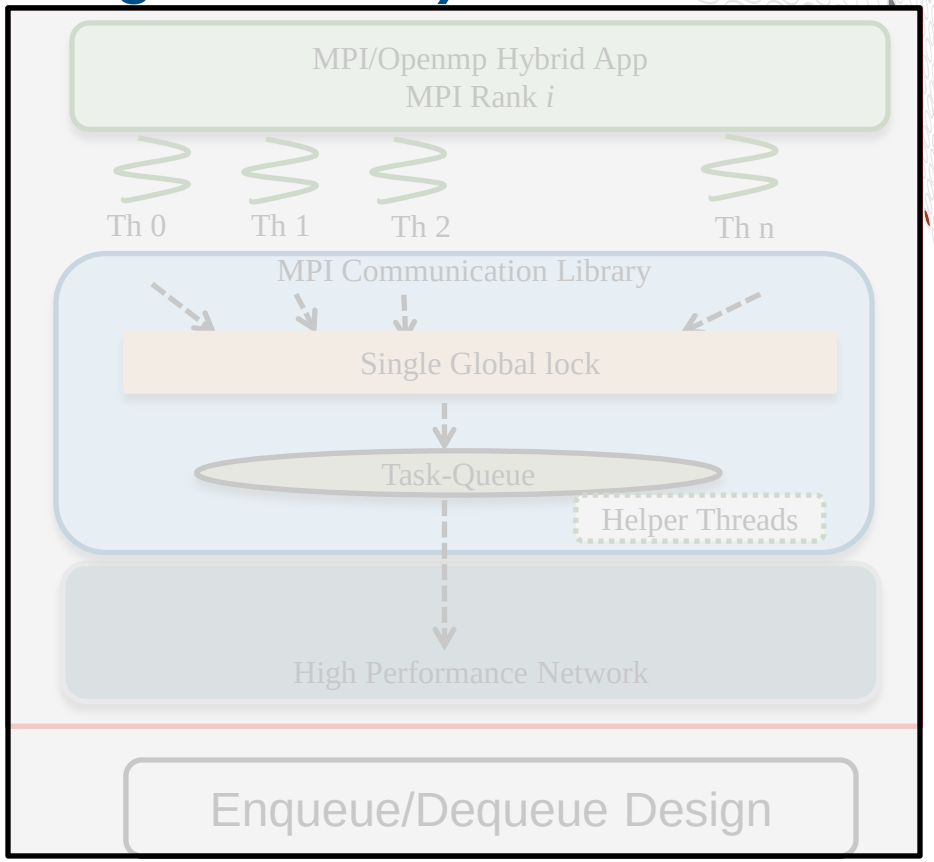
Global lock
(default in Cray MPI)

Per-Object Locks
(Alt. impl. in CrayMPI, "–craympich-mt" link time flag)

Optimized Multi-Threaded MPI (Design Choices)

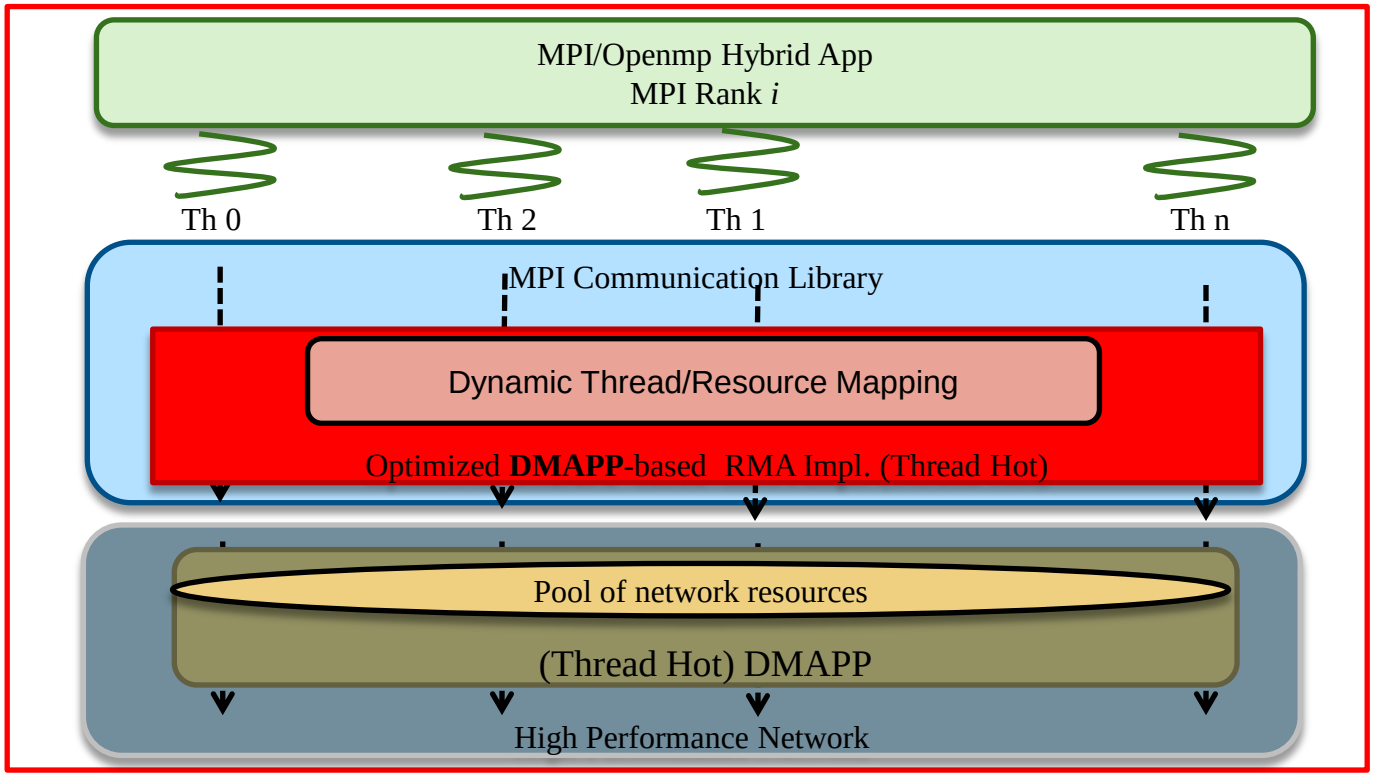


Proposed Thread-Hot Design



Enqueue/Dequeue Design

Thread Hot Communication in Cray MPI



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
- **“Bottom-Up” OpenMP development approach is very common**
  - Most likely will not offer best performance and scaling
- **“Top-Down” SPMD model is more appealing for KNL**
  - 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

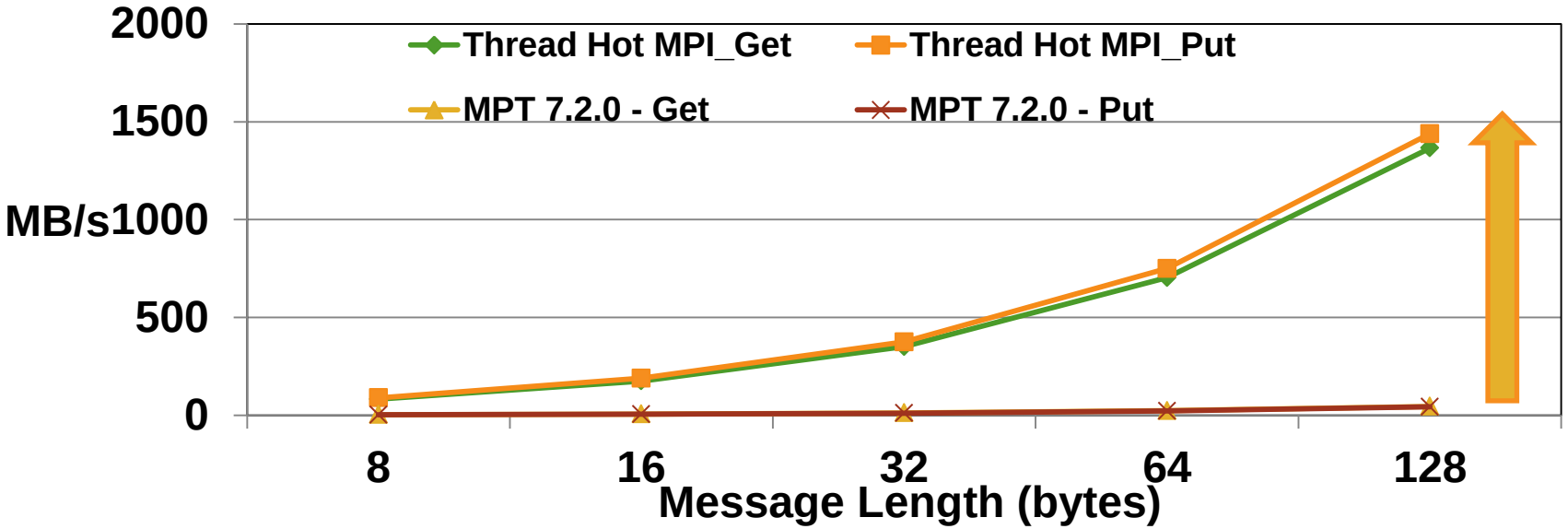
Experimental Setup

- Cray XC systems with Intel Haswell and Broadwell
- *Modified* OSU Micro Benchmarks (OMB) to study multi-threaded MPI Communication performance
  - RMA: osu_put_latency.c, osu_get_latency.c
    osu_put_bw.c, osu_get_bw.c

- Enabling the RMA over DMAPP optimization:
  - Link against DMAPP:
    If the code uses static linking:
    -Wl,--whole-archive,-ldmapp,--no-whole-archive
    If the code uses dynamic linking:
    -ldmapp
  - Set MPICH_RMA_OVER_DMAPP env. variable to 1 (export MPICH_RMA_OVER_DMAPP=1)

MPI-3 RMA Communication Bandwidth

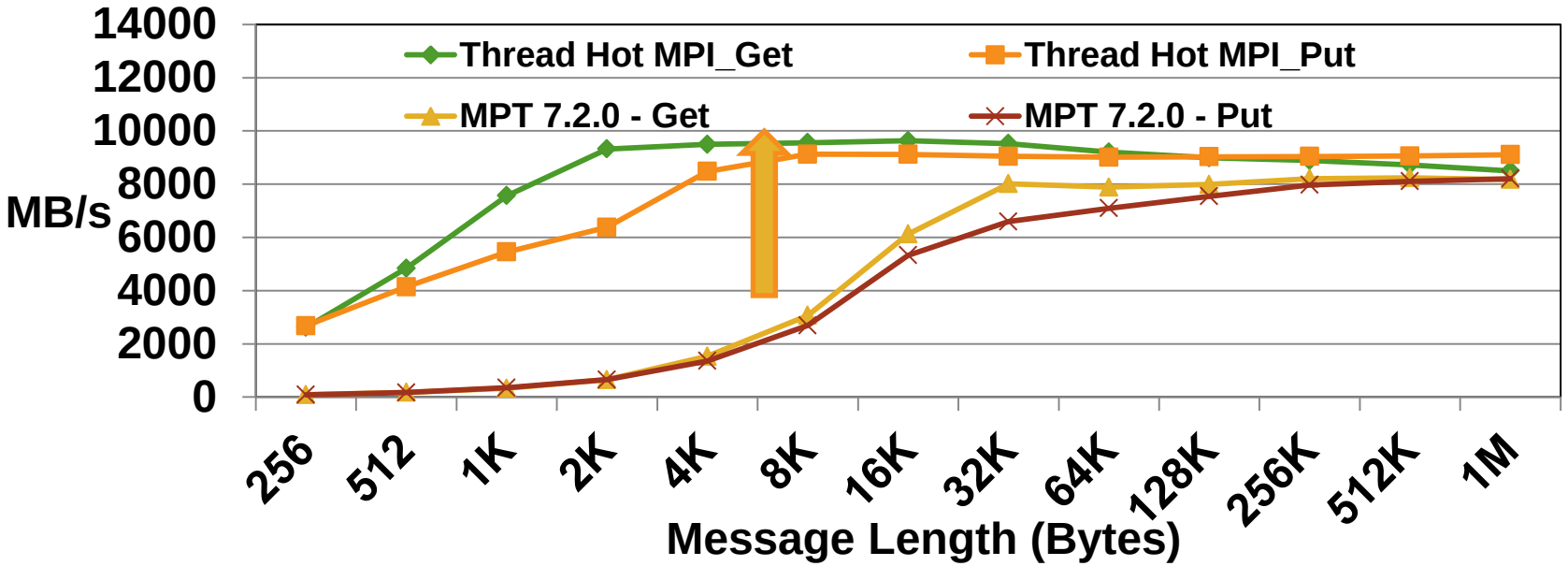
1 MPI process per node, 32 threads, Haswell, small messages



- **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**

Wombat Project Hybrid Application Study

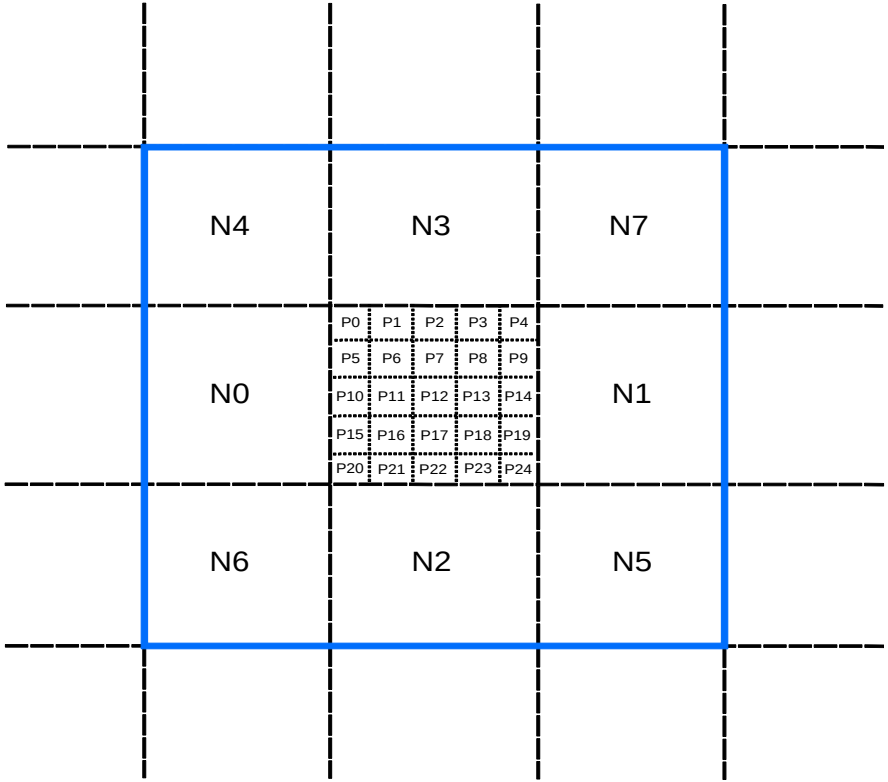
- **Wombat is being developed through a collaboration between Cray Programming Environments and the University of Minnesota Institute for Astrophysics**
  - Peter Mendygral (Cray) is lead developer
  - Tom Jones (UofM) supervises graduate students contributing to the code and drives scientific goals
  - Other contributors/users: Dongsu Ryu (UNIST), Julius Donnert (INAF)
- **Scientific goal is to study turbulence in astrophysical fluids over cosmological scales**
  - MHD + dark matter
- **Application goal is to develop a code capable of achieving the science goals on the latest HPC architectures**

Primary Development Concerns

- **Two main issues drove the design of Wombat and explain why other codes have not been sufficient for the science**
  - Scaling to extreme core count required to get to resolutions needed for MHD turbulence
  - Load balancing for SMR/AMR and dark matter particles
- **The approach to these problems in Wombat was**
  - Make communication matter as little as possible
  - Wide OpenMP on a node to soften impacts of load imbalances (and hardware imbalances) as much as possible before communicating work between ranks
  - Data structures that reduce AMR/SMR complexity and avoid significant global communication for refined patch tracking
  - Do it all in Fortran as that's what works best for me
    - Wombat uses object oriented features from Fortran 2003 and 2008

Domain Decomposition

- Domain decomposition is represented in the data classes and structures
- “Domain” is an array of “Patches” managed by a MPI rank
- “Patch” is a self-contained, self-describing piece of the world grid at some fixed logical location



Domains and Patch

- **Domains manage bookkeeping for an array of Patches**
  - Track a Patch's neighbors
  - Manage allocating/deallocating a Patch's internal grid arrays as needed
  - Multiple Domains are used on a rank for accepting Patches from neighbors

- **Patches are**
  - Of some uniform fixed size (for a given Domain refinement level)
  - Fixed in a location that is known for all times by all ranks

- **Patches provide**
  - An atomic unit of work
  - Units to thread across
  - Unit to transfer for load balancing
  - A level of cache blocking

High-level OpenMP

Option A – "bottom up"

```
! Keep OpenMP within a "compute" loop

DO WHILE (t .LT. tend)

   DO patch = 1, npatches

      CALL update_patch()

      …CALL MPI…

   END DO

END DO

SUBROUTINE update_patch()

   !$OMP PARALLEL DO
   DO i = 1, nx
   …do work…
   END DO

END SUBROUTINE
```

Option B – "SPMD"

```
! Move OpenMP near the top of the call stack

!#OMP PARALLEL
DO WHILE (t .LT. tend)

   !#OMP DO
   DO patch = 1, npatches

      CALL update_patch()

      …CALL MPI…

   END DO

END DO
```

High-level OpenMP

- **Benefits of high-level SPMD OpenMP**
  - Application more closely mimics completely independent processes
    - Less likely to be in the same portion of code at the same time
    - Bandwidth competition may decrease
    - Amdahl's law
  - Threads are less coupled => infrequent thread synchronization
  - Much less likely to have issues with memory conflicts between threads
  - Simpler to implement when done right
    - Large reduction in the amount of OpenMP directives
    - Very little variable scoping needed as most everything is shared => reduced memory footprint
  - Easier to make use of all cores on node (e.g., 68) that can be hard to use for domain decomposition reasons
  - Effective way to manage natural hardware induced imbalance and algorithmic load imbalance

High-level OpenMP

- **Challenges for high-level SPMD OpenMP**
  - Requires full understanding of data dependencies and potential for race conditions
  - For best performance requires revisiting approach to MPI
    - Goal should be to remove any thread synchronization you can
    - Serializing MPI will limit the benefit and scalability of SPMD
  - In my experience, it is easier to implement SPMD in a data centric model
    - Present work as independent units
    - Let threads work on that set in any order with a non-static schedule
  - MPI work should be treated the same as data if possible
    - Independent units of communication to be worked on in any order with non-static schedule

Wombat Driver and Parallel Region

Setup and object constructors

Thread parallel region

Array allocation and initialization

Time step loop

DomainSolver%solve(mhd)
DomainSolver%solve(ct)
DomainSolver%solve(passive)
DomainSolver%solve(particle kick+drift)
FMGSolver%solve(all levels)
DomainSolver%solve(particle kick)

I/O data dump(s)

Update time step

Array cleanup

Object destructors

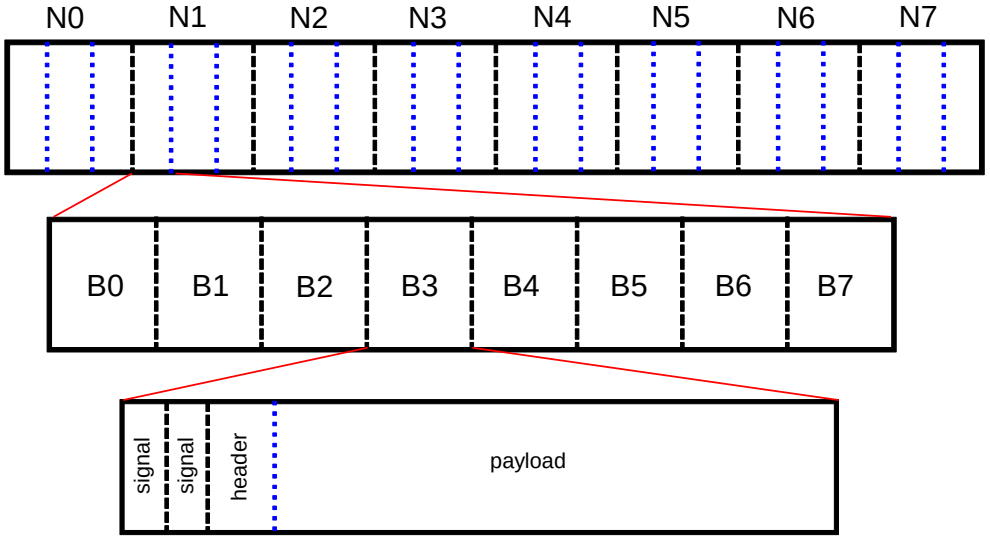
Simulation complete

Communication Concerns

- **If a rank is made much wider with threads, serialization around MPI will limit thread scaling and overall performance**
  - Nearly all MPI libraries implement thread safety with a global lock
  - Cray (and other vendors) is addressing this issue
- **Wide OpenMP also means more communication to process per rank**
  - Every Patch now has its own smaller boundaries to communicate
  - Starts tipping the behavior towards the message rate limit
- **Slower serial performance of KNL => maybe look for the lightest weight MPI layer available**
  - MPI-RMA over DMAPP on Cray systems is a thin software layer that achieves similar performance to SHMEM

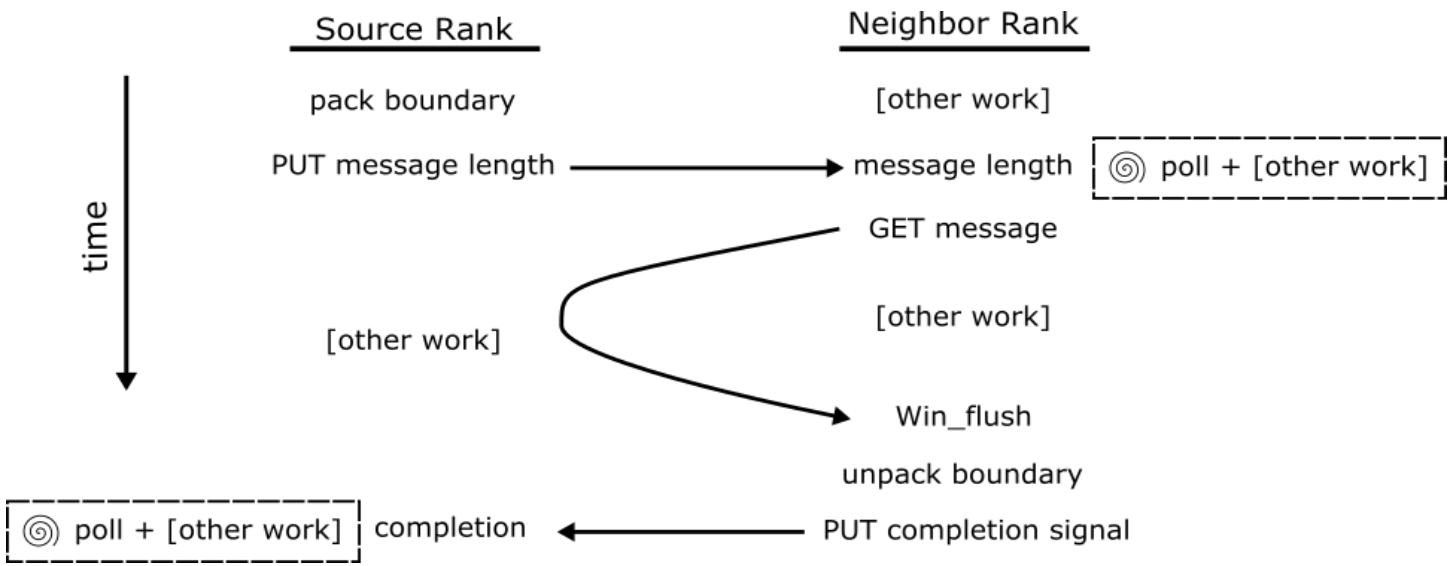
Single RMA Window Buffer

- Single buffer used for (almost) all communication
- Messages can be processed concurrently if MPI allows it
- Design is similar to mailboxes within MPI
  - Can process an arbitrary amount of communication



RMA Boundary Communication Cycle

- **Single passive RMA window used for the duration of the application**
  - No explicit synchronization between ranks
  - RMA semantics make computation/communication overlap simpler to achieve
- **All "solvers" in Wombat utilize a generalized class that implements the communication/computation cycle below**



Thread Hot MPI-RMA

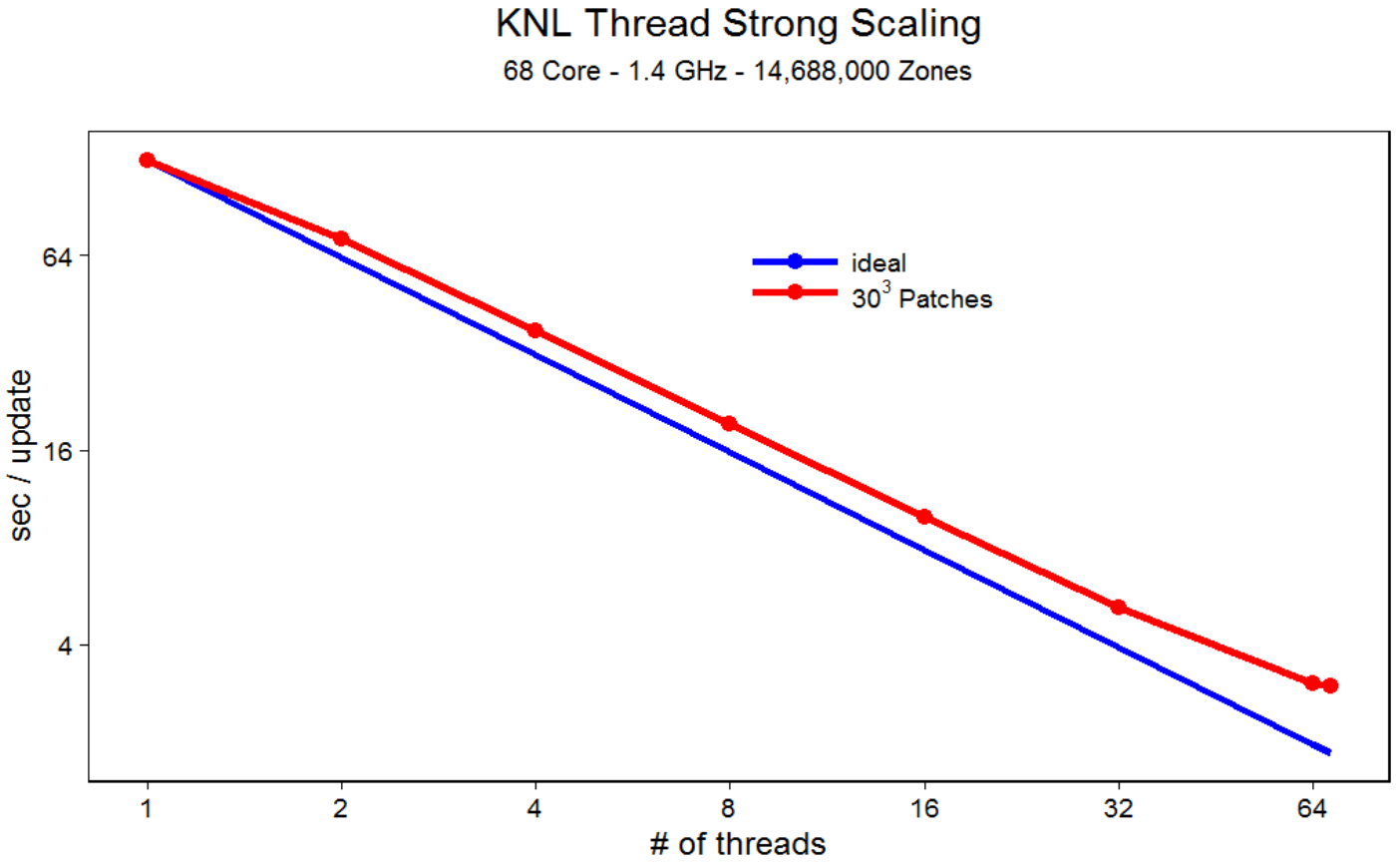
- **Future release of Cray MPICH will include new performance feature that allows for efficient message completion in a threaded region**
  - Can call completion routines (e.g., MPI_WIN_FLUSH) in a threaded region (not required)
  - Threads collaboratively complete messages from all threads
  - Removes the need for any additional thread barriers after MPI_PUT/MPI_GET/etc in threaded region

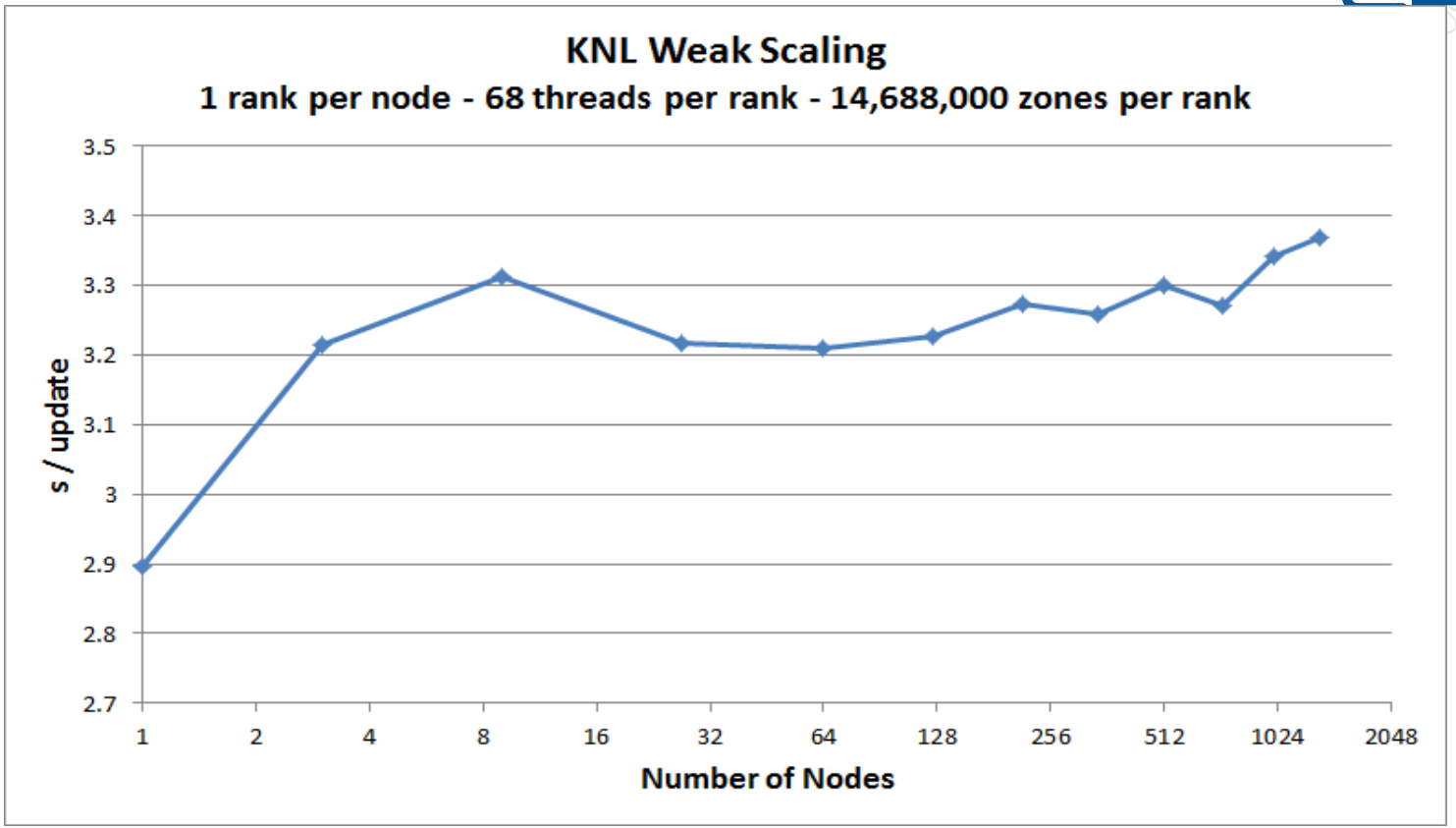
```
!$OMP DO
DO n = 1, n_neighbors

CALL MPI_PUT(…)

END DO
!$OMP END DO

!$OMP MASTER
CALL  MPI_WIN_FLUSH_ALL()
!$OMP BARRIER
```

```
!$OMP DO
DO n = 1, n_neighbors

CALL MPI_PUT(…)

END DO
!$OMP END DO NOWAIT

CALL  MPI_WIN_FLUSH_ALL()
```

Wombat Benchmarking Notes

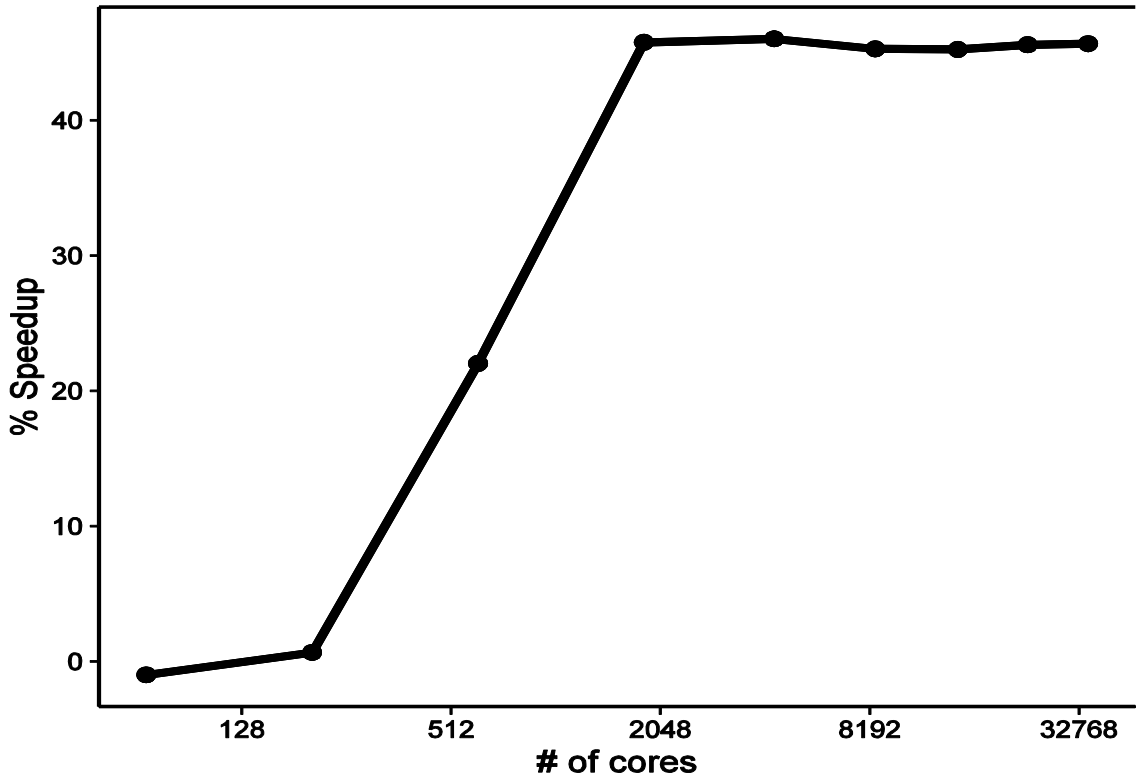
- **Only MHD was enabled on fixed grid**
- **In all cases 2 MB hugepages were used**
  - module load craype-hugepages2MB
  - Loaded at link and run time
  - Link with dmapp (statically)
    -Wl,--whole-archive,-ldmapp,--no-whole-archive
- **The following environment variables were set**
  - export MPICH_MAX_THREAD_SAFETY=multiple
  - export MPICH_RMA_OVER_DMAPP=1
- **CCE was always used**
  - Needed for vectorization in eigenvector/flux calculation
- **KNL was configured with MCDRAM as cache**

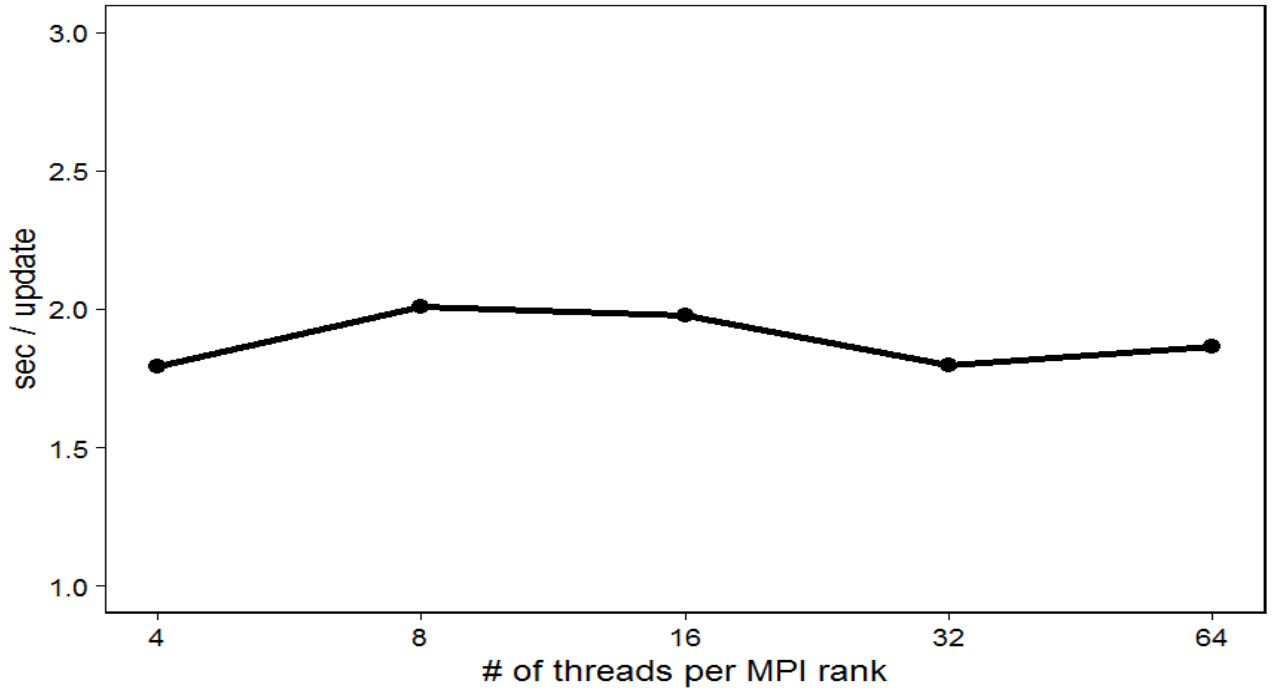




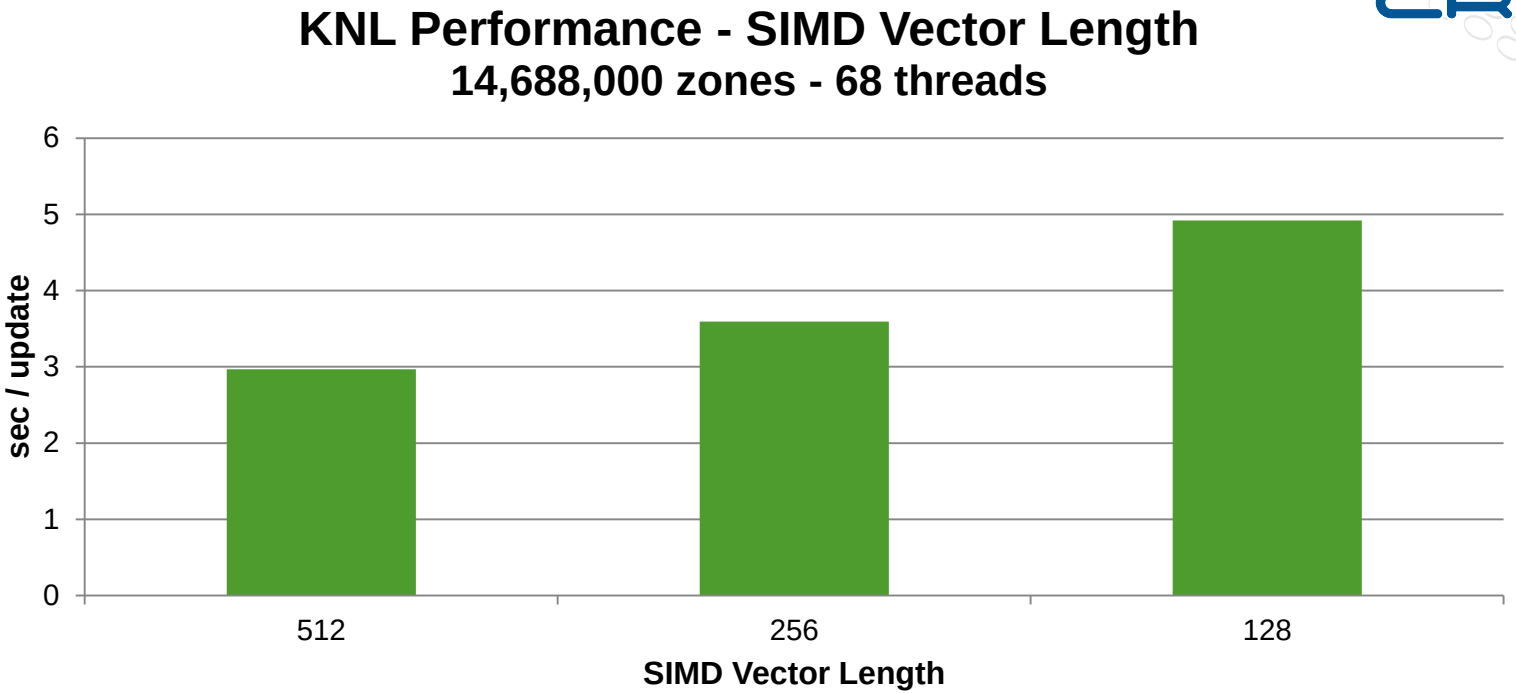
Thread-Hot / Global Lock MPI-RMA Speedup on KNL

1 Rank per Node - 68 threads per rank - 14,688,000 Zones per Rank





- Less than 5% difference between full ranks and full threads
- Ideal for application like Wombat for fixed grid is 0%



- Vector length very important
- Significant effort went into making solver loops (compute and copy) vectorize

Summary and Recommendations

- Optimizations in Cray MPI to improve pt2pt and collective latency on KNL
- Enhancements in Cray MPI to enable users best utilize the MCDRAM technology on KNL
- New solutions in Cray MPI to offer Thread-Hot capabilities on Intel Xeon and Intel KNL architectures
- Design and development details of Wombat, a high performance astrophysics application that relies on multi-threaded MPI-3 RMA implementation in Cray MPI

- MPI-only works quite well on KNL
  - Threading can be helpful, but unless SPMD with “thread-hot” MPI is used scaling to more than 2-8 threads not recommended
- Necessary to use –r1 to reduce performance variability
- Using hugepages on MCDRAM can improve large message communication performance
- Multi-threaded MPI will be a key 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

Q&A