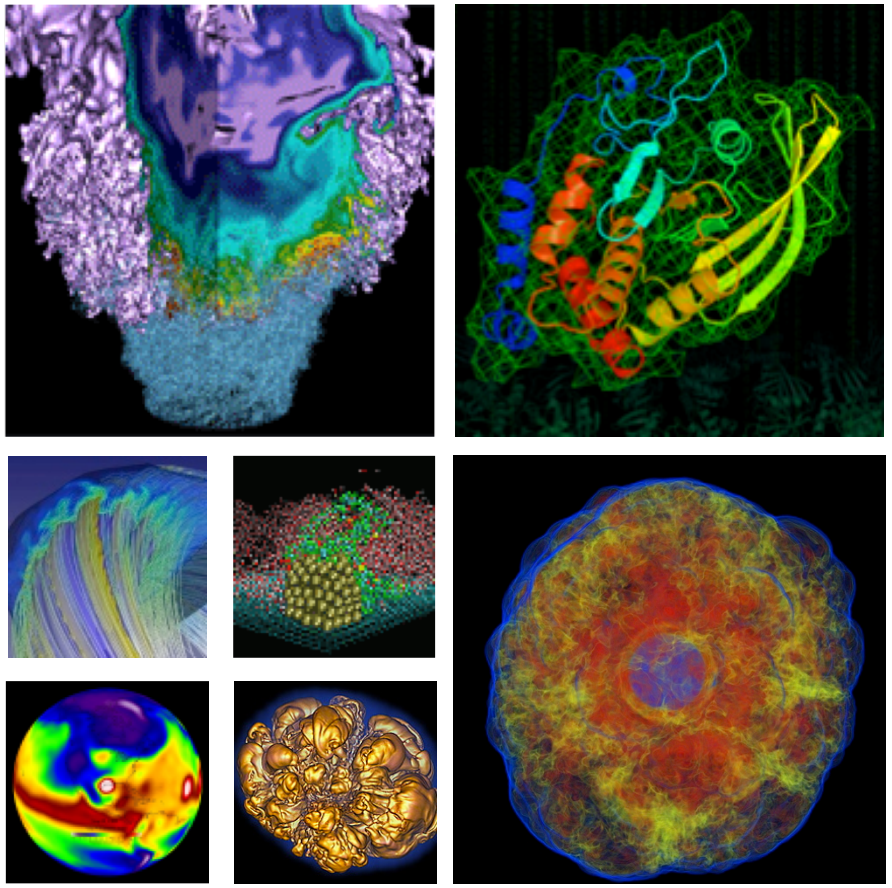
# Using Cori KNL Nodes



Helen He

Cori KNL Hackathon, March 7, 2018

NERSC 40 YEARS at the FOREFRONT 1974-2014

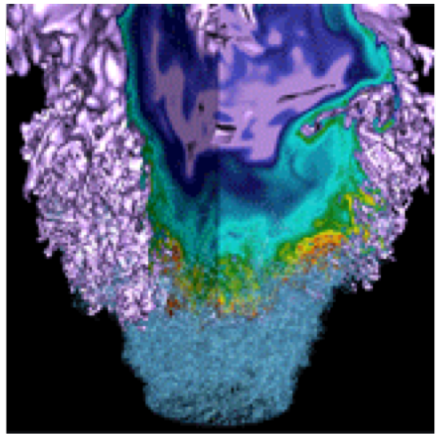
U.S. DEPARTMENT OF ENERGY | Office of Science

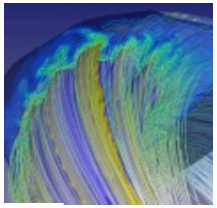
BERKELEY LAB

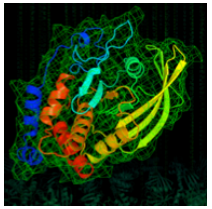
# Outline

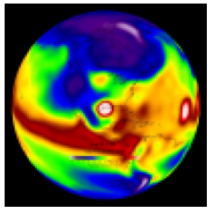
- **How to Compile for KNL?**
  - How to compile and link
  - How to use MCDRAM
- **How to Run on KNL?**
  - Process/thread/memory affinity
  - Script examples
  - Recommendations
- **How to use Performance Tools**
  - Vtune (and APS)
  - Advisor
  - Inspector

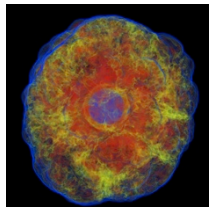
# How to Compile for Cori KNL

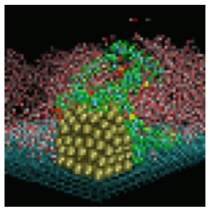












# Cori Login and Compute Nodes

- **Haswell compute nodes and KNL compute nodes**
- **All login nodes are Haswell nodes**
- **Binaries built for Haswell can run on KNL nodes, but not vice versa**
- **We need to cross-compile**
  - Directly compile on KNL compute nodes is very slow
  - Meaning to compile on Haswell login nodes to generate binaries for KNL compute nodes

# How to Compile and Link (1)

- **The default loaded architecture target module is "craype-haswell" on the Haswell login nodes.**
- **This module sets CRAY_CPU_TARGET to haswell**

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

- **Best recommendation to build for KNL target**

  % module swap craype-haswell craype-mic-knl

  **which will set CRAY_CPU_TARGET to mic-knl**

# How to Compile and Link (2)

- Then the compiler wrappers will take care of adding specific KNL target (such as "-h cpu=mic-knl" for CCE, or "-X MIC-AVX512" or "-xKNL" for Intel compilers), and it will also link the optimized KNL libraries, if exist, from the modules loaded (such as cray-libsci, cray-petsc,crsay-tpsl, etc.)
- Some libraries that are not performance-critical on KNL will use the Haswell version (such as cray-netcdf, cray-hdf5, etc.), will also be linked in transparently with the compiler wrappers from the modules loaded.

# How to Compile and Link (3)

- **Alternate: build a fat binary by adding MIC-AVX512 in target option**

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

- **Only valid when using Intel compilers (cc, CC or ftn)**
- **-ax<arch> adds an "alternate execution path" optimized for different architectures**
  - Makes 2 (or more) versions of code in same object file
  - Runtime will choose which version to use
- **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

# 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**

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

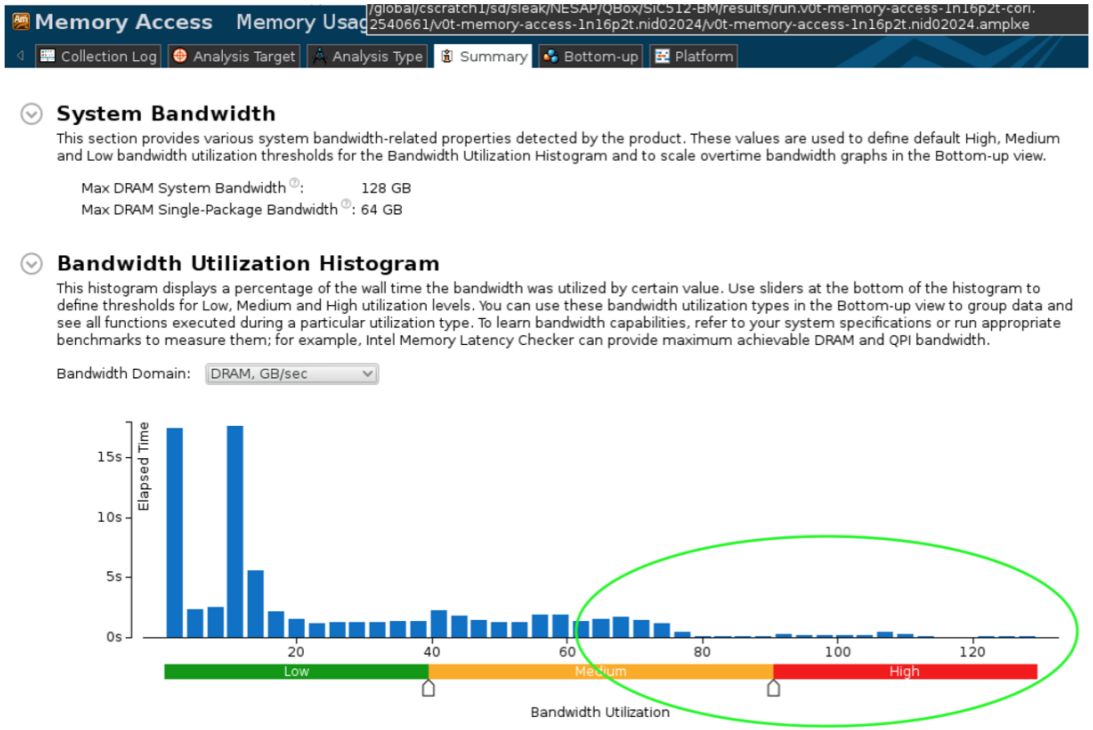
# Using MCDRAM

- **Cori KNL has 16 GB on-chip memory per node**
  - and 96 GB off-chip DDR (Cori)
- **Not (exactly) a cache**
  - Latency similar to DDR
- **But very high bandwidth**
  - ~5x DDR
- **MCDRAM modes**
  - "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.
  - "Hybrid" mode: some memory as cache, some not
  - If not Cache: Accessible via numactl, libnuma, srun --mem_bind, autohbw, libmemkind

# Which Arrays to Put in MCDRAM?

- **VTune memory-access measurements:**

  % amplxe-cl -collect memory-access …



# 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 (need to request a reservation to use this mode)**
- **#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
```

# 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**

# Use libmemkind

- **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

# 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
```

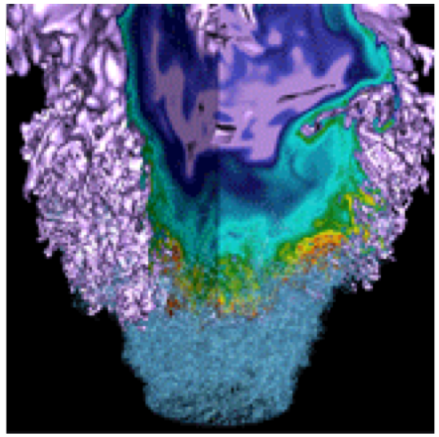
# Summary: How to Compile and Link

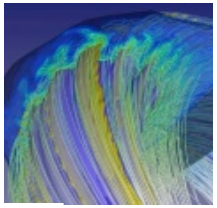
- **Build on login nodes**
- `module swap craype-haswell craype-mic-knl`
  - For KNL-specific executables

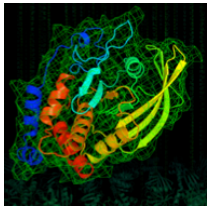
**or**

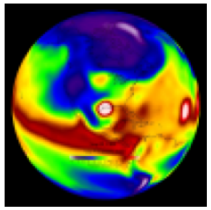
- `CC -axKNL,HSWELL ...`
  - For Haswell/KNL portability
- **Libraries will be automatically linked by compiler wrappers when specific modules are loaded**
- **Think about MCDRAM**
  - --mem_bind, numactl, memkind, autohbm
- **More details at**
  - https://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/#toc-anchor-6

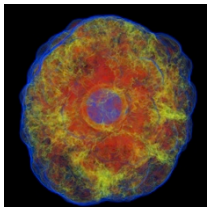
# How to Run Jobs on Cori KNL

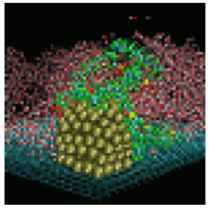












# Cori KNL Example Compute Nodes

- **A Cori KNL node has 68 cores/272 CPUs, 96 GB DDR memory, 16 GB high 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**

**Arrangement of Hardware Threads for 68 Core KNL**

| Core # → | 0 | 1 | 2 | 3 | … | 16 | 17 | 18 | … | 33 | 34 | 35 | … | 50 | 51 | 52 | … | 65 | 66 | 67 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HW Thread # | 0 | 1 | 2 | 3 | … | 16 | 17 | 18 | … | 33 | 34 | 35 | … | 50 | 51 | 52 | … | 65 | 66 | 67 |
| | 68 | 69 | 70 | 71 | … | 84 | 85 | 86 | … | 101 | 102 | 103 | … | 118 | 119 | 120 | … | 133 | 134 | 135 |
| | 136 | 137 | 138 | 139 | … | 152 | 153 | 154 | … | 169 | 170 | 171 | … | 186 | 187 | 188 | … | 201 | 202 | 203 |
| | 204 | 205 | 206 | 207 | … | 220 | 221 | 222 | … | 237 | 238 | 239 | … | 254 | 255 | 256 | … | 269 | 270 | 271 |

- A quad,cache node has only 1 NUMA node with all CPUs on the NUMA node 0 (DDR memory). MCDRAM is hidden from the “numactl -H” result since it is a cache.
- A quad,flat node has only 2 NUMA nodes with all CPUs on the NUMA node 0 (DDR memory). NUMA node 1 has MCDRAM only
- A snc2,flat node has 4 NUMA domains with DDR memory and all CPUs on NUMA nodes 0 and 1. (NUMA node 0 has physical cores 0 to 33 and all corresponding hyperthreads, and NUMA node 1 has physical cores 34 to 67 and all corresponding hyperthreads). NUMA nodes 2 and 3 have MCDRAM only

# 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)     (on 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)     (on 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)     (on 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)     (on 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)
```

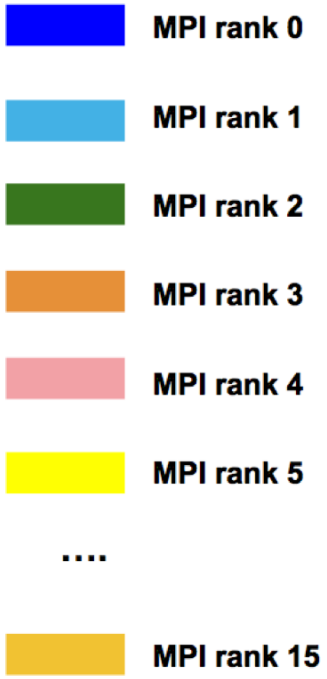
It is a mess!

# Importance of -c and --cpu_bind Options

- **The reason: 68 is not divisible by #MPI tasks!**
  - 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.
  - Now it looks good!
  - % 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)
```

# 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 | | | | | | | | | | | | | | | | | |
| 188 | | | | | | | | 196 | 197 | 198 | 199 | | | | | | |
| 256 | | | | | | | | | | | | | | | | | |

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

# Sample Job Script to Run on KNL quad,cache Nodes

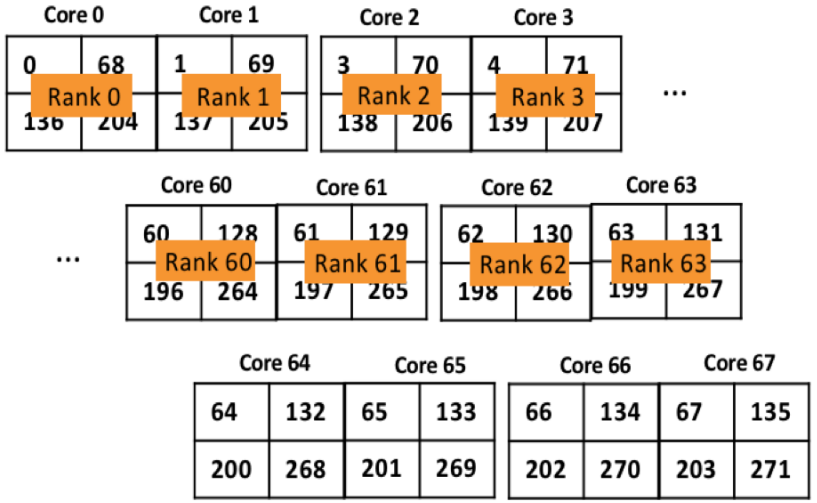
## Sample Job script (MPI+OpenMP)

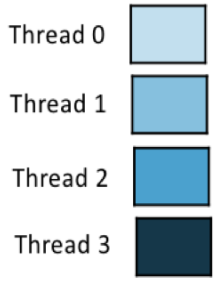
```
#!/bin/bash -l
#SBATCH -N 2
#SBATCH -q regular
#SBATCH -t 1:00:00
#SBATCH -L SCRATCH
#SBATCH -C knl,quad,cache

export OMP_NUM_THREADS=4
srun -n 128 -c 4 --cpu_bind=cores ./a.out
```

This job script requests 2 KNL quad, cache nodes to run 128 MPI tasks, 64 MPI tasks per node, allocating 4 CPUs per task, and binds each task to the 4 CPUs allocated within each core. The 4 OpenMP threads per MPI task are bound to the 4 CPUs in the core.

## Process and thread affinity




*Courtesy of Zhengji Zhao, NERSC*

# Sample Job Script to Run on KNL quad,cache Nodes

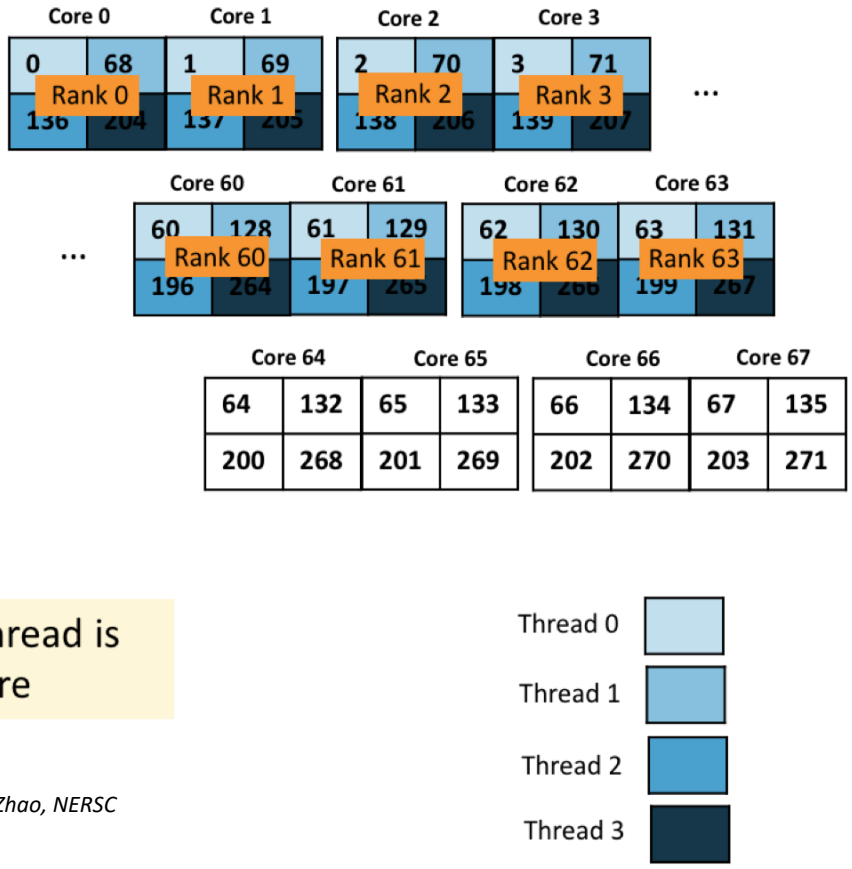
## Sample Job script (MPI+OpenMP)

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

export OMP_PROC_BIND=true
export OMP_PLACES=threads
export OMP_NUM_THREADS=4
srun -n 128 -c 4 --cpu_bind=cores ./a.out
```

With the above two OpenMP envs, each thread is now pinned to a single CPU within each core

## Process and thread affinity







*Courtesy of Zhengji Zhao, NERSC*

# 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 (need to request a reservation to use this mode)**
- **#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
```

# How to Run Debug and Interactive Jobs

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

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

- **interactive**
  - Instant allocation (or reject), run limit 1, max walltime 4 hrs, up to 64 nodes on Cori (Haswell and KNL) per repo

  **% salloc -N 2 -q interactive -C knl,quad,cache -t 2:00:00**

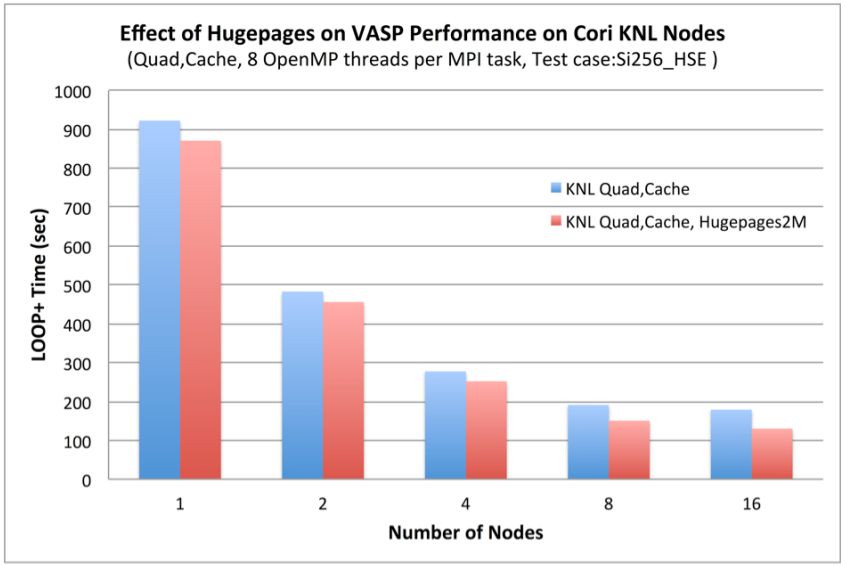
**Reservation is made for today (500 KNL nodes total):**

**% salloc -N 2 -q regular -C knl,quad,cache -t 1:00:00**

**--reservation=KNL_hack-a-thon -A ntrain**

# Recommend: Use 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 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
- Only works with sbatch, can not be used with salloc, which is already a wrapper script for srun.

# 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).**

# 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**

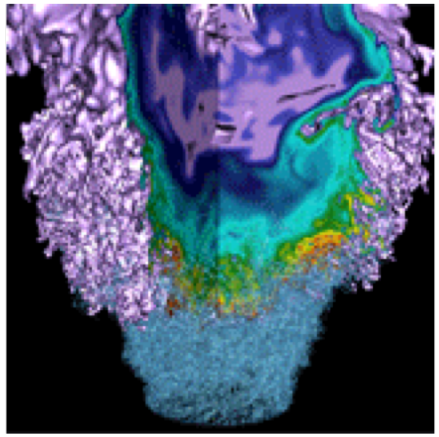
# Use Burst Buffer for faster IO

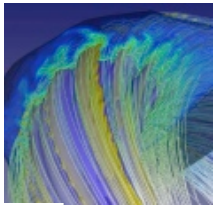
- Cori has 1.8PB of SSD-based “Burst Buffer” to support I/O intensive workloads
- Jobs can request a job-temporary BB filesystem, or a persistent (up to a few weeks) reservation
- More info at http://www.nersc.gov/users/computational-systems/cori/burst-buffer/

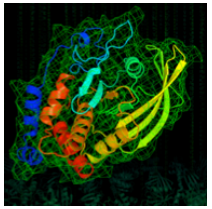
# Running Jobs on KNL Summary

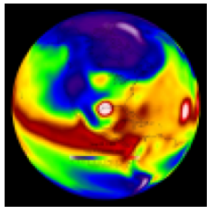
- **Use -C knl,<NUMA>,<MCDRAM> to request KNL nodes**
- **Consider using 64 cores out of 68 in most cases**
- **Always explicitly use srun's -c and --cpu_bind flags to spread the MPI tasks evenly over the cores/CPUs on the nodes**
- **Use OpenMP envs, OMP_PROC_BIND and OMP_PLACES to fine pin threads 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
- **Use the NERSC Job Script Generator tool**
- **More details at**
  - http://www.nersc.gov/users/computational-systems/cori/running-jobs/example-batch-scripts-for-knl/

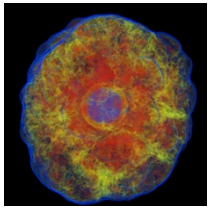
# How to Use Intel Tools on Cori KNL

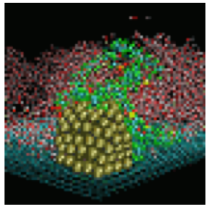












# VTune

- **VTune is a performance analysis tool**
- **Intel presentation**
  - http://www.nersc.gov/assets/Uploads/Intel-VTune-Amplifier-NERSC-2018.pdf
- **NERSC documentation**
  - http://www.nersc.gov/users/software/performance-and-debugging-tools/vtune/
- **Use Lustre file system (scratch) instead of GPFS (home or project)**
- **Compile: Use Cray compiler wrappers such as "cc' with "-g -dynamic"**
- **Run: add --perf=vtune in sbatch or salloc; use "srun" to launch a job**
- **View: use GUI to view results**

# Using VTune Example

## Compile and Run

% module swap craype-haswell craype-mic-knl

% cc -g -dynamic -qopenmp -O3 -o mycode.exe mycode.c *# can still use "-O3" along with "-g"*

**% module load vtune**

% salloc -N 2 -t 30:00 -C knl,quad,cache --reservation=KNL_hack-a-thon -q regular -A ntrain --perf=vtune *# today only*

Or % salloc -N 2 -t 30:00 -C knl,quad,cache -q interactive --perf=vtune *# or "-q debug"*

% export OMP_NUM_THREADS=8

% export OMP_PROC_BIND=spread

% export OMP_PLACES=threads

% srun -n 8 -c 64 --cpu_bind=cores amplxe-cl **-collect hotspots** -r res_dir -trace-mpi -finalization-mode=none -- ./mycode.exe

# can add -data-limit=0 to get >500 MB profiling data

% srun -n 8 -c 64 –cpu_bind=cores amplxe-cl **-collect memory-access -knob analyze-openmp=true -knob analyze-mem-objects=true** -r res_dir -trace-mpi -- ./mycode.exe

# Using VTune Example

**Finalize Result on a login node:**

% module load vtune

% amplxe-cl **-finalize** --finalization-mode=full -r <your_result_dir> -search-dir <path_to_your_binary>

**View with GUI (NX is recommended)**

% ssh -Y <username>@cori.nersc.gov

% module load vtune

% amplxe-gui

# Current VTune Limitations

- **Due to the impact of Spectre/Meltdown security issues, the patches installed are not compatible with the Vtune kernel driver, so currently the sone of the collections using hardware event-based counters are not available, such as:**
  - Advanced hotspots
  - General Exploration
  - Memory Access

# APS (Application Performance Snapshot)

- **Available within VTune**
- **Intel presentation**
  - http://www.nersc.gov/assets/Uploads/Intel-VTune-Amplifier-NERSC-2018.pdf. (slides 29-37)
- **Use Lustre file system (scratch) instead of GPFS (home or project)**
- **Compile: Use Cray compiler wrappers such as "cc' with "-g -dynamic"**
- **Run: add --perf=vtune in sbatch or salloc; use "srun" to launch a job**
- **View: view text or html results**

# Using APS Example

## Compile and Run

% module swap craype-haswell craype-mic-knl
% cc -g -dynamic -qopenmp -O3 -o mycode.exe mycode.c

% module load vtune
% salloc -N 2 -t 1:00:00 -C knl,quad,cache --reservation=KNL_hack-a-thon -q regular -A ntrain --perf=vtune. *# today only*
Or % salloc -N 2 -t 1:00:00 -C knl,quad,cache -q interactive *# or "-t 30:00 -q debug"*
% . $APS_DIR/apsvars.sh
% export OMP_NUM_THREADS=8
srun -n 8 -c 64 --cpu_bind=cores aps mycode.exe
aps --report <my_report_name> # generate text and html reports
aps-report -t <my_report_name> # generate MPI statistics

# Advisor

- **Advisor includes threading advisor and vectorization advisor**
- **Intel presentation**
  - http://www.nersc.gov/assets/Uploads/Intel-Advisor-NERSC-2018.pdf
- **NERSC documentation**
  - http://www.nersc.gov/users/software/performance-and-debugging-tools/advisor/
- **Use Lustre file system (scratch) instead of GPFS (home or project)**
- **Compile: Use cray compiler wrappers such as "cc" with "-g -dynamic" or use native Intel compilers such as "mpiicc" with "-g"**
- **Run: use "srun" or "mpirun" to launch a job**
- **View: use GUI to view results**

# Using Advisor Roofline Example

## Compile and Run

% module swap craype-haswell craype-mic-knl
% cc -g -dynamic -qopenmp -O3 -o mycode.exe mycode.c *# can still use "-O3" along with "-g"*

% salloc -N 2 -t 1:00:00 -C knl,quad,cache --reservation=KNL_hack-a-thon -q regular -A ntrain *# today only*
Or % salloc -N 2 -t 1:00:00 -C knl,quad,cache -q interactive *# or "-t 30:00 -q debug"*
% module load advisor
% export OMP_NUM_THREADS=8
# From advisor/2018.up1 (which is the default) one step collection for non-MPI codes
# still needs to do 2-step collection for MPI codes
% srun -n 8 -c 64 --cpu_bind=cores advixe-cl **-collect roofline** -project-dir=./myproj -data-limit=0 **-trace-mpi** -- ./mycode.exe
# add "-no-auto-finalize" if the collection is too slow
# "-collect roofline" does the following two steps automatically:
% srun -n 8 -c 64 --cpu_bind=cores advixe-cl **-collect survey** -project-dir=./myproj -trace-mpi -- ./mycode.exe
% srun -n 8 -c 64 --cpu_bind=cores advixe-cl **-collect tripcounts -flop** -project-dir=./myproj -trace-mpi -- ./mycode.exe

# Using Advisor Roofline Example

## View with GUI (NX is recommended)

% ssh -Y <username>cori.nersc.gov
% module load advisor
% advixe-gui

## Write Report into CSV

% advixe-cl --report survey --project-dir <my-project-dir> --show-all-columns --format=csv --report-output <my-output-file.csv>

# Inspector

- **Inspector is a memory and threading error checking tool**
- **Intel presentation**
  - https://www.nersc.gov/assets/Uploads/Inspector-NERSC-2018.pdf
- **NERSC documentation**
  - http://www.nersc.gov/users/software/performance-and-debugging-tools/inspector/
- **Use Lustre file system (scratch) instead of GPFS (home or project)**
- **Compile: Use native Intel compilers such as "mpiicc" (instead of Cray compiler wrappers such as "cc') with "-g -dynamic"**
- **Run: use "mpirun" to launch a job**
- **View: use GUI to view results**

# Using Inspector Example

## Compile and Run

% module swap craype-haswell craype-mic-knl
% module load impi
% mpiicc -g -dynamic -qopenmp -O3 -o mycode.exe mycode.c *# can still use "-O3" along with "-g"*
% salloc -N 2 -t 30:00 -C knl,quad,cache --reservation=KNL_hack-a-thon -q regular -A ntrain *# today only*
Or % salloc -N 2 -t 30:00 -C knl,quad,cache -q interactive *# or -q debug*
% module load inspector
% module load impi
% export I_MPI_PMI_LIBRARY=/usr/lib64/slurmpmi/libpmi.so
# use small number of MPI ranks and OpenMP threads, the goal is to detect threading and memory issues.
% export OMP_NUM_THREADS=2
% mpirun -n 2 inspxe-cl **-collect ti2** -result-dir=./myresult -- ./mycode.exe

## View with GUI (NX is recommended)

% ssh -Y <username>@cori.nersc.gov
% module load inspector
% inspxe-gui


# Thank you.