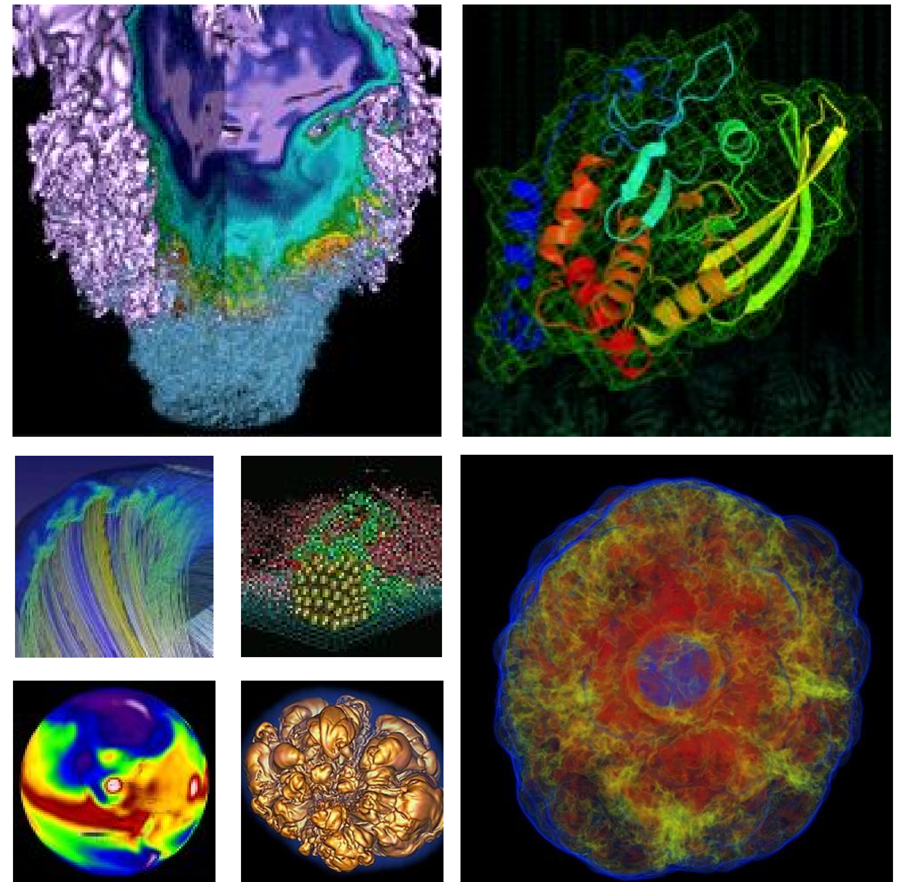
Using Cori KNL Nodes



Helen He, Steve Leak, and Zhengji Zhao

Cori KNL Training, June 9, 2017

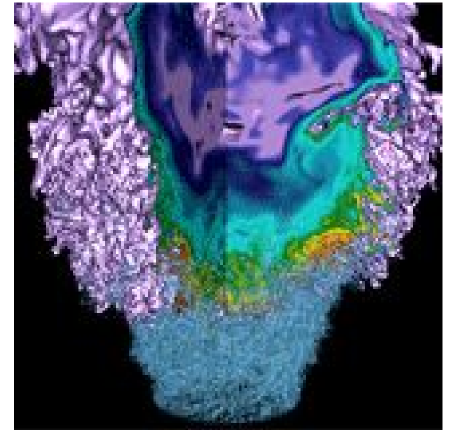
U.S. DEPARTMENT OF ENERGY | Office of Science

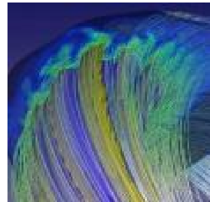
BERKELEY LAB
Lawrence Berkeley National Laboratory

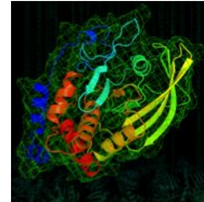
Outline

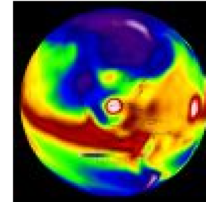
- **How to Compile for KNL?**
  - Some KNL Basics
  - How to compile
  - How to link
  - How to use MCDRAM
- **How to Run on KNL?**
  - Process/thread/memory affinity
  - Available NUMA and MCDRAM modes
  - Script examples
  - Recommendations

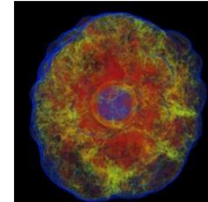
How to Compile for Cori KNL

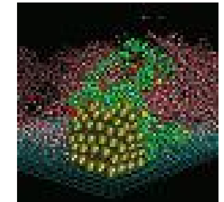












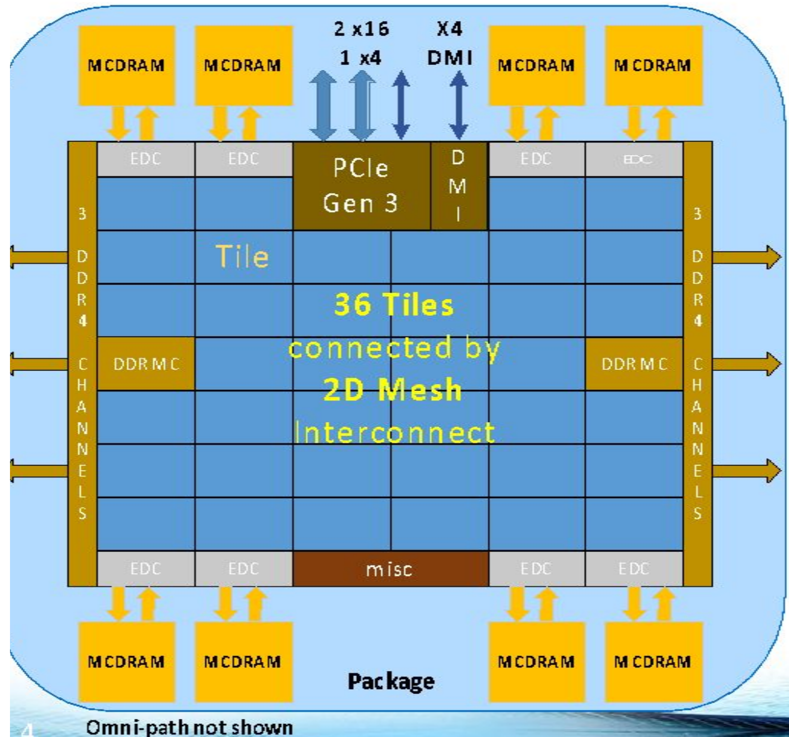
U.S. DEPARTMENT OF ENERGY | Office of Science

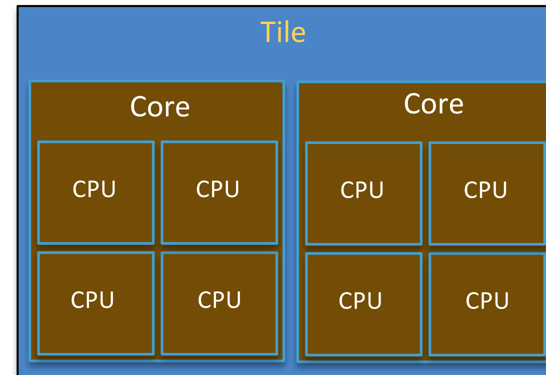
BERKELEY LAB

Cori login and compute nodes

- **Haswell compute nodes and KNL compute nodes.**
- **Compared to Haswell nodes, KNL nodes have**
  – slower and more CPUs per node and smaller memory per CPU
  – longer vector length
  – more complicated memory hierarchy with MCDRAM option
- **All login nodes are Haswell nodes**
- **Binaries built for Haswell can run on KNL nodes, but not vice versa**
  – KNL introduced more AVX-512 instructions that are not recognized on Haswell
  – Needs to further explore threading and vectorization
- **We need to cross-compile**
  – Directly compile on KNL compute nodes is not supported on Cori
  – Meaning to compile on Haswell login nodes to generate binaries for KNL compute nodes

KNL overview and legend





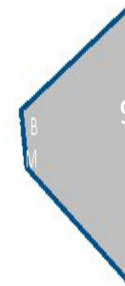
Use Slurm's terminology of cores, CPUs (hardware threads or logical cores).

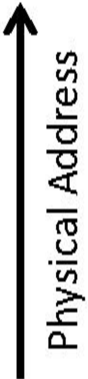
1 Socket/Node
68 Cores (272 CPUs) /Node
36 Tiles/Node (34 active)
2 Cores/Tile; 4 CPUs/Core
1.4 GB/Core DDR memory
235 MB/Core MCDRAM

- **A Cori KNL node has 68 cores/272 CPUs running at 1.4 GHz, 96 GB DDR memory, 16 GB high (~5xDDR) bandwidth on package memory (MCDRAM)**
- **Three cluster modes, all-to-all, quadrant, sub-NUMA clustering, are available at boot time to configure the KNL mesh interconnect.**

KNL overview – MCDRAM modes

Cache Mode | **Flat Mode** | **Hybrid Mode**





Cache Mode:
- No source code changes needed
- Misses are expensive

Flat Mode:
- Code changes required
- Exposed as a NUMA node
- Access via memkind library, job launchers, and/or numactl

Hybrid Mode:
- Combination of the cache and flat modes

MCDRAM can be configured in three different modes at boot time - cache, flat, and hybrid modes

What do compiler wrappers do?

- **The default is PrgEnv-intel. To use another compiler, switch to a corresponding PrgEnv environment, for example:**
  - % module swap PrgEnv-intel PrgEnv-cray
- **Use compiler wrappers (ftn for Fortran, cc for C, and CC for C++).**
- **Wrappers include the architecture specific compiler flags into the compilation/link line automatically**

	Intel	GNU	Cray	Module
Cori KNL	-xMIC-AVX512	-march=knl	-h cpu=mic-knl	craype-mic-knl
Cori Haswell	-xCORE-AVX2	-march=core-avx2	-h cpu=haswell	craype-haswell

-
  - Use the pkg-config tools to dynamically detect paths and libs from the environment (loaded cray modules and some NERSC modules)

How to compile (1)

- **The default loaded architecture target module is "craype-haswell" on the Haswell login nodes.**
- **This module sets CRAY_CPU_TARGET to haswell**
- **Best recommendation to build for KNL target:**

  % module swap craype-haswell craype-mic-knl

  which will set CRAY_CPU_TARGET to mic-knl

  % cc <options> mycode.c

- **Then the compiler wrappers will take care of adding specific KNL target (such as -h cpu=mic-knl for CCE, or -X MIC-AVX512 for Intel compilers), and it will also link the corresponding KNL libraries from the modules loaded (such as cray-libsci)**

How to compile (2)

NERSC 40 YEARS at the FOREFRONT

- **Alternate: add MIC-AVX512 in target option**

```
% cc -axMIC-AVX512,CORE-AVX2 <options> mycode.c
```

- **Only valid when using Intel compilers (cc, CC or ftn)**
- **-ax<arch> adds an "alternate execution paths" optimized for different architectures**
  - Makes 2 (or more) versions of code in same object file
- **NOT AS GOOD as the craype-mic-knl module**
  - craype-haswell module will still cause the Haswell versions of libraries being used, *e.g.* MKL

What to link (1)

Utility libraries

- **Not performance-critical (by definition)**
  - KNL can run Xeon binaries .. can use Haswell-targeted versions
- **I/O libraries (HDF5, NetCDF, etc) should fit in this category too**
  - (for Cray-provided libraries, compiler wrappers will use craype-* to select the best build anyway)

What to link (2)

Performance-critical libraries

- **MKL: has KNL-targeted optimizations**
  - Note: need to link with with libmemkind (more soon)
  - Should be invisibly integrated in future version
- **SLEPc, Caffe, Metis, etc:**
  - (soon) has KNL-targeted builds
- **Modulefiles will use craype-{haswell,mic-knl} to find appropriate library**
  - Currently FFTW, LibSci, Petsc, TPSL have separate builds for KNL
- **Key points:**
  - Someone else has already prepared libraries for KNL
  - No need to do-it-yourself
  - Load the right craype- module

What to link (3)

- **NERSC convention:**

```
/usr/common/software/<name>/<version>/<arch>/[<PrgEnv>]
```

- **Eg:**

```
/usr/common/software/petsc/3.7.4/hsw/intel
/usr/common/software/petsc/3.7.4/knl/intel
```

- **KNL subfolder may be a symlink to hsw**
  - Libraries compiled with `-axMIC-AVX512,CORE-AVX2`

- **Modulefiles should *do the right thing*™**
  - Using CRAY_CPU_TARGET, set by craype-{haswell,mic-knl}

autoconf and cmake solutions

- **In some build systems, certain steps need to run a small test program in order to proceed, *e.g.,* to generate a Makefile.**
  - It will fail with cross-compilation
- **Workaround for autoconf**

```
% module load craype-haswell
% ./configure CC=cc FTN=ftn CXX=CC ...
% module swap craype-haswell craype-mic-knl
% make
```

- **Solution for cmake (from version 3.5.0)**

```
% export CRAYOS_VERSION=6
% cmake -DCMAKE_SYSTEM_NAME=CrayLinuxEnvironment …
```

MCDRAM in a nutshell

- **16GB on-chip memory**
  - cf 96GB off-chip DDR (Cori)
- **Not (exactly) a cache**
  - Latency similar to DDR
- **But very high bandwidth**
  - ~5x DDR
- **2 ways to use it:**
  - “Cache” mode: invisible to OS, memory pages are cached in MCDRAM (cache-line granularity)
  - “Flat” mode: appears to OS as separate NUMA node, with no local CPUs.
    - Accessible via numactl, libnuma, srun --mem_bind

How to use MCDRAM (1)

- **Option 1: Let the system figure it out**
  - Cache mode, no changes to code, build procedure or run procedure
  - Most of the benefit, free, most of the time

How to use MCDRAM (2)

- **Option 2: Run-time settings only**
  - Flat mode, no changes to code or build procedure
  - Does whole job fit within 16GB/node?
    - srun <options> numactl -m 1 ./a.out
    - Or
    - srun <option> --mem_bind=map_mem:1 ./a.out
  - Too big?
    - srun <options> numactl -p 1 ./a.out
    - Or
    - srun <option> --mem_bind=preferred,map_mem:1 ./a.out

Option 3: Use libmemkind

- **Option 3: Use libmemkind**
  - Flat mode
  - Use libmemkind to explicitly allocate selected arrays in MCDRAM
- **C/C++ hbw_malloc() replaces malloc()**

```
#include <hbwmalloc.h>
// malloc(size) -> hbw_malloc(size)
```

- **Fortran**

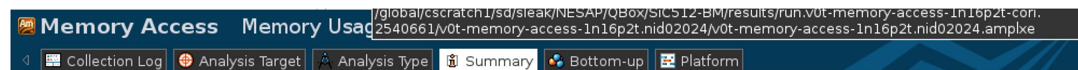
```
!DIR$ MEMORY(bandwidth) a,b,c                    ! cray
real, allocatable :: a(:,:), b(:,:), c(:)
!DIR$ ATTRIBUTES FASTMEM :: a,b,c                ! intel
```

- **Caveat: only for dynamically-allocated arrays**
  - Not local (stack) variables
  - Or Fortran pointers

Which arrays to put in MCDRAM?

- **Vtune memory-access measurements:**

  % amplxe-cl -collect memory-access …



System Bandwidth

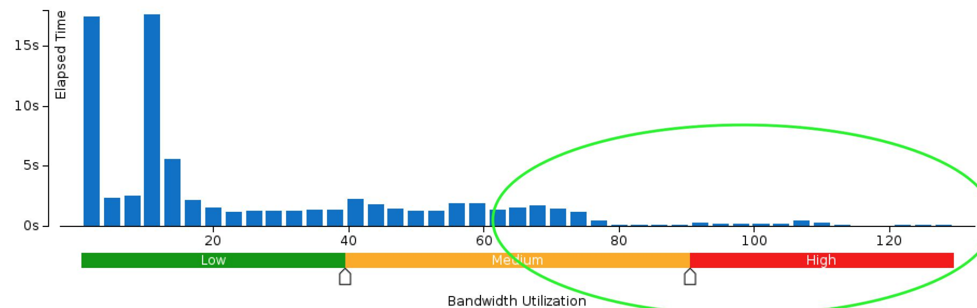
This section provides various system bandwidth-related properties detected by the product. These values are used to define default High, Medium and Low bandwidth utilization thresholds for the Bandwidth Utilization Histogram and to scale overtime bandwidth graphs in the Bottom-up view.

Max DRAM System Bandwidth: 128 GB
Max DRAM Single-Package Bandwidth: 64 GB

Bandwidth Utilization Histogram

This histogram displays a percentage of the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.

Bandwidth Domain: DRAM, GB/sec



Building with libmemkind

- `% module load cray-memkind`
  Compiler wrappers will add
  `-dynamic -lmemkind -lnuma`
- **The binary is built dynamically by default**

Or use NERSC built module:

- `% module load memkind`
  Compiler wrappers will add
  `-lmemkind -ljemalloc -lnuma`
- **The binary is built statically by default**

AutoHBW: Automatic memkind

- Uses array size to determine whether an array should be allocated to MCDRAM
- No code changes necessary!

```
% module load autohbw
```

- Link with -lautohbw

Runtime environment variables:

```
export AUTO_HBW_SIZE=4K     # any allocation
            # >4KB will be placed in MCDRAM
export AUTO_HBW_SIZE=4K:8K # allocations
            # between 4KB and 8KB will
            # be placed in MCDRAM
```

Summary: How to compile

In Summary:

- **Build on login nodes (like you do now)**
- **Use provided libraries (like you probably do now)**
- **Here's the new bit:**

- `module swap craype-haswell craype-mic-knl`
  - For KNL-specific executables

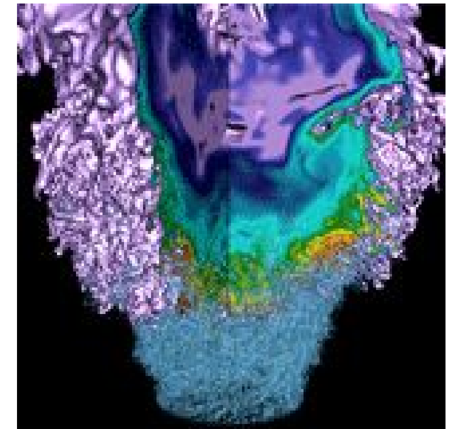
or

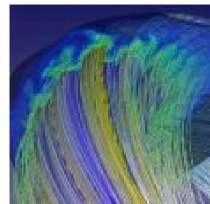
- `CC -axMIC-AVX512,CORE-AVX2 ...`
  - For Haswell/KNL portability (Intel compiler only)

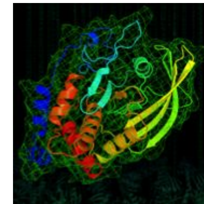
And:

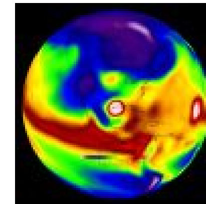
- **Think about MCDRAM**
  - --mem_bind, numactl, memkind, autohbm

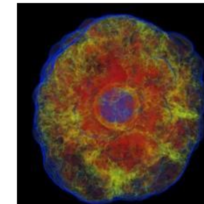
Running jobs on Cori KNL

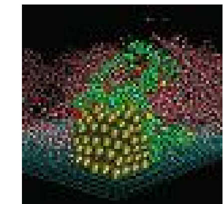












U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Process / Thread / Memory Affinity

- **Correct process, thread and memory affinity is the basis for getting optimal performance on KNL. It is also essential for guiding further performance optimizations.**
  - Process Affinity: bind MPI tasks to CPUs
  - Thread Affinity: bind threads to CPUs allocated to its MPI process
  - Memory Affinity: allocate memory from specific NUMA domains
- **Our goal is to promote OpenMP4 standard settings for portability. For example, OMP_PROC_BIND and OMP_PLACES are preferred to Intel specific KMP_AFFINITY and KMP_PLACE_THREADS settings.**

Process / Thread / Memory Affinity

NERSC 40 YEARS at the FOREFRONT

- **Besides CPU binding, there are also memory affinity concerns.**
  - Cluster modes (NUMA modes)
    - No SNC: All2All, Hemisphere, Quadrant
    - Sub NUMA domains: SNC4, SNC2
  - Memory modes (MCDRAM modes)
    - No extra NUMA domains: Cache
    - Extra NUMA domains: Flat, Hybrid

“hwloc-ls” provides hardware locality info

On a quad,cache node:

% hwloc-ls (the wider the screen, the better)

2 physical cores shares a tile

lstopo

Machine (94GB total)

NUMANode P#0 (94GB)

Package P#0

MemorySideCache (16GB)

L2 (1024KB)		L2 (1024KB)		L2 (1024KB)		L2 (1024KB)
L1d (32KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)
L1i (32KB)	L1i (32KB)	L1i (32KB)	L1i (32KB)	L1i (32KB)	L1i (32KB)	L1i (32KB)
Core P#0	Core P#1	Core P#2	Core P#3	Core P#6	Core P#7	Core P#8
PU P#0, PU P#68, PU P#136, PU P#204	PU P#1, PU P#69, PU P#137, PU P#205	PU P#2, PU P#70, PU P#138, PU P#206	PU P#3, PU P#71, PU P#139, PU P#207	PU P#4, PU P#72, PU P#140, PU P#208	PU P#5, PU P#73, PU P#141, PU P#209	PU P#6, PU P#142

Host: nid02305

Indexes: physical

Date: Thu Jun 8 20:57:41 2017

“numactl -H” displays NUMA domain info (1)

68-core Quad Cache node:

NUMA Domain 0: all 68 cores (272 logic cores)

```
yunhe@cori01:> salloc -N 1 --qos=interactive -C knl,quad,cache -t 30:00
salloc: Granted job allocation 5291739

yunhe@nid02305:> numactl -H
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246
247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 0 size: 96762 MB
node 0 free: 93067 MB
node distances:
node   0
  0:  10
```

- The quad,cache mode has only 1 NUMA node with all CPUs on the NUMA node 0 (DDR memory)
- The MCDRAM is hidden from the numactl –H command (it is a cache).

“numactl -H” displays NUMA domain info (2)

68-core Quad Flat node:

NUMA Domain 0: all 68 cores (272 logic cores)

NUMA Domain 1: HBM with no CPUs

```
yunhe@nid00034:~> numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246
247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 0 size: 96723 MB
node 0 free: 93924 MB
node 1 cpus:
node 1 size: 16157 MB
node 1 free: 16088 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

- The quad,flat mode has only 2 NUMA nodes with all CPUs on the NUMA node 0 (DDR memory).
- And NUMA node 1 has MCDRAM.

"numactl -H" displays NUMA domain info (3)

68-core SNC2 Flat node:

NUMA Domain 0: all 68 cores (272 logic cores)

NUMA Domain 1: HBM with no CPUs

yunhe@nid00034:~> numactl -H
numactl -H
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
node 0 size: 48293 MB
node 0 free: 46047 MB
node 1 cpus: 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 1 size: 48466 MB
node 1 free: 44949 MB
node 2 cpus:
node 2 size: 8079 MB
node 2 free: 7983 MB
node 3 cpus:
node 3 size: 8077 MB
node 3 free: 7980 MB

node distances:

node	0	1	2	3
0:	10	21	31	41
1:	21	10	41	31
2:	31	41	10	41
3:	41	31	41	10

- snc2,flat node has 4 NUMA domains with all CPUs on NUMA nodes 0 and 1 (DDR memory).
- NUMA nodes 2 and 3 have MCDRAM.

Can we just do a naïve srun?

Example: 16 MPI tasks x 8 OpenMP threads per task on a single 68-core KNL quad,cache node:

% export OMP_NUM_THREADS=8
% export OMP_PROC_BIND=spread (other choice are "close","master","true","false")
% export OMP_PLACES=threads (other choices are: cores, sockets, and various ways to specify explicit lists, etc.)
% srun -n 16 ./xthi |sort -k4n,6n

```
Hello from rank 0, thread 0, on nid02304. (core affinity = 0)
Hello from rank 0, thread 1, on nid02304. (core affinity = 144)    (in physical core 8)
Hello from rank 0, thread 2, on nid02304. (core affinity = 17)
Hello from rank 0, thread 3, on nid02304. (core affinity = 161)    (in physical core 25)
Hello from rank 0, thread 4, on nid02304. (core affinity = 34)
Hello from rank 0, thread 5, on nid02304. (core affinity = 178)    (in physical core 42)
Hello from rank 0, thread 6, on nid02304. (core affinity = 51)
Hello from rank 0, thread 7, on nid02304. (core affinity = 195)    (in physical core 59)
Hello from rank 1, thread 0, on nid02304. (core affinity = 0)
Hello from rank 1, thread 1, on nid02304. (core affinity = 144)
```

It is a mess!

The importance of -c and --cpu_bind options

- **The reason is #MPI tasks is not divisible by 68!**
  - Each MPI task is getting 68x4/#MPI tasks of logical cores as the domain size
  - MPI tasks are crossing tile boundaries
- **Let's set number of logical cores per MPI task (-c) manually by wasting extra 4 cores on purpose, which is 256/#MPI tasks.**
  - Meaning to use 64 cores only on the 68-core KNL node., and spread the logical cores allocated to each MPI task evenly among these 64 cores.
  - % srun -n 16 -c 16 --cpu_bind=cores ./xthi

```
Hello from rank 0, thread 0, on nid09244. (core affinity = 0)
Hello from rank 0, thread 1, on nid09244. (core affinity = 136)
Hello from rank 0, thread 2, on nid09244. (core affinity = 1)
Hello from rank 0, thread 3, on nid09244. (core affinity = 137)
```

U.S. DEPARTMENT OF ENERGY | Office of Science

Now it looks good!

Process/thread affinity are good! (Marked first 6 and last MPI tasks only)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153
204																220	221
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
86																102	103
154	155	156	157	158	159	160	161									170	171
222																238	239
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51		
104																	
172																	
240			....														
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67		
120															135		
188								196	197	198	199	200	201	202	203		
256															271		

And so on for other MPI tasks and threads

Essential runtime settings for process/thread affinity

- **Use srun -c and --cpu_bind flags to bind tasks to CPUs**
  - **-c <n> (or --cpus-per-task=n) allocates (reserves) n CPUs per task (process). It helps to evenly spread MPI tasks**
  - Use **--cpu_bind=cores** (no hyperthreads) or **--cpu_bind=threads** (if hyperthreads are used)
- **Use OpenMP envs, OMP_PROC_BIND and OMP_PLACES to fine pin each thread to a subset of CPUs allocated to the host task**
  - Different compilers may have different default values for them.
  - The following provide compatible thread affinity among Intel, GNU and Cray compilers:

OMP_PROC_BIND=true # Specifying threads may not be moved between CPUs

OMP_PLACES=threads # Specifying a thread should be placed in a single CPU

Essential runtime settings for MCDRAM memory affinity

- **In cache mode, no special setting is needed to use MCDRAM**
- **In flat mode, using quad,flat as an example: NUMA node 1 is MCDRAM**
- **#SBATCH -C knl,quad,flat**
  - Enforced memory mapping to MCDRAM
  - If using >16 GB, malloc will fail

```
srun -n 16 -c 16 -cpu_bind=cores --mem_bind=map_mem:1 ./a.out
Or:
srun -n 16 -c 16 -cpu_bind=cores … numactl -m 1 ./a.out
```

  - Preferred memory mapping to MCDRAM:
  - If using >16 GB, malloc will spill to DDR

```
srun -n 16 -c 16 -cpu_bind-=cores
                 --mem_bind=preferred,map_mem:1 ./a.out
Or:
srun -n 16 -c 16 -cou_bind=cores numactl -p 1 ./a.out
```

Request KNL NUMA/MCDRAM modes

- **Use the -C knl,<NUMA>,<MCDRAM> options to request KNL nodes with desired features**
  - #SBATCH -C knl,quad,flat
- **Cori supports combination of the following NUMA/MCDRAM modes:**
  - AllowNUMA=a2a,snc2,snc4,hemi,quad
  - AllowMCDRAM=cache,split,equal,flat
  - So you can request any combinations, such as "-C knl,snc2,split"
  - quad,cache is the default for now
  - Jobs requesting other modes have higher chance of needing to reboot nodes before run, which takes ~40 min.
  - Job will be in CF state (nodes allocated, job won't start) when nodes are getting rebooted.

Sample job script to run under the quad,cache mode

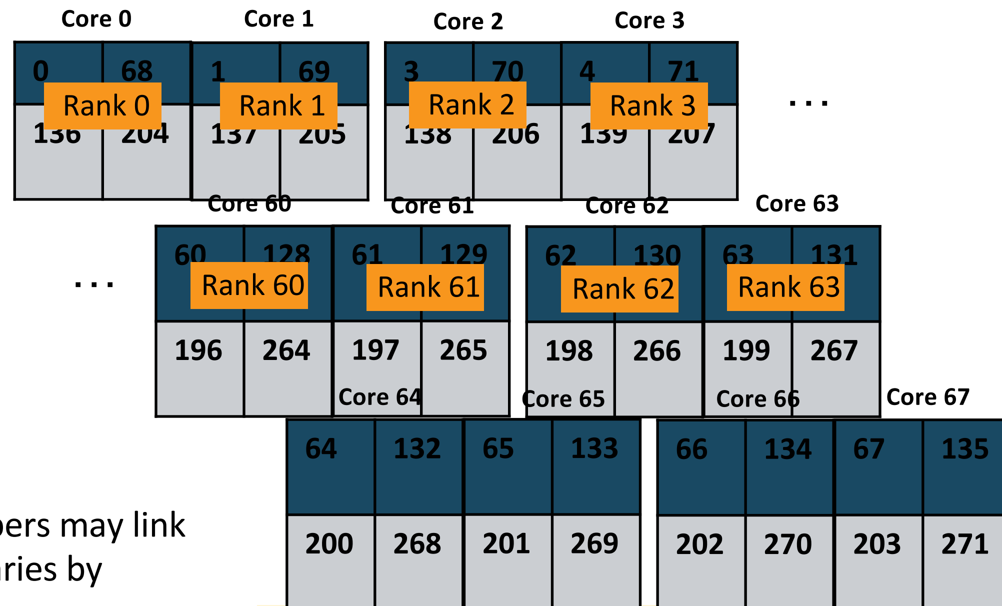
Sample Job script (Pure MPI)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -t 1:00:00
#SBATCH -L SCRATCH
#SBATCH -S 2
#SBATCH -C knl,quad,cache

export OMP_NUM_THREADS=1      # optional*
srun -n64 -c4 --cpu_bind=cores ./a.out
```

* To avoid unexpected thread forking (compiler wrappers may link your code to the multi-threaded system provided libraries by default).

This job script requests 1 KNL node in the quad,cache mode. The srun command launches 64 MPI tasks on the node, allocating 4 CPUs per task, and binds processes to cores. The Rank 0 will be pinned to Core0, Rank1 to Core1, …, Rank63 will be pinned to Core63. Each MPI task may move within the 4 CPUs in the cores.

Process affinity outcome



Each 2x2 box above is a core with 4 CPUs (hardware threads). The numbers shown in each box is the CPU ids. The last 4 cores (64-67) are not used in this example.

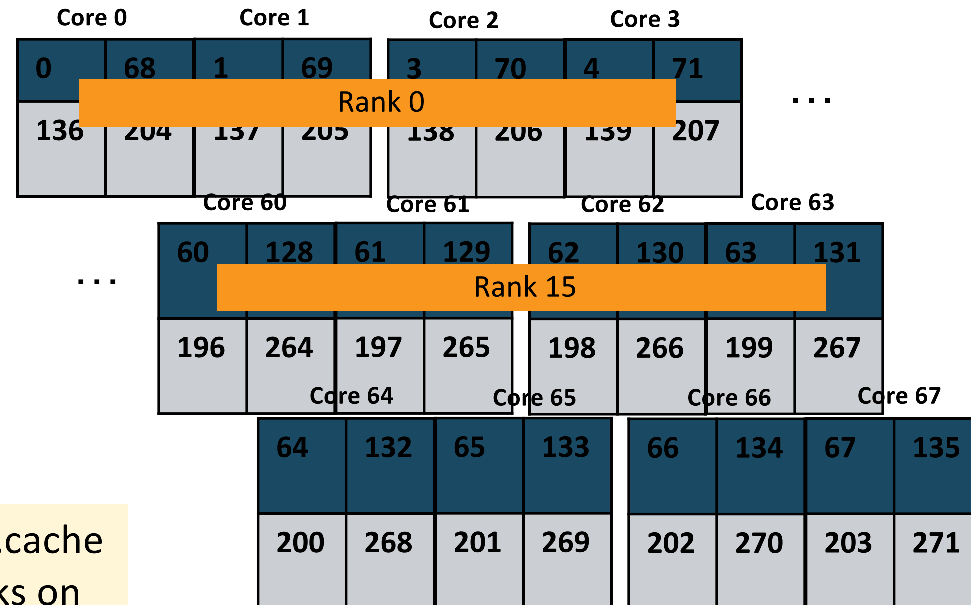
Sample job script to run under the quad,cache mode

Sample Job script (Pure MPI)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -t 1:00:00
#SBATCH -L SCRATCH
#SBATCH -S 2
#SBATCH -C knl,quad,cache

export OMP_NUM_THREADS=1      # optional
srun -n16 -c16 --cpu_bind=cores ./a.out
```

This job script requests 1 KNL node in the quad,cache mode. The srun command launches 16 MPI tasks on the node, allocating 16 CPUs per task, and binds each process to 4 cores/16 CPUs. The Rank 0 is pinned to Core 0-3, and Rank 1 to Core 4-7, …, Rank 15 to Core 60-63. The MPI task may move within the 16 CPUs in the 4 cores.

Process affinity outcome



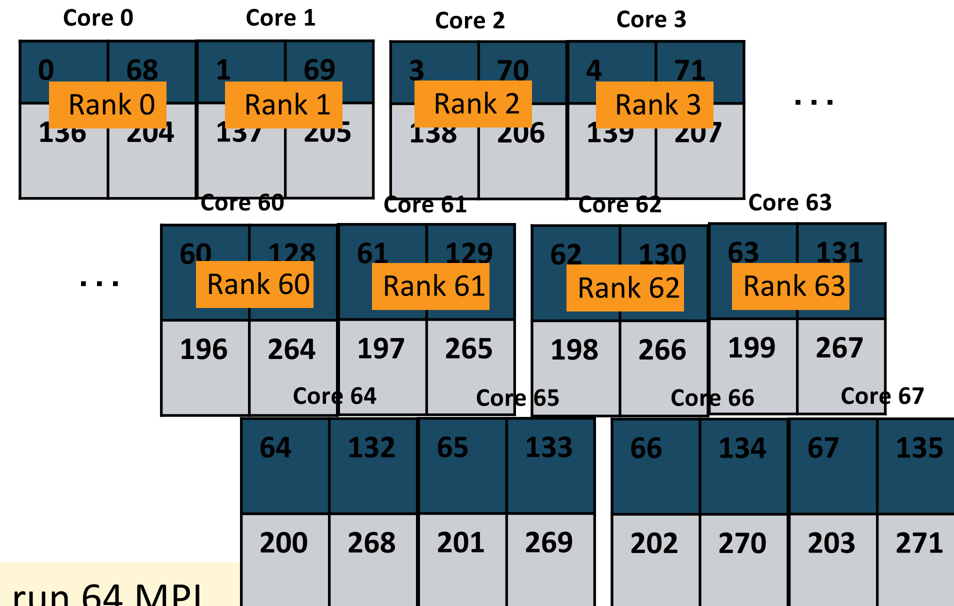
Sample job script to run under the quad,cache mode

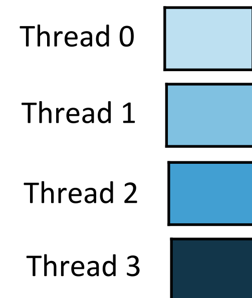
Sample Job script (MPI+OpenMP)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -t 1:00:00
#SBATCH -L SCRATCH
#SBATCH -S 2
#SBATCH -C knl,quad,cache

export OMP_NUM_THREADS=4
srun -n64 -c4 --cpu_bind=cores ./a.out
```

This job script requests 1 KNL quad, cache node to run 64 MPI tasks, allocating 4 CPUs per task, and binds each task to the 4 CPUs allocated within the cores.
Each MPI task runs 4 OpenMP threads. The Rank 0 will be pinned to Core 0, Rank 1 to Core 1, …, Rank 63 to Core 63. The 4 threads of each task are pinned within the core.
Depending on the compilers used to compile the code, the 4 threads in each core may or may not move between the 4 CPUs.

Process and thread affinity





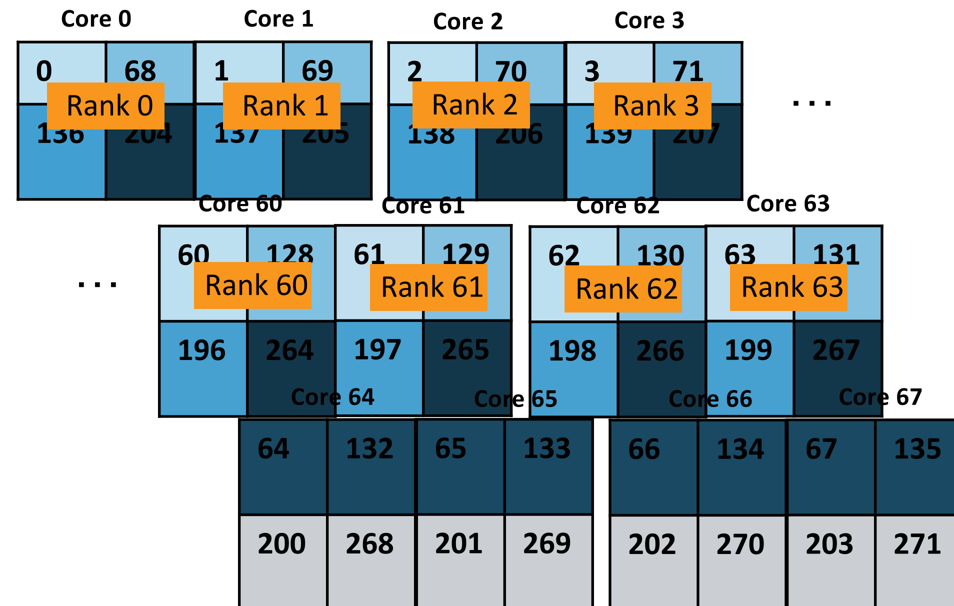
Sample job script to run under the quad,cache mode

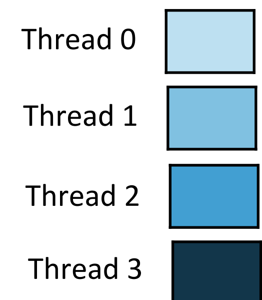
Sample Job script (MPI+OpenMP)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -t 1:00:00
#SBATCH -L SCRATCH
#SBATCH -S 2
#SBATCH -C knl,quad,cache

export OMP_PROC_BIND=true
export OMP_PLACES=threads
export OMP_NUM_THREADS=4
srun -n64 -c4 --cpu_bind=cores ./a.out
```

With the above two OpenMP envs, each thread is pinned to a single CPU within each core. The resulting thread affinity (and task affinity) is shown in the right figure. E.g., for Rank 0, Thread 0 is pinned to CPU 0, Thread 1 to CPU 68, Thread 2 to CPU 136, and Thread 3 is pinned to CPU 204.

Process/thread affinity outcome





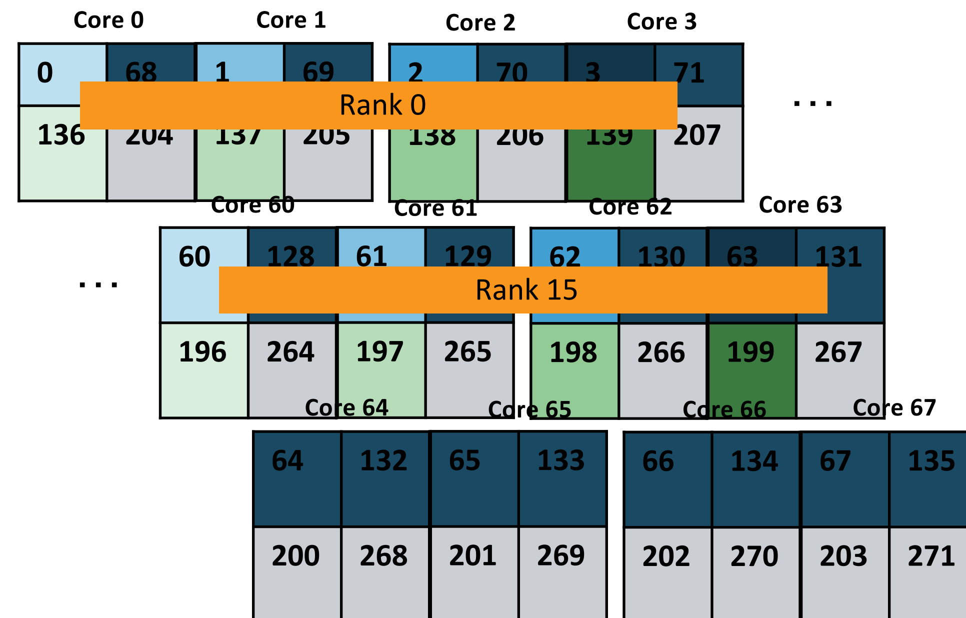
Sample job script to run under the quad,cache mode

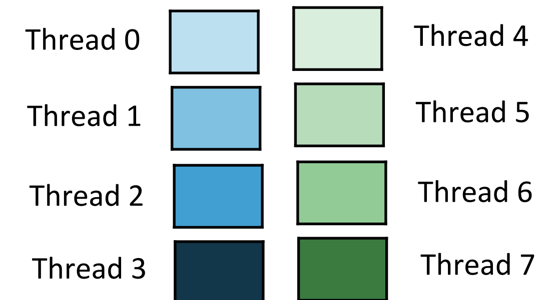
Sample Job script (MPI+OpenMP)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -t 1:00:00
#SBATCH -S 2
#SBATCH -C knl,quad,cache

export OMP_PROC_BIND=true
export OMP_PLACES=threads
export OMP_NUM_THREADS=8
srun -n16 -c16 --cpu_bind=cores ./a.out
```

Each MPI task is allocated 16 CPUs, i.e., 4 physical cores. Each thread is pinned to a single CPU on the cores allocated to the task. E.g., for Rank 0, Thread 0 is pinned to the CPU 0 (on Core 0), Thread 1 to the CPU 1 (on Core1), Threads 2 to CPU 2 (on Core 2), and so on.

Process/thread affinity outcome



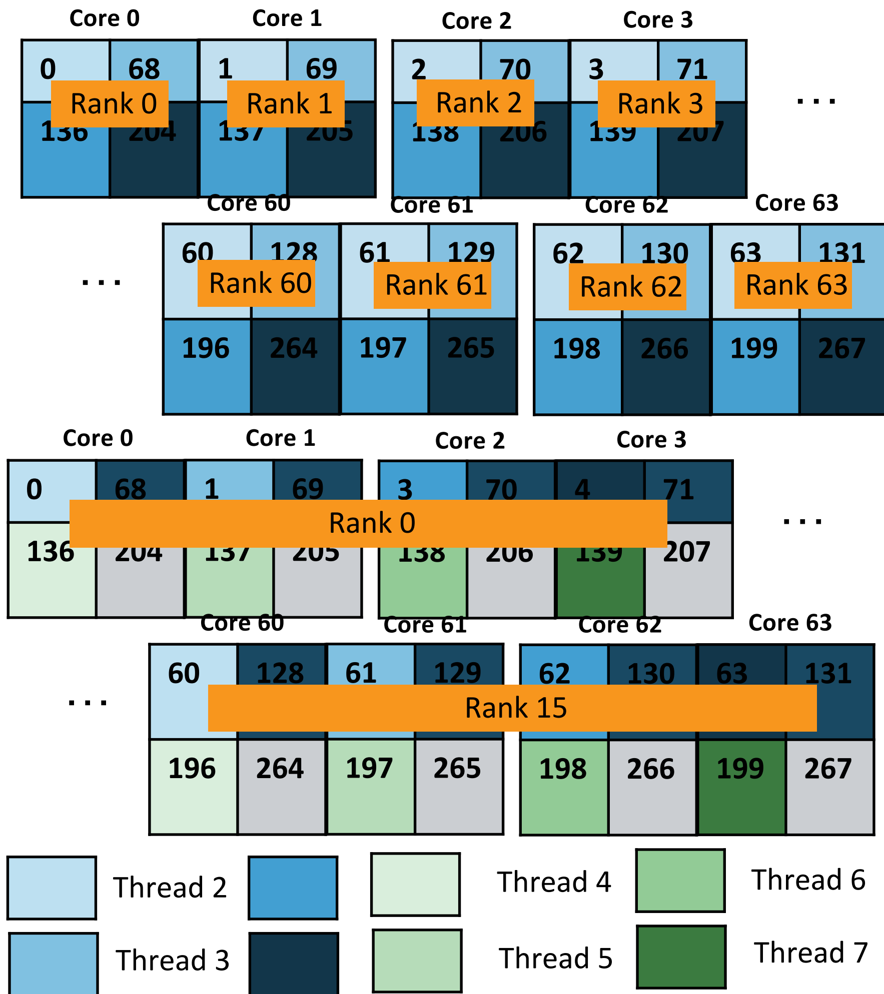


Sample job script to run under the quad,flat mode

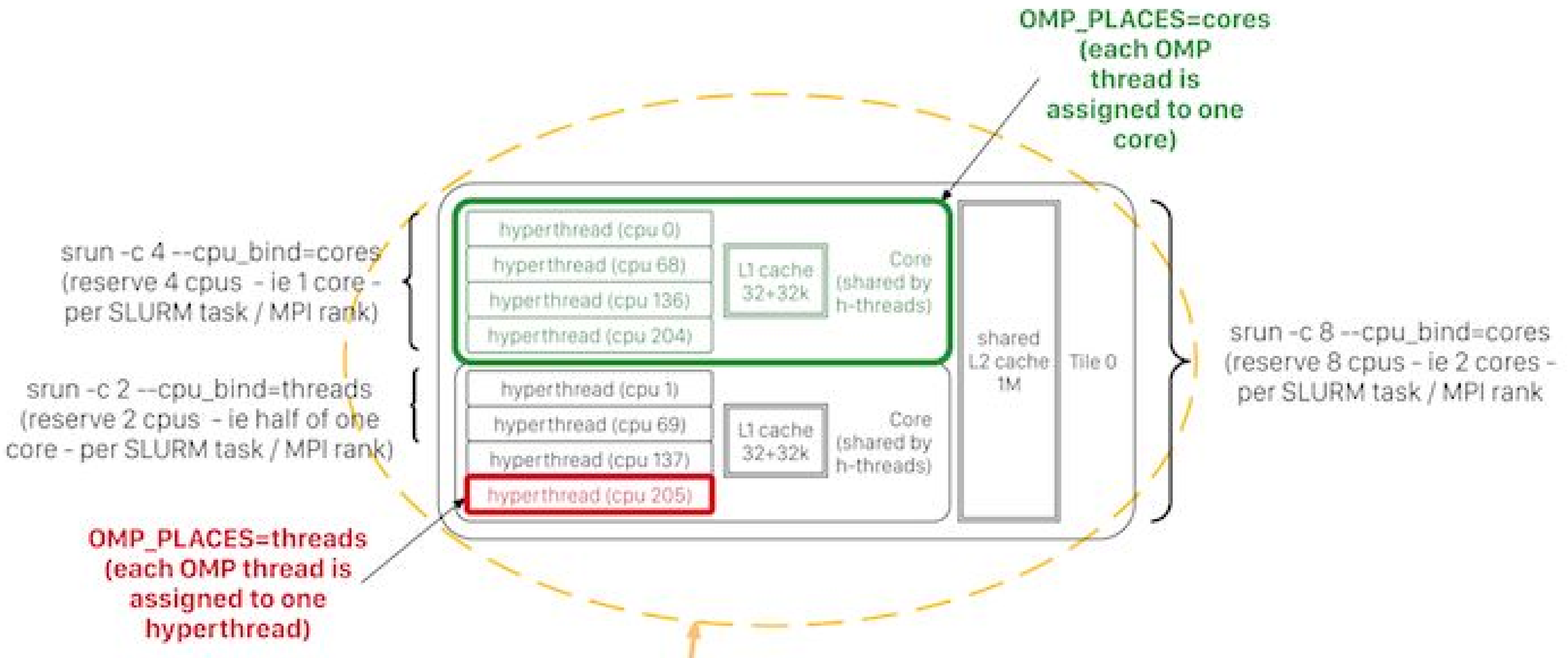
Sample Job script (MPI+OpenMP)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -t 1:00:00
#SBATCH -S 2
#SBATCH -C knl,quad,flat

export OMP_PROC_BIND=true
export OMP_PLACES=threads

export OMP_NUM_THREADS=4
srun -n64 -c4 --cpu_bind=cores ./a.out

export OMP_NUM_THREADS=8
srun -n16 -c16 --cpu_bind=cores ./a.out
```

Process/thread affinity outcome



Illustrations of cpu_bind and OMP_PLACES



How to Run Debug and Interactive Jobs

- **Debug**
  - Max 512 nodes, 30 min, run limit 1, queue limit 5

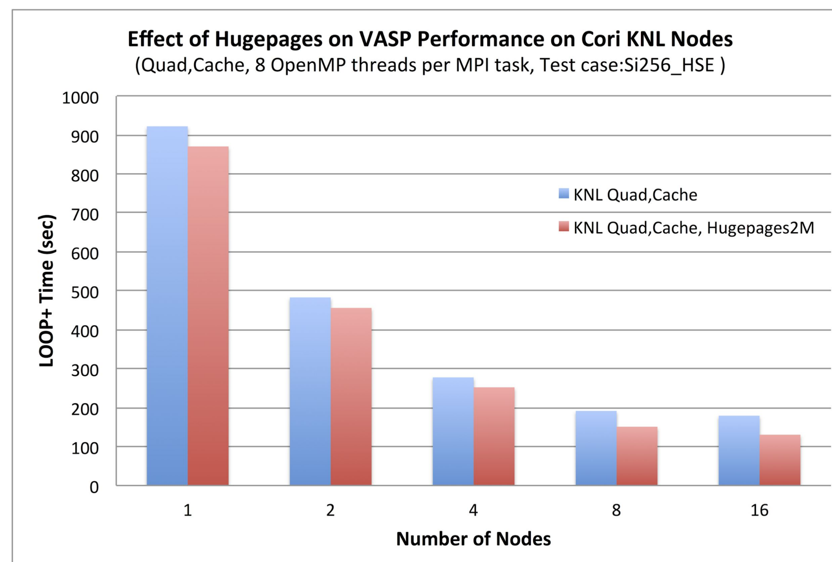
  % salloc -N 20 -p debug -C knl,quad,cache -t 30:00

- **interactive**
  - Instant allocation (or reject), run limit 1, max walltime 4 hrs, up to 20 nodes on Cori (Haswell and KNL) per repo

  % salloc -N 2 -qos=interactive -C knl,quad,cache –t 2:00:00

Recommend: using hugepages

- **The default page size is 4K.**
- **General recommend to use hugepages on KNL**
- **% module load craype-hugepages2M**
  - There are many other modules with different hugepage sizes. 4M, 8M, … 512M
  - 2M is genenally helpful already
  - More details in “man intro_hugepages”
- **See benefits with many applications: VASP, MILC, MFDn, etc.**
- **Hugepage availability decreases the longer a node is up. Recommend system admins monitor and reboot from time to time.**



Recommend: Use switches to minimize performance variations

- **Request max count of switches and max wait time in the queue**
- **#SBATCH --switches=<count>[@<max-time>]**
- **For example, requesting 1024 nodes, it needs roughly 3 switches (1 switch for ~384 nodes)**
  - #SBATCH --switches=3@24:00:00
- **Helps to minimize inter-group network communication to improve latency and bandwidth and limit variations for codes with many MPI calls. (especially helpful if only 1 switch is needed).**

Recommend: Use Zone Sort (mostly by default)

- **MCDRAM in cache mode is direct-mapped cache.**
  - Over time, cache conflicts increase to impact performance.
- **Zonesort helps to sort Linux kernel's list of free pages.**
- **By default zonesort will run once per srun (immediately prior to the code starting).**
- **Users can disable it or request to run zonesort frequently during an srun:**

```
sbatch --zonesort=on ... # default, run once per srun
sbatch --zonesort=off ... # do not run zonesort
sbatch --zonesort=<n, positive integer> ...  # run zonesort every n seconds
```

Recommend: Use sbcast (for larger jobs)

- **By default, SLURM does not copy the executable onto each node. Large delay can happen between first and last node starting the job.**
- **Use "sbcast" to broadcast the executable to each compute node prior to job start, especially for jobs >1500 MPI tasks.**

```
% sbcast ./mycode.exe /tmp/mycode.exe
% srun <srun options> numactl <numactl options>
/tmp/mycode.exe

# or in the case of when numactl is not needed:
% srun --bcast=/tmp/mycode.exe <srun options> ./mycode.exe
```

Recommend: Use -S for core specialization

- **Core specialization is a feature designed to isolate system overhead (system interrupts, etc.) to designated cores on a compute node.**
- **#SBATCH -S 4 or #SBATCH -S 2**
- **Use it since we mostly use 64 cores out of 68 anyway**
- **Only works with sbatch, can not be used with salloc, which is already a wrapper script for srun.**

NERSC Job Script Generator

https://my.nersc.gov/script_generator.php

My NERSC

- Sign In
- Dashboard
- Jobs
  - Jobscript Generator
  - Completed Jobs
  - Cori Queues
  - Edison Queues
  - PDSF Queues
  - Queue Backlog
  - Jobs IO Statistics
- Center Status
- File Browser
- My Tickets
- Data Dashboard

Jobscript Generator

Job Information

This tool generates a batch script template which also realizes specific process and thread binding configurations.

Machine

Select the machine on which you want to submit your job.

Cori - KNL

Application Name

Specify your application including the full path.

myapp.x

Job Name

Specify a name for your job.

mytest_KNL

Email Address

Specify your email address to get notified when the job enters a certain state.

```
#!/bin/bash
#SBATCH -N 150
#SBATCH -C knl,quad,flat
#SBATCH -p regular
#SBATCH -J mytest_KNL
#SBATCH -t 03:30:00

#OpenMP settings:
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread


#run the application:
srun -n 2400 -c 16 --cpu_bind=cores numactl -p 1 myapp.x
```

Affinity Verification Methods (1)

- **NERSC has provided pre-built binaries from a Cray code (xthi.c) to display process thread affinity: check-mpi.intel.cori, check-mpi.cray.cori, check-hybrid.intel.cori, etc.**

  % srun -n 32 -c 8 --cpu_bind=cores check-mpi.intel.cori|sort -nk 4

  Hello from rank 0, on nid02305. (core affinity = 0,1,68,69,136,137,204,205)
  Hello from rank 1, on nid02305. (core affinity = 2,3,70,71,138,139,206,207)
  Hello from rank 2, on nid02305. (core affinity = 4,5,72,73,140,141,208,209)
  Hello from rank 3, on nid02305. (core affinity = 6,7,74,75,142,143,210,211)

- **Intel compiler has a run time environment variable KMP_AFFINITY, when set to "verbose”:**

  OMP: Info #242: KMP_AFFINITY: pid 255705 thread 0 bound to OS proc set {55}
  OMP: Info #242: KMP_AFFINITY: pid 255660 thread 1 bound to OS proc set {10,78}
  OMP: Info #242: OMP_PROC_BIND: pid 255660 thread 1 bound to OS proc set {78} …

- **Cray compiler has a similar env CRAY_OMP_CHECK_AFFINITY, when set to "TRUE”:**

  [CCE OMP: host=nid00033 pid=14506 tid=17606 id=1] thread 1 affinity: 90
  [CCE OMP: host=nid00033 pid=14510 tid=17597 id=1] thread 1 affinity: 94 …

Affinity Verification Methods (2)

- **Use srun flag --cpu_bind=verbose to check thread affinity**
  - Need to read the cpu masks in hexidecimal format
- **Use srun flag --mem_bind=verbose,<type> to check memory affinity**
- **Use the numastat -p <PID> command to confirm while a job is running**

A few useful commands (1)

% sinfo --format="%F %b" for features of available nodes (or sinfo –format="%C %b" for CPUs).

- A/I/O/T (allocated/idle/other/total)

```
yunhe@cori02:~> sinfo --format="%F
%b"
NODES(A/I/O/T) ACTIVE_FEATURES
1879/498/11/2388 haswell
14/0/0/14 quad,cache,knl
0/0/7/7 knl
1766/0/0/1766 knl,flat,quad
52/0/0/52 cache,quad
7584/240/22/7846 knl,cache,quad
0/0/3/3 knl,flat,snc2
```

```
yunhe@cori02:~> sinfo --format="%C %b"
CPUS(A/I/O/T) ACTIVE_FEATURES
119520/32608/704/152832 haswell
3808/0/0/3808 quad,cache,knl
0/0/1904/1904 knl
480352/0/0/480352 knl,flat,quad
14144/0/0/14144 cache,quad
2055504/72624/5984/2134112
knl,cache,quad
0/0/816/816 knl,flat,snc2
```

All these are quad,cache KNL nodes

A few useful commands (2)

% scontrol show node <nid>   to see details of a node.

```
yunhe@cori02:~> scontrol show node nid11412
NodeName=nid11412 Arch=x86_64 CoresPerSocket=68
  CPUAlloc=272 CPUErr=0 CPUTot=272 CPULoad=64.01
  AvailableFeatures=knl,flat,split,equal,cache,a2a,snc2,snc4,hemi,quad
  ActiveFeatures=knl,cache,quad
  Gres=craynetwork:4,hbm:0
  NodeAddr=nid11412 NodeHostName=nid11412
  OS=Linux RealMemory=92160 AllocMem=92160 FreeMem=64648 Sockets=1 Boards=1
  State=ALLOCATED ThreadsPerCore=4 TmpDisk=0 Weight=1000 Owner=N/A MCS_label=N/A
  Partitions=debug,regular,regularx,special,realtime,knl,knl_regularx,knl_reboot
  BootTime=2017-06-08T09:54:08 SlurmdStartTime=2017-06-08T10:17:38
  CfgTRES=cpu=272,mem=90G
  AllocTRES=cpu=272,mem=90G
  CapWatts=n/a
  CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
  ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

A few useful commands (3)

% scontrol show job <jobid> for details of a job

```
yunhe@cori02:~> scontrol show job 52967xx
JobId=52967xx JobName=h5write
   UserId=fbench(42034) GroupId=fbench(42034) MCS_label=N/A
   Priority=64824 Nice=0 Account=mpcray QOS=normal_knl_2
   JobState=PENDING Reason=Priority Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:20:00 TimeMin=N/A
   SubmitTime=2017-06-08T22:20:14 EligibleTime=2017-06-08T22:20:14
   StartTime=Unknown EndTime=Unknown Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=knl_reboot AllocNode:Sid=cori02:55443
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null)
   NumNodes=32-32 NumCPUs=32 NumTasks=32 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=32,node=32
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=4
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=knl&quad&flat DelayBoot=00:00:00
   Gres=craynetwork:1 Reservation=(null)
   OverSubscribe=NO Contiguous=0 Licenses=project:1 Network=(null)
   Command=/global/project/projectdirs/mpccc/fbench/H5_HSW_KNL/h5io/knl-write-binding_flat.sh
   WorkDir=/global/project/projectdirs/mpccc/fbench/H5_HSW_KNL/h5io
   StdErr=/global/project/projectdirs/mpccc/fbench/H5_HSW_KNL/h5io/5296744.err
   StdIn=/dev/null
   StdOut=/global/project/projectdirs/mpccc/fbench/H5_HSW_KNL/h5io/5296744.out
   Power=
```

A few useful commands (4)

- **“sacct -u <username> -X” to see all jobs from a user ran/queued in the last 24 hrs.**
- **“sqs -u <username> or “squeue -u <username>” to see all jobs from a user currently running/queued in the system.**
- **See sbatch, srun, squeue, sinfo, sacct, and other Slurm command man pages for more options and usage info.**

Use Burst Buffer for faster IO

- **Cori has 1.8PB of SSD-based "Burst Buffer" to support I/O intensive workloads**
- **Jobs can request a job-temporary BB filesystem, or a persistent (up to a few weeks) reservation**
- **More info at http://www.nersc.gov/users/computational-systems/cori/burst-buffer/**

Queue structures and policies

KNL Nodes

Partition	Nodes	Availability	Physical Cores	Max Walltime per Job	QOS	Per User Limits		
						Max Running Jobs	Max Nodes in Use	Max queued jobs
debug	1-512	All NERSC users	1-34,816	30 min	normal	1	-	5
regular	1-512	All NERSC users	68-79,016	2 hrs	normal	-	-	-
	1-1,162	All NERSC users	68-79,016	24 hrs	normal	-	-	40
	1,163+	All NERSC users	79,017+	12 hrs	normal	-	-	10

- N
  - Use “--qos-debug”, “--qos=regular”, “--qos=premium”, etc. to request nodes instead of “-p debug”, “-p regular” …
  - Max queued jobs limits
  - Job size bucket boundaries to be increased from 1162 to 1187.
  - Watch for announcements!

Cori KNL will start charging from July 1

- NERSC hours charged for a job will be 96 (KNL base charge factor) * #nodes used * actual wall time.
- For example, a 100 node job ran for 2 hrs on KNL will be charged: 96*100*2 = 19,200 NERSC hours.
- As comparison, Edison base charge factor is 48, and Cori Haswell base charge factor is 80.
- Possible queue configuration changes, including priority boost and charging discounts for large jobs.

Summary

- **Use -C knl,<NUMA>,<MCDRAM> to request KNL nodes**
- **Consider using 64 cores out of 68 in most cases**
- **Always explicitly use srun's –c and --cpu_bind flags to spread the MPI tasks evenly over the cores/CPUs on the nodes**
- **Use OpenMP envs, OMP_PROC_BIND and OMP_PLACES to fine pin threads to the CPUs allocated to the MPI tasks**
- **Use srun's --mem_bind or numactl to control memory affinity and access MCDRAM**
  - memkind/autoHBW libraries can be used to allocate only selected arrays/memory allocations to MCDRAM
- **Take advantage of the NERSC Job Script Generator tool**

Further documentations

- **Compiling**
  - http://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/
- **Running Jobs**
  - http://www.nersc.gov/users/computational-systems/cori/running-jobs/running-jobs-on-cori-knl-nodes/