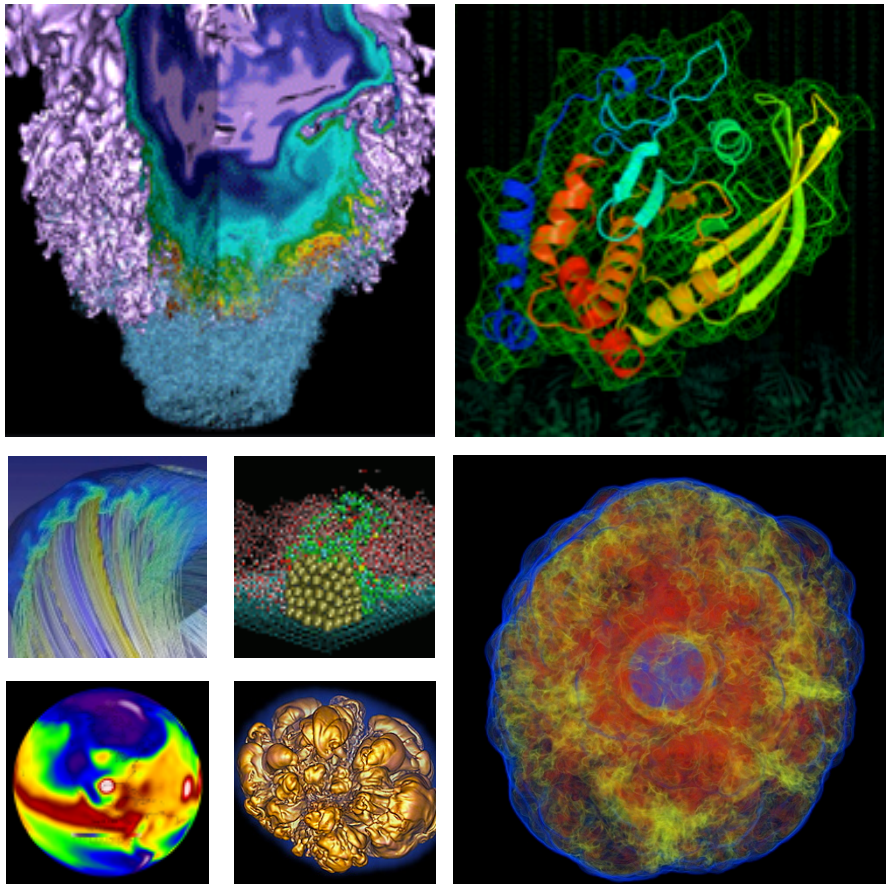
Nested OpenMP

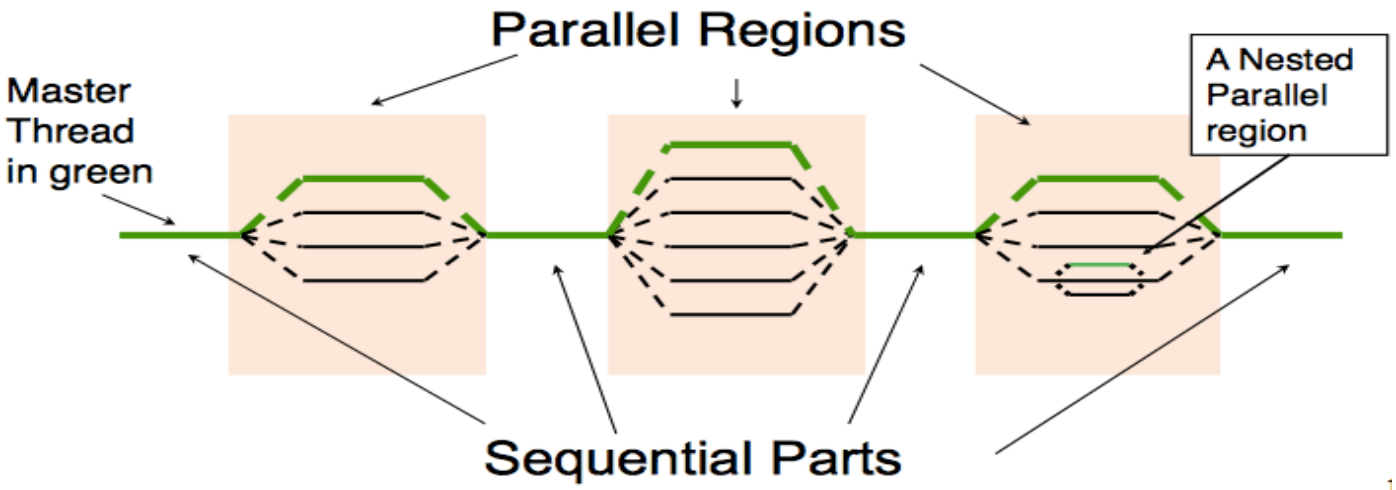


Helen He

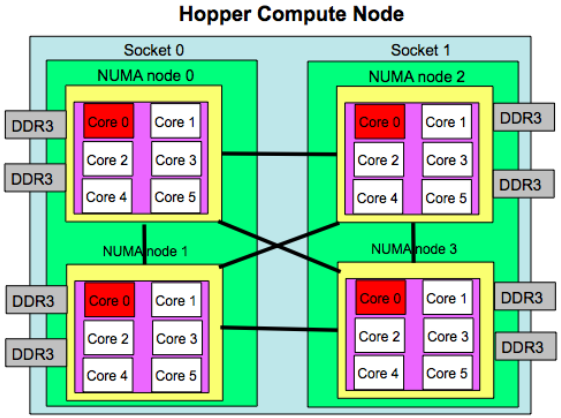
NUG Meeting, 10/08/2015

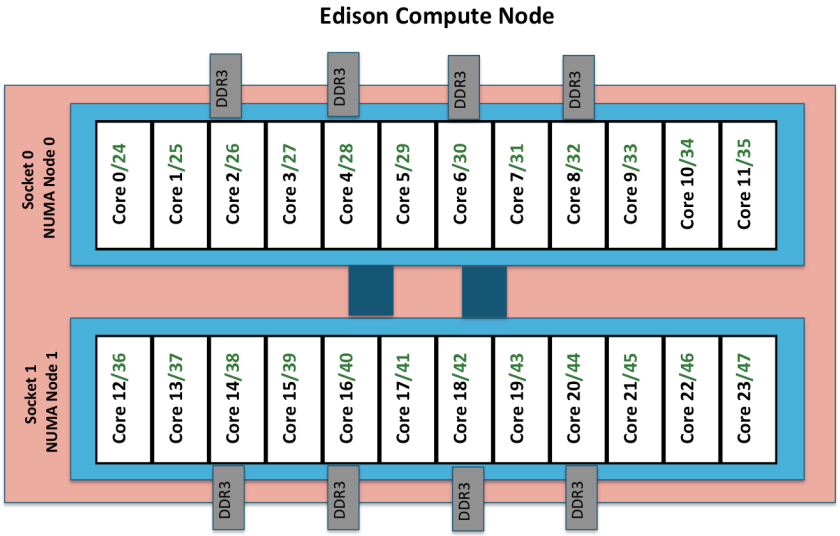
OpenMP Execution Model

- **Fork and Join Model**
  - Master thread forks new threads at the beginning of parallel regions.
  - Multiple threads share work in parallel.
  - Threads join at the end of the parallel regions.



Hopper/Edison Compute Nodes



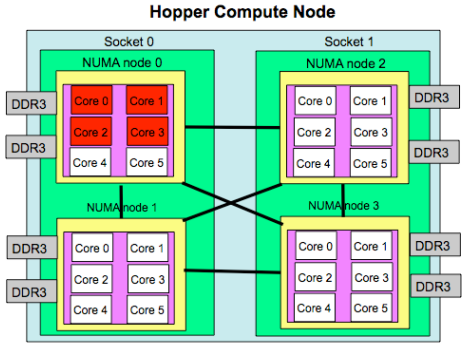


- Hopper: NERSC Cray XE6, 6,384 nodes, 153,126 cores.
  - 4 NUMA domains per node, 6 cores per NUMA domain.
- Edison: NERSC Cray XC30, 5,576 nodes, 133,824 cores.
  - 2 NUMA domains per node, 12 cores per NUMA domain.
    2 hardware threads per core.
- Memory bandwidth is non-homogeneous among NUMA domains.

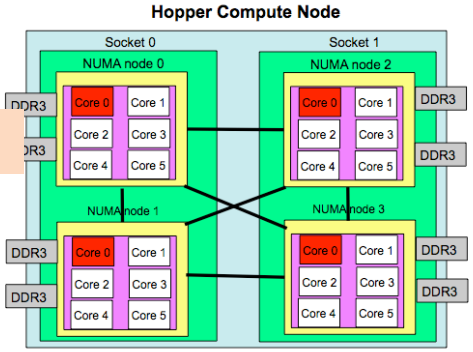
MPI Process Affinity: aprun "-S" Option

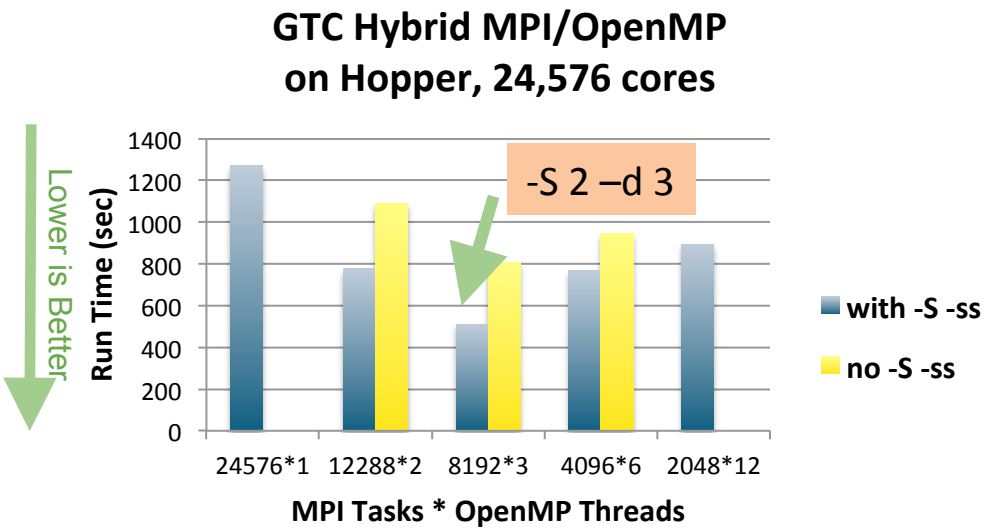
- Process affinity: or CPU pinning, binds MPI process to a CPU or a ranges of CPUs on the node.
- Important to spread MPI ranks evenly onto different NUMA nodes.
- Use the "-S" option for Hopper/Edison.

aprun –n 4 –d 6



aprun –n 4 –S 1–d 6



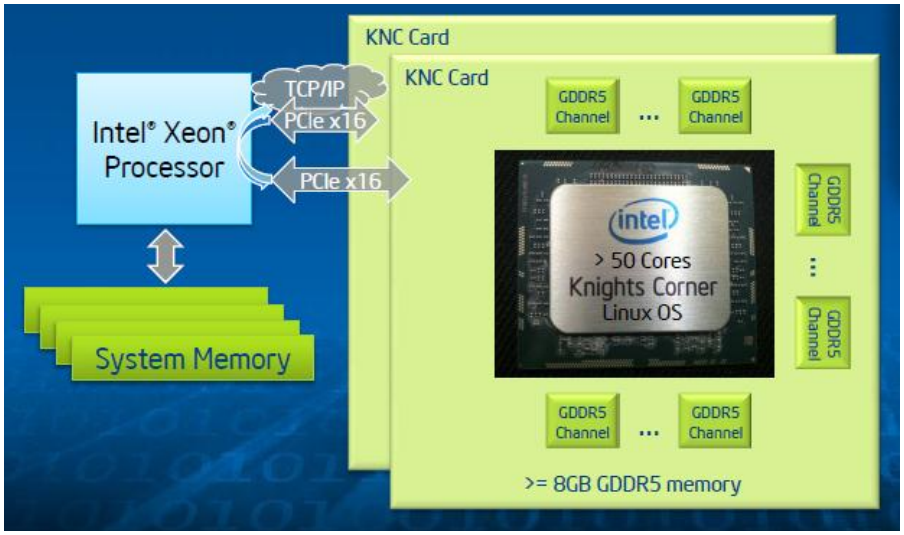


Thread Affinity: aprun "-cc" Option

- **Thread affinity: forces each process or thread to run on a specific subset of processors, to take advantage of local process state.**
- **Thread locality is important since it impacts both memory and intra-node performance.**
- **On Hopper/Edison:**
  - The default option is "-cc cpu" (use it for non-Intel compilers), binds each PE to a CPU within the assigned NUMA node.
  - Pay attention to Intel compiler, which uses an extra thread.
    - Use "-cc none" if 1 MPI process per node
    - Use "-cc numa_node" (Hopper) or "-cc depth" (Edison) if multiple MPI processes per node

NERSC KNC Testbed: Babbage

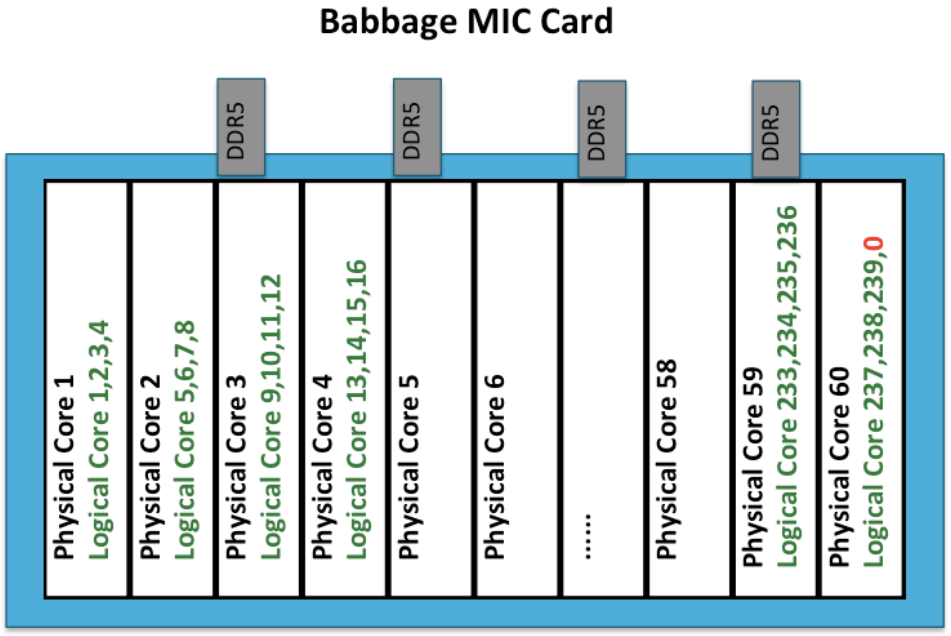
- **NERSC Intel Xeon Phi Knights Corner (KNC) testbed**
- **45 compute nodes, each has:**
  - Host node: 2 Intel Xeon Sandybridge processors, 8 cores each.
  - 2 MIC cards each has 60 native cores and 4 hardware threads per core.
  - MIC cards attached to host nodes via PCI-express.
  - 8 GB memory on each MIC card
- **Recommend to use at least 2 threads per core to hide latency of in-order execution.**



To best prepare codes on Babbage for Cori:

- Use "native" mode on KNC to mimic KNL, which means ignore the host, just run completely on KNC cards.
- Encourage to explore single node optimization for threading scaling and vectorization on KNC cards with problem sizes that can fit.
- "Symmetric", "Offload" modes on KNC and "OpenMP 4.0 target" work, but are not our promoted usage models for Babbage.

Babbage MIC Card



Babbage: NERSC Intel Xeon Phi testbed, 45 nodes. 2 MIC cards per node.

- **1 NUMA domain per MIC card: 60 physical cores, 240 logical cores. OpenMP threading potential to 240-way.**
- **KMP_AFFINITY, KMP_PLACE_THREADS, OMP_PLACES, OMP_PROC_BIND for thread affinity control**
- **I_MPI_PIN_DOMAIN for MPI/OpenMP process and thread affinity control.**

Full OpenMP 4.0 Support in Compilers

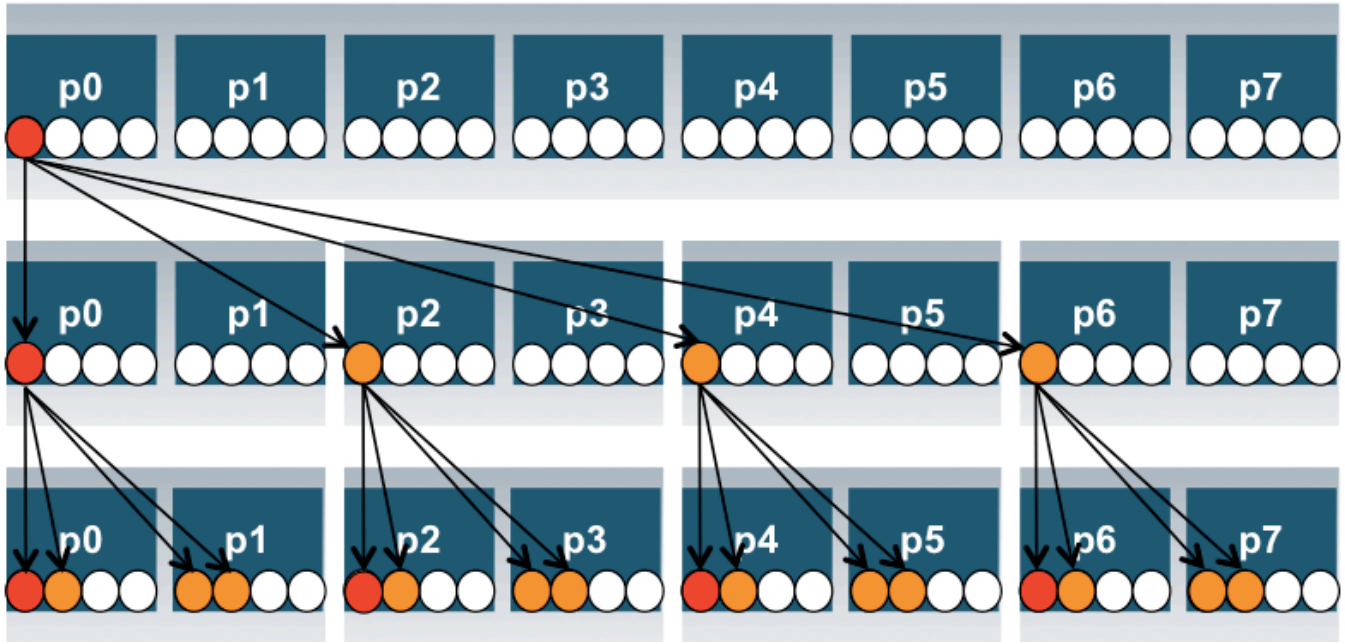
- **GNU compiler**
  - From 4.9.0 for C/C++
  - From gcc/4.9.1 for Fortran
- **Intel compiler**
  - From intel/15.0: supports most features in OpenMP 4.0; From Intel/16.0: full support
- **Cray compiler**
  - From cce/8.4.0

Thread Affinity Control in OpenMP 4.0

- **OMP_PLACES: a list of places that threads can be pinned on**
  - threads: Each place corresponds to a single hardware thread on the target machine.
  - cores: Each place corresponds to a single core (having one or more hardware threads) on the target machine.
  - sockets: Each place corresponds to a single socket (consisting of one or more cores) on the target machine.
  - A list with explicit place values: such as:
    - "{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}"
    - "{0:4},{4:4},{8:4},{12:4}"
- **OMP_PROC_BIND**
  - spread: Bind threads as evenly distributed (spread) as possible
  - close: Bind threads close to the master thread while still distributing threads for load balancing, wrap around once each place receives one thread
  - master: Bind threads the same place as the master thread

Nested OpenMP Thread Affinity Illustration

```
setenv OMP_PLACES threads
setenv  OMP_NUM_THREADS 4,4
setenv OMP_PROC_BIND spread,close
```



Sample Nested OpenMP Code

```
#include <omp.h>
#include <stdio.h>
void report_num_threads(int level)
{
   #pragma omp single {
       printf("Level %d: number of threads in the
team: %d\n", level, omp_get_num_threads());
      }
}
int main()
{
   omp_set_dynamic(0);
   #pragma omp parallel num_threads(2) {
      report_num_threads(1);
      #pragma omp parallel num_threads(2) {
         report_num_threads(2);
         #pragma omp parallel num_threads(2) {
            report_num_threads(3);
         }
      }
   }
   return(0);
}
```

```
% a.out
Level 1: number of threads in the team: 2
Level 2: number of threads in the team: 1
Level 3: number of threads in the team: 1
Level 2: number of threads in the team: 1
Level 3: number of threads in the team: 1
```

```
% setenv OMP_NESTED TRUE
% a.out
Level 1: number of threads in the team: 2
Level 2: number of threads in the team: 2
Level 2: number of threads in the team: 2
Level 3: number of threads in the team: 2
Level 3: number of threads in the team: 2
Level 3: number of threads in the team: 2
Level 3: number of threads in the team: 2
```

```
Level 0: P0
Level 1: P0 P1
Level 2: P0 P2; P1 P3
Level 3: P0 P4; P2 P5; P1 P6; P3 P7
```

When to Use Nested OpenMP

- **Some application teams are exploring with nested OpenMP to allow more fine-grained thread parallelism.**
  - MPI/Hybrid not using node fully packed
  - Top level OpenMP loop does not use all available threads
  - Multiple levels of OpenMP loops are not easily collapsed
  - Certain computational intensive kernels could use more threads
  - MKL can use extra cores with nested OpenMP

Process and Thread Affinity in Nested OpenMP

- **Achieving best process and thread affinity is crucial in getting good performance with nested OpenMP, yet it is not straightforward to do so.**
- **A combination of OpenMP environment variables and run time flags are needed for different compilers and different batch schedulers on different systems.**

Example: Use Intel compiler with Torque/Moab on Edison:

```
setenv OMP_NESTED true
setenv  OMP_NUM_THREADS 4,3
setenv OMP_PROC_BIND spread,close
aprun -n 2 -S 1 -d 12 –cc numa_node ./nested.intel.edison
```

Edison: Run Time Environment Variables

- **setenv OMP_NESTED true**
  - Default is false for most compilers
- **setenv OMP_MAX_ACTIVE_LEVELS 2**
  - The default was 1 for CCE prior to cce/8.4.0
- **setenv OMP_NUM_THREADS 4,3**
- **setenv OMP_PROC_BIND spread,close**
- **setenv KMP_HOT_TEAMS 1**
  - Intel only env. Default is false
- **setenv KMP_HOT_TEAMS_MAX_LEVELS 2**
  - Intel only env. Allow nested level OpenMP threads to stay alive instead of being destroyed and created again to reduce thread creation overhead.
- **aprun -n 2 -S 1 -d 12 –cc numa_node ./nested.intel.edison**
  - Use -d for total number of threads (products of num_threads from all levels). –cc numa_node to allow threads migrate within NUMA node to not affected by Intel's extra manager thread.

Babbage: Run Time Environment Variables

- Set I_MPI_PIN_DOMAIN=auto to get basic MPI process affinity
- Do not set KMP_AFFINITY, otherwise OMP_PROC_BIND will be ignored.
- Use OMP_PLACES = threads (default) instead of sockets
- setenv OMP_NESTED true
- setenv OMP_NUM_THREADS 4,3
- setenv OMP_PROC_BIND spread,close
- setenv KMP_HOT_TEAMS 1
- setenv KMP_HOT_TEAMS_MAX_LEVELS 2
- mpirun.mic -n 2 -host bc1109-mic0 ./xthi-nested.mic |sort

XGC1: Nested OpenMP

- **Always make sure to use best thread affinity. Avoid using threads across NUMA domains.**
- **Currently:**

```
export OMP_NUM_THREADS=6,4
export OMP_PROC_BIND=spread,close
export OMP_NESTED=TRUE
export OMP_STACKSIZE=8000000
aprun -n 200 -N 2 -S 1 -j 2 -cc numa_node ./xgca
```

- **Is a bit slower than (work ongoing):**

Courtesy of Robert Hager, PPPL and NESAP XGC1 team.

```
export OMP_NUM_THREADS=24
export OMP_NESTED=TRUE
export OMP_STACKSIZE=8000000
aprun -n 200 -d 24 -N 2 -S 1 -j 2 -cc numa_node ./xgca
```

- **Will try:**

```
export KMP_HOT_TEAMS=1
export KMP_HOT_TEAMS_MAX_LEVELS=2
```

- **Use num_threads clause in source codes to set threads for nested regions. For most other non-nested regions, use OMP_NUM_THREADS env for simplicity and flexibility.**

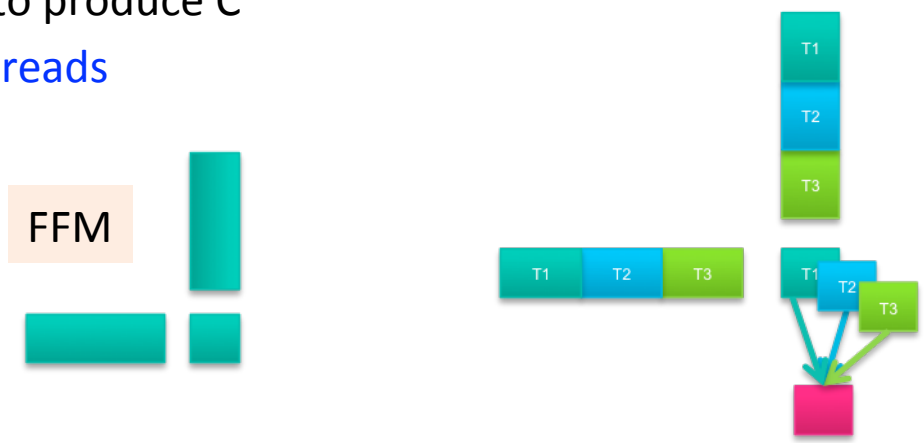
Use Multiple Threads in MKL

- **By Default, in OpenMP parallel regions, only 1 thread will be used for MKL calls.**
  - MKL_DYNAMICS is true by default
- **Nested OpenMP can be used to enable multiple threads for MKL calls. Treat MKL as a nested inner OpenMP region.**
- **Sample settings**

```
export OMP_NESTED=true
export OMP_PLACES=cores
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=6,4
export MKL_DYNAMICS=false
export KMP_HOT_TEAMS=1
export KMP_HOT_TEAMS_MAX_LEVELS=2
```

NWChem: OpenMP "Reduce" Algorithm

- **Plane wave Lagrange multiplier**
  - Many matrix multiplications of complex numbers, C = A x B
  - Smaller matrix products: FFM, typical size 100 x 10,000 x 100
  - Original threading scaling with MKL not satisfactory
- **OpenMP "Reduce" or "Block" algorithm**
  - Distribute work on A and B along the k dimension
  - A thread puts its contribution in a buffer of size m x n
  - Buffers reduced to produce C
  - OMP teams of threads

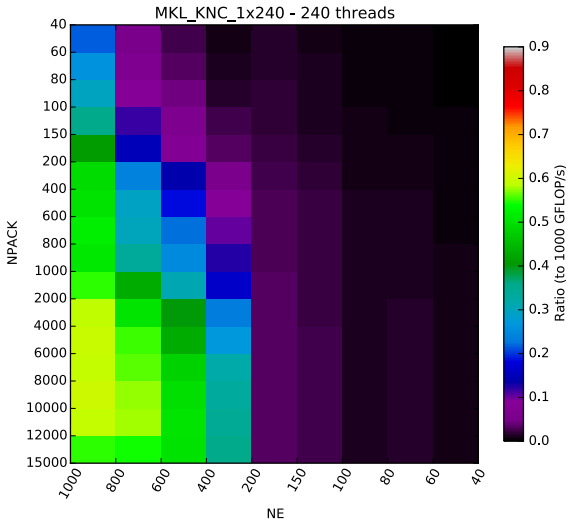


Courtesy of Mathias Jacquelin, LBNL

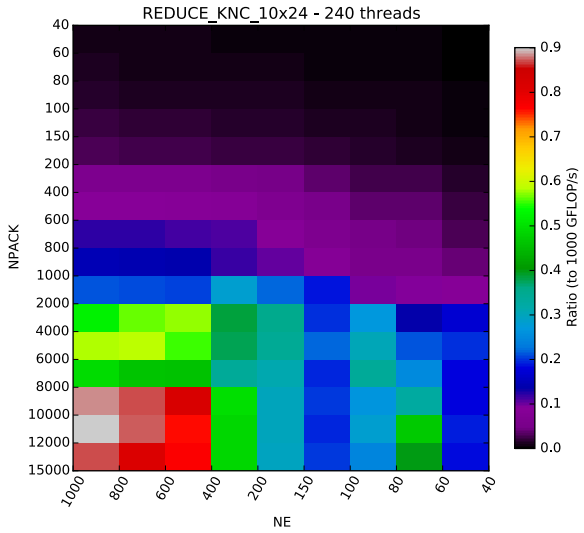
NWChem: OpenMP "Reduce" Algorithm

- Better for smaller inner dimensions, i.e. for FFMs
- Multiple FFMs can be done concurrently in different thread pools
- Threading enables us to use all 240 hardware threads
- Best "Reduce": 10 MPI, 6 teams of 4 threads (nested OpenMP with MKL)

MKL
1MPI, 240 threads

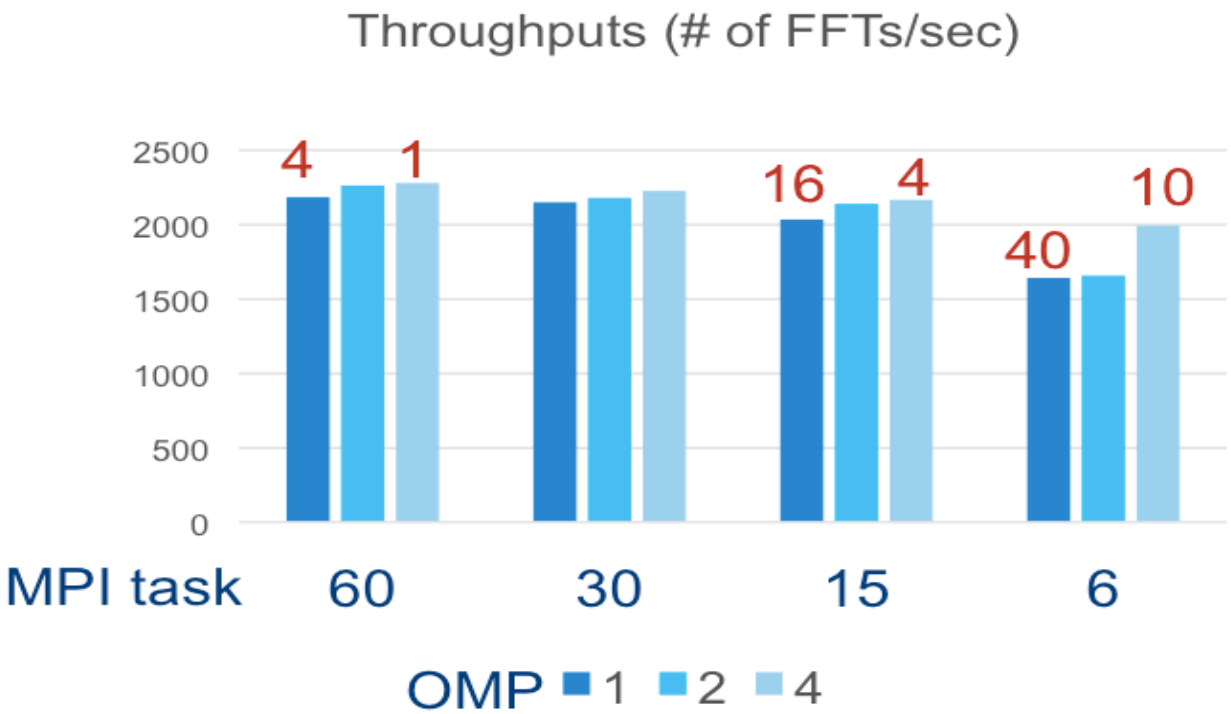


Best "Reduce"
10 MPI, 6 teams of 4 threads



Courtesy of Mathias Jacquelin, LBNL

FFT3D on KNC, Ng=64^3



$$N_{MKL} = 240/(N_{MPI} * OMP)$$

Courtesy of Jeongnim Kim, Intel

Nested OpenMP on NERSC Systems

- **Please see detailed example settings in the "Nested OpenMP" web page:**
  - Run on Edison and Babbage
  - With Intel and Cray compilers
  - Use Torque/Moab and SLURM batch schedulers
  - https://www.nersc.gov/users/computational-systems/edison/running-jobs/using-openmp-with-mpi/nested-openmp/

Thank you.