Argonne NATIONAL LABORATORY

TUNING MPI ON THETA

SUDHEER CHUNDURI

PERFORMANCE ENGINEERING GROUP

ARGONNE LEADERSHIP COMPUTING FACILITY

ARGONNE NATIONAL LABORATORY

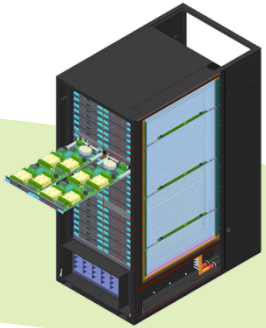
Acknowledgements:
Krishna Kandalla, Cray Inc
Peter Mendygral, Cray Inc

OUTLINE

- Cray XC40 (Theta) network architecture and software stack
- MPI software stack on Theta (Cray MPICH)
- Baseline MPI performance on Theta
- MPI tuning parameters
  - KNL/MCDRAM optimizations
  - Optimizations for Hybrid MPI+OpenMP applications

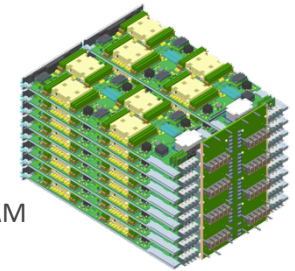
INTRODUCTION TO CRAY XC NETWORK

THETA SYSTEM OVERVIEW

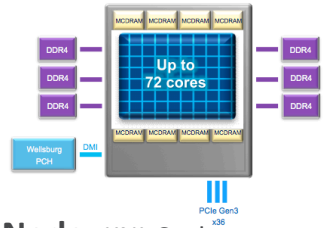


Cabinet: 3 Chassis, 75kW liquid/air cooled
510.72 TF

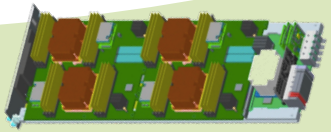
System: 24 Cabinets
4392 Nodes, 1152 Switches
Dual-plane, **12 groups**, Dragonfly ~12.14 TB/s Bi-Sec
11.69 PF Peak



Chassis: 16 Blades, 16 Cards
64 Nodes, 16 Switches
170.24 TF 1TB MCDRAM, 12TB DRAM



Node: KNL Socket
192 GB DDR4 (6 channels)
2.66 TF, 16GB MCDRAM

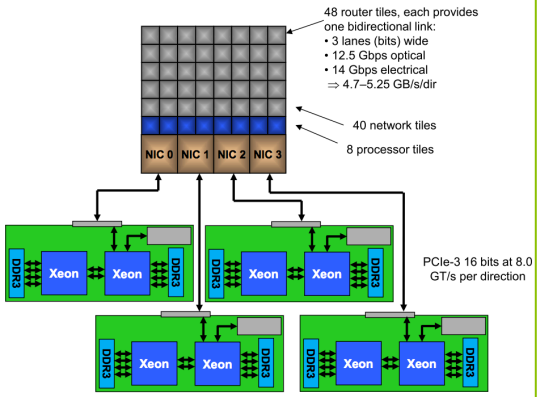


Compute Blade:
4 Nodes/Blade + Aries switch (connected over 16 GB/s PCIe interface)
128GB SSD
10.64 TF 64GB MCDRAM
768GB DRAM

ARIES DRAGONFLY NETWORK

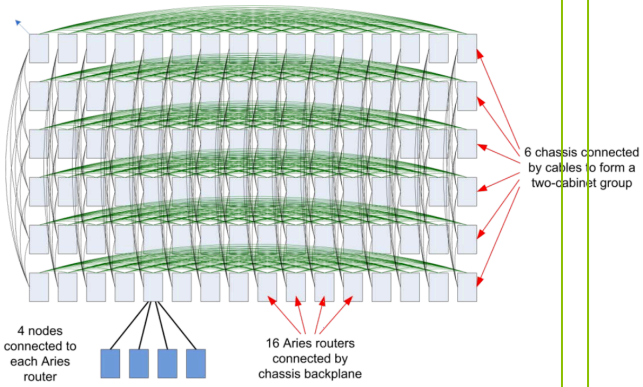
Aries Router:

- 4 Nodes connect to an Aries
- 4 NIC's connected via PCIe
- 40 Network tiles/links
- 4.7-5.25 GB/s/dir per link



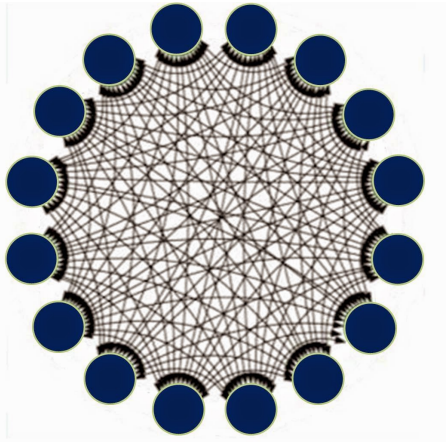
Connections within a group:

- 2 Local all-to-all dimensions
  - 16 all-to-all horizontal
  - 6 all-to-all vertical
- 384 nodes in local group



Connectivity between groups:

- Each group connected to every other group
- Restricted bandwidth between groups



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

CRAY XC NETWORK SOFTWARE STACK

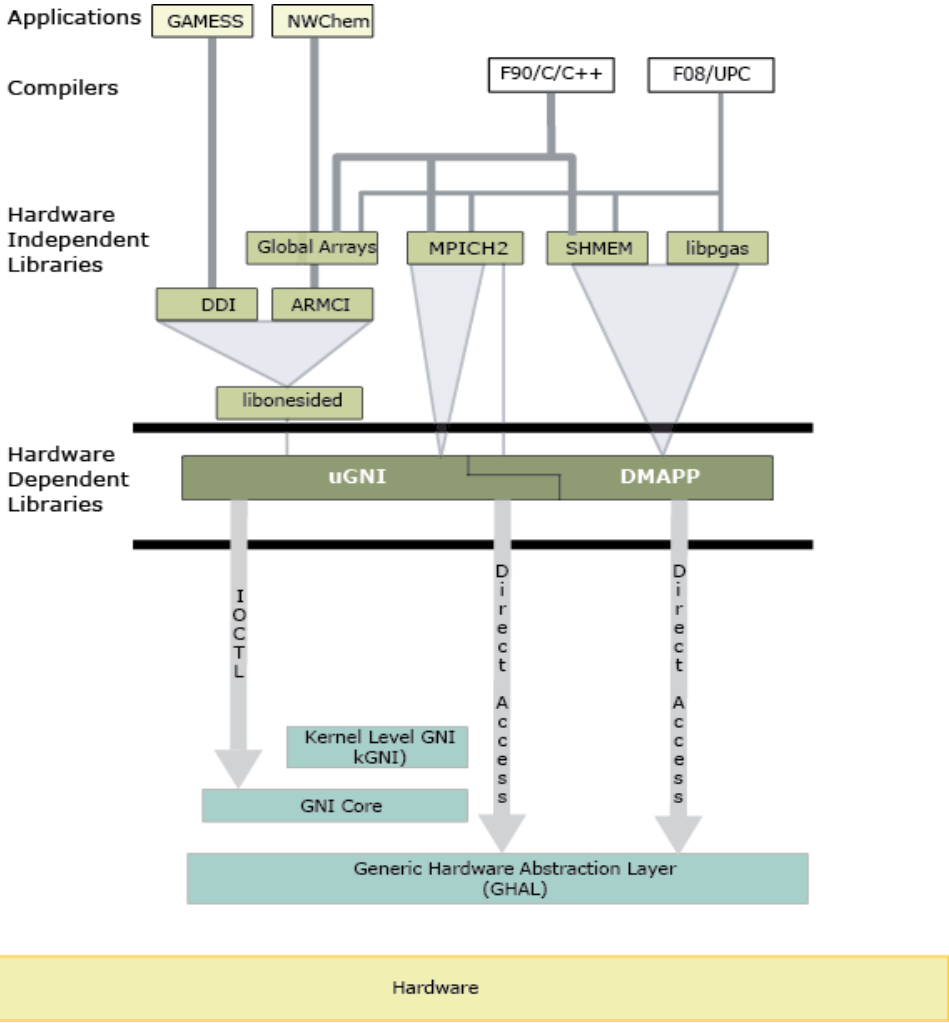
DMAPP - Distributed Shared Memory Application APIs

(shared memory)

GNI - Generic Network Interface

(message passing based)

uGNI and DMAPP provide low-level communication services to user-space software



INTRODUCTION TO CRAY MPICH

BRIEF INTRODUCTION TO CRAY MPICH

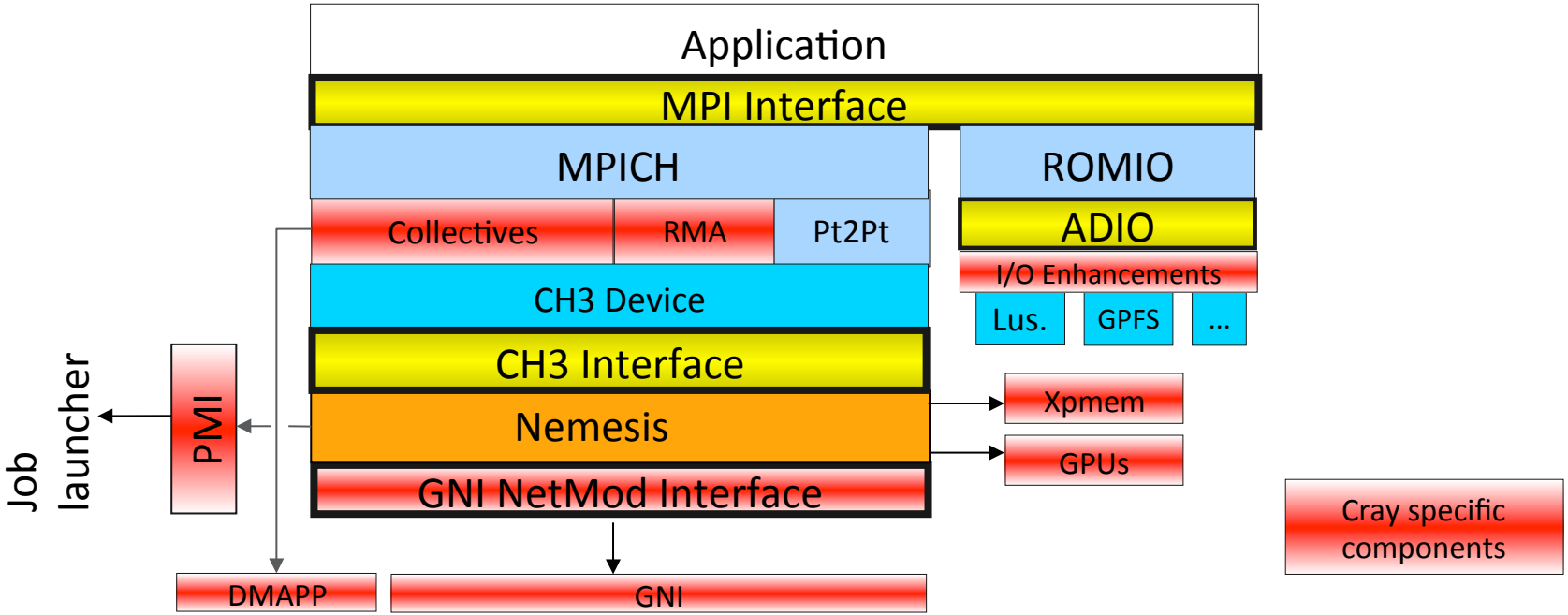
- Cray MPI compliant with MPI 3.1
  - Cray MPI uses MPICH3 distribution from Argonne
  - Merge to ANL MPICH 3.2 – latest release MPT 7.7.0 (Dec 2017)

- 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

CRAY MPI SOFTWARE STACK (CH3 DEVICE)



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

MPI BASELINE PERFORMANCE

MPI BANDWIDTH AND MESSAGING RATE

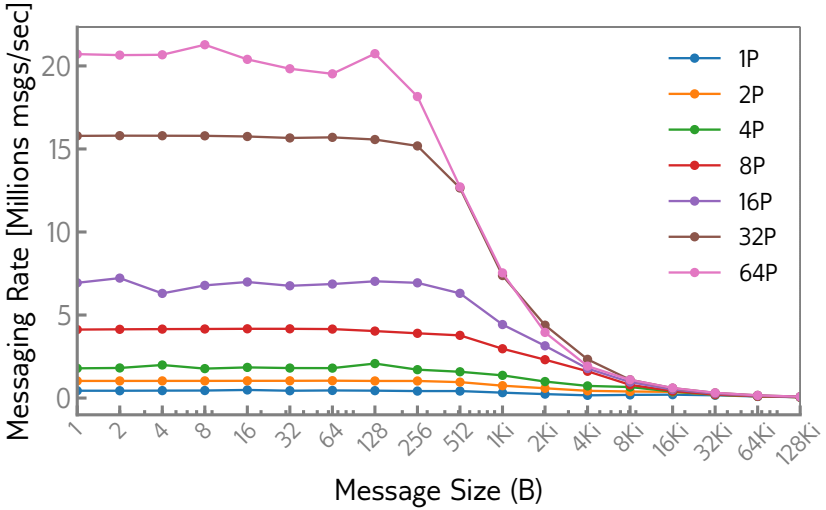
OSU PtoP MPI Multiple Bandwidth / Message Rate Test on Theta

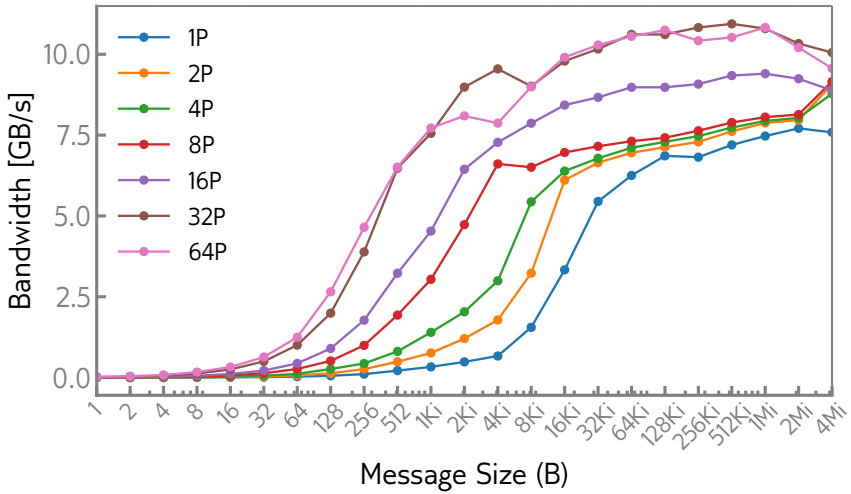
Messaging Rate:

- Maximum rate of **21.2 MMPS**
  - At 64 ranks per node, 1 byte, window size 128
- Increases generally proportional to core count for small message sizes

Bandwidth:

- Peak sustained bandwidth of **11.4 GB/s** to nearest neighbor
- **1 rank (KNL core) capable of only 8 GB/s**
- For smaller messages more ranks improve aggregate off node bandwidth





MPI LATENCY

OSU Ping Pong, Put, Get Latency

Benchmark	Zero Bytes (µs)	One Byte (µs)
Ping Pong	~ 3.07	~ 3.22
Put	~ 0.61	~ 2.9
Get	~ 0.61	~ 4.7

With each extra hop roughly 100 ns additional latency

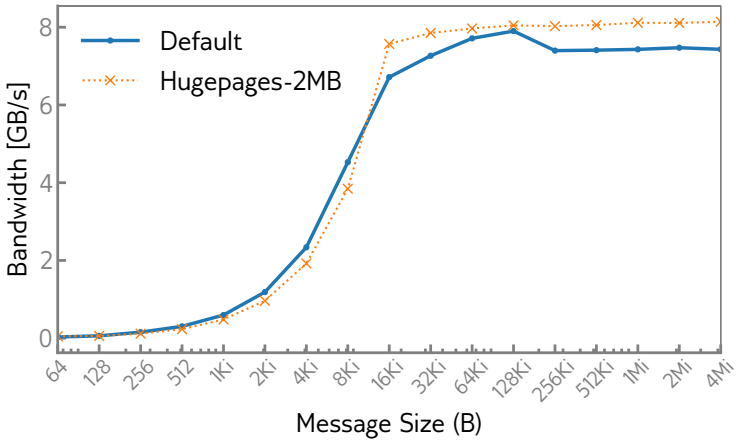
MPI ONE SIDED (RMA)

OSU One Sided MPI Get Bandwidth and Bi-Directional Put Bandwidth

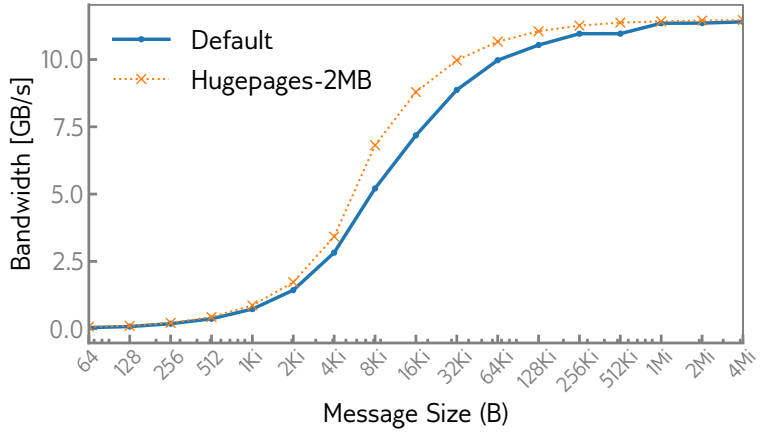
RMA Get
- ~ 8.1 GB/s using RMA over uGNI
- Huge pages help
- uGNI version performs as good as the DMAPP version

RMA Put
- 11.4 GB/s peak bi-directional bandwidth
- Slight benefit from using huge pages

RMA Get



RMA Put Bi-directional



MPI PERFORMANCE TUNING

PROFILING MILC WITH CRAYPAT*

```
module load perftools
<<Build app>>
pat_build -g mpi ./
a.out
<<Run app+pat>>
pat_report PROFILEDIR
```

*note: other profiling tools could as well be used

Table 1: Profile by Function Group and Function

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

PROFILING MILC WITH CRAYPAT

```
=========================================================
=====================
  Total
---------------------------------------------------------
--------------------
  MPI Msg Bytes%                          100.0%
  MPI Msg Bytes                 18,052,938,280.0
  MPI Msg Count                      1,460,959.0 msgs
  MsgSz <16 Count                      157,529.0 msgs
  16<= MsgSz <256 Count                    65.0 msgs
  256<= MsgSz <4KiB Count                2,815.0 msgs
  4KiB<= MsgSz <64KiB Count          1,300,511.0 msgs
  64KiB<= MsgSz <1MiB Count                39.0 msgs
=========================================================
=====================
  MPI_Isend
---------------------------------------------------------
--------------------
  MPI Msg Bytes%                          100.0%
  MPI Msg Bytes                 18,051,670,432.0
  MPI Msg Count                      1,303,378.0 msgs
  MsgSz <16 Count                          16.0 msgs
  16<= MsgSz <256 Count                     0.0 msgs
  256<= MsgSz <4KiB Count                2,812.0 msgs
  4KiB<= MsgSz <64KiB Count          1,300,511.0 msgs
  64KiB<= MsgSz <1MiB Count                39.0 msgs
=========================================================
=====================
```

KEY ENVIRONMENT VARIABLES FOR XC

- MPICH_RANK_REORDER_METHOD
  - Vary 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

- 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"`
- MILC shows around **22%** performance improvement (on average) with rank reordering

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 and **large data footprint**
  - 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 and 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 (GNI's small message interface has an in-built re-transmit option that allows messages to be replayed in case of failures)
  - 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

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
- MPI
  - `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)
- OS noise events in the order of 150-200us* on KNL
- 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`

**for more information refer: https://dl.acm.org/citation.cfm?id=3126908.3126926*

KNL OPTIMIZATIONS

MCDRAM ON KNL

- KNL nodes in cache mode are a good starting point for many applications
  - No extra work for user
  - Typically good performance

- KNL nodes in flat mode is attractive when
  - there is benefit from extra memory of DDR+MCDRAM
  - observing performance variability related to MCDRAM direct map cache*

- Effective use of flat mode requires users understand more about what memory in their application matters and how it is being used
  - All performance critical memory should reside in MCDRAM if possible

**for more information refer: https://dl.acm.org/citation.cfm?id=3126908.3126926*

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 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)

CRAY MPI support for MCDRAM on KNL
TYPICAL 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
- **MPICH_MEMORY_REPORT**
- 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 USER CASES (DATASET SIZE > 16GB)

- 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 (DATASET SIZE > 16GB)

- 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. Memory 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)
```

MULTI-THREADED MPI SUPPORT AND OPTIMIZATIONS

MPI THREAD MULTIPLE SUPPORT IN CRAY MPI

- Thread multiple support for
  - point to point operations (optimized global lock – instead of a pthread_mutex())
  - Collectives (optimized global lock)
  - MPI-RMA (thread hot)

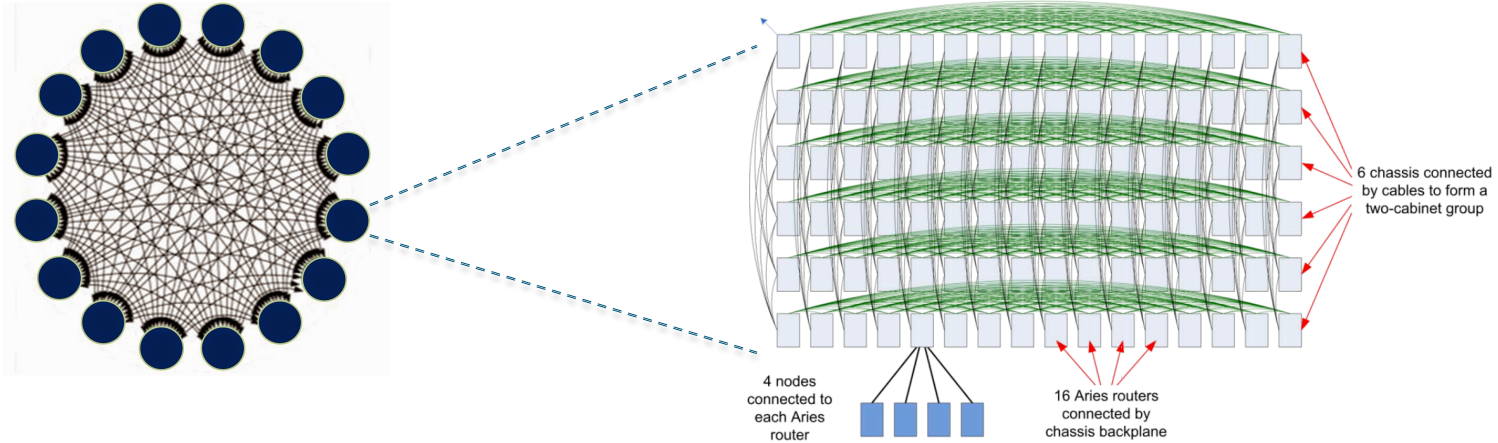
- All supported in default library

- `export MPICH_MAX_THREAD_SAFETY=multiple`

- Global lock optimization on by default (N/A for MPI-RMA)
  - 50% better 8B latency than pthread_mutex() (OSU latency_mt, 32 threads per node, Broadwell)
  - `export MPICH_OPT_THREAD_SYNC=0` falls back to pthread_mutex()

PERFORMANCE VARIABILITY

NETWORK-LEVEL VARIABILITY



- Cray XC Dragonfly topology
  - Potential links sharing between the user jobs
  - High chances for inter-job contention
- Sources of variability -> Inter-job contention
  - Size of the job, Node placement , Workload characteristics , Co-located job mix

NETWORK-LEVEL VARIABILITY

MPI Collectives

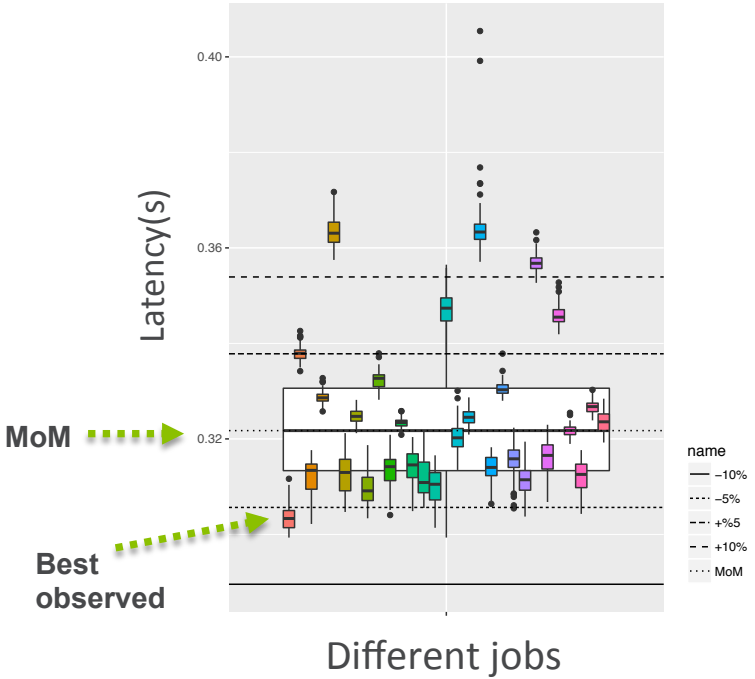
- **MPI_Allreduce** using 64 processes with 8 MB message
- Repeated 100 times within a job
- Measured on several days
  - Changes in node placement and Job mix
- Isolated system run:
  - < **1%** variability
  - Best observed

NETWORK-LEVEL VARIABILITY

"system-configuration" fix was applied to Theta in Jan 2018 that should potentially resolve this to some extent.

MPI Collectives

- **MPI_Allreduce** using 64 processes with 8 MB message
- Repeated 100 times within a job
- Measured on several days
  - Changes in node placement and Job mix
- Isolated system run:
  - < **1%** variability
  - Best observed
- Variability is around **35%**
  - Much higher variability with smaller message sizes (not shown here)
- Each box shows the median, IQR (Inter-Quartile Range) and the outliers



128 nodes Allreduce 8MB 64 PPN

SUMMARY

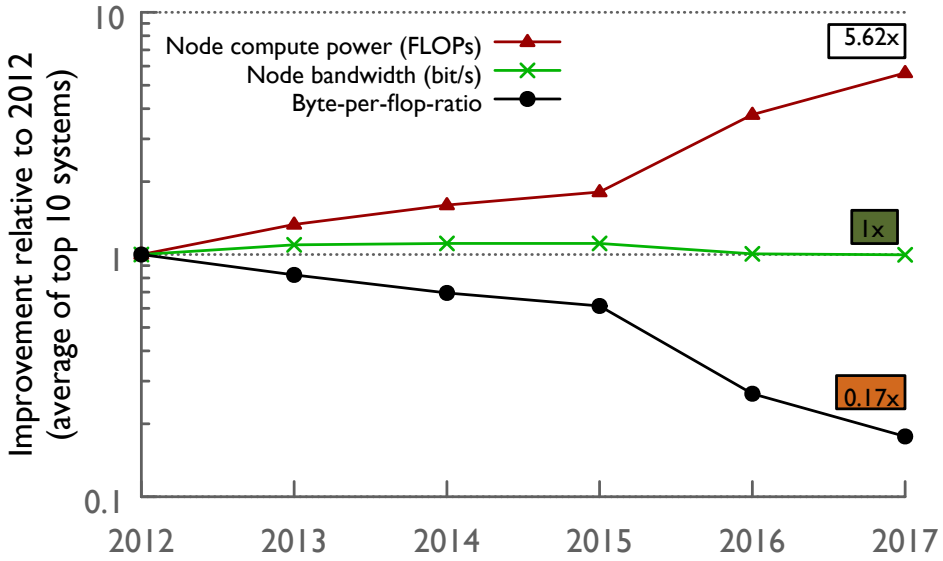
- Optimizations were done in Cray MPI to improve pt2pt and collective latency on KNL
- Enhancements in Cray MPI to enable users best utilize the MCDRAM technology on KNL
- Using Hugepages on MCDRAM can improve large message communication performance
- MPI-only works quite well on KNL; Threading can be helpful, but unless SPMD with “thread-hot” MPI is used scaling to more than 2-8 threads not recommended
- Necessary to use -r1 (core spec) to reduce performance variability due to OS noise
- Owing to network design choices (cost vs. performance tradeoffs), Dragonfly is prone to congestion compared to a Torus network – useful to be aware of.

References:

- Cray XC series Network: *https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf*
- Cray MPI for KNL: *https://www.alcf.anl.gov/files/ANL_MPI_on_KNL.pdf*
- *Low-overhead MPI profiling tool (Autoperf):* *https://www.alcf.anl.gov/user-guides/automatic-performance-collection-autoperf*
- *Variability:* *https://dl.acm.org/citation.cfm?id=3126908.3126926*

QUESTIONS?

EVOLUTION OF TOP 10 SYSTEMS IN TOP500 LIST



Evolution of top 10 systems average node computer power, node bandwidth, and resulting byte/flop ratio (normalized to top500-June rankings).

Node network injection bandwidth is staying relatively constant.

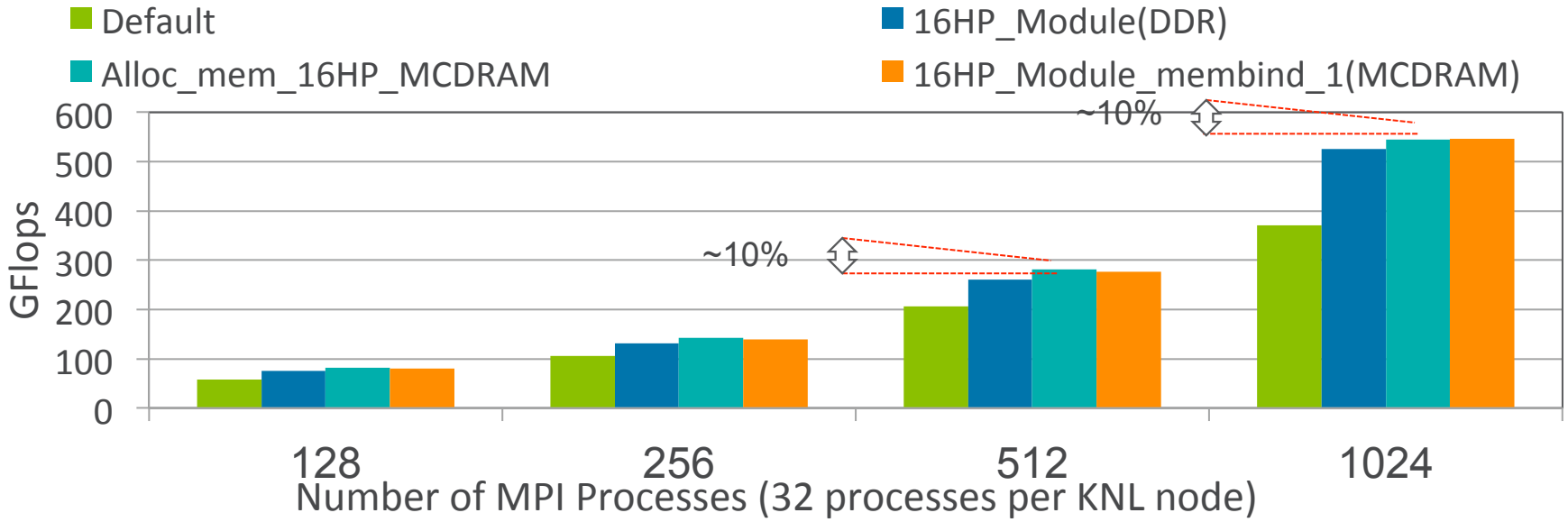
Byte/flop aspect ratio is decreasing.

Need for better network designs, ways to optimize the use of the network (job scheduling policies, routing schemes, topology specific optimizations)

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

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

MPI-3 DYNAMIC PROCESS MANAGEMENT (DPM) SUPPORT

- Cray MPT supports DPM.
- Available in the non-default DPM library

  "-craympich-dpm" linker flag to use this feature.
- Users are recommended to set MPICH_DPM_DIR (if home dir not mounted)
- Only ALPS is supported right now. Server/Client should be launched by the same user.
- Cray MPT June '18 release will have full DRC Support (SLURM)

  Users need to set: PMI_USE_DRC = 1
- Cray MPT also supports MPIX_Comm_rankpool() (Feedback?)
- Verifying the library version:

  (MPICH_VERSION_DISPLAY=1)

  BUILD INFO : Built Date Time Year (git hash commit) MT-G DPM

MPI-3 DYNAMIC PROCESS MANAGEMENT (DPM) SUPPORT *WITHOUT* DRC

Argonne NATIONAL LABORATORY

1. DPM.sh

```
#!/bin/bash
dpmdir="./dpmdir"
cc -craympich-dpm -o
      accept accept.c
cc -craympich-dpm -o
      connect connect.c
mkdir -p $dpmdir
apmgr pdomain –c
my_pdomain

qsub dpm_accept.sh
sleep 5
qsub dpm_connect.sh
apmgr pdomain –r
my_pdomain
```

2. **dpm_accept.sh**

```
#!/bin/bash
#PBS -l nodes=2:ppn=1
#PBS -l walltime=00:05:00
#PBS -j oe
#PBS -N dpm
#PBS -V

cd $PBS_O_WORKDIR

export MPICH_DPM_DIR="`pwd`/
dpmdir"

aprun -p my_pdomain -n 1 -N 1 ./
accept
```

3. **dpm_connect.sh**

```
#!/bin/bash
#
#PBS -l nodes=2:ppn=1
#PBS -l walltime=00:05:00
#PBS -j oe
#PBS -N dpm
#PBS -V

cd $PBS_O_WORKDIR

export MPICH_DPM_DIR="`pwd`/
dpmdir"

aprun -p my_pdomain -n 1 -N 1 ./
connect
```

SUPPORT FOR LARGER MPI TAGS IN CRAY MPI

- In response to MPI_TAG_UB requests from users.

  The DPM feature steals yet another bit from the tag space.

- Cray MPT now offers larger tag spaces and an optimized message matching implementation.
- Available in the non-default *Cray MPICH DPM* library
- MPI_TAG_UB

  Default Cray MPT 2^21

  CrayMPICH-DPM 2^29

- New Message Matching algorithm follows a “Binning” approach to optimize message matching overheads

MPI-RMA THREAD HOT COMMUNICATION IN CRAY MPI

Argonne NATIONAL LABORATORY

- "Thread hot" means high performance thread multiple support

- 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

MULTI-THREADING APPROACHES WITH 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
  - Likely will not offer best performance and thread 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
  - Leads to multiple threads calling 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+OPENMP MODELS

“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
```

“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
  - Less likely to have issues with memory conflicts between threads
  - Potentially simpler to implement
    - 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 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
  - Best performance requires new approach to MPI
    - Goal should be to remove any thread synchronization you can
    - Serializing MPI will limit the benefit and scalability of SPMD
  - SPMD is 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 compute if possible
    - Independent units of communication to be worked on in any order with non-static schedule