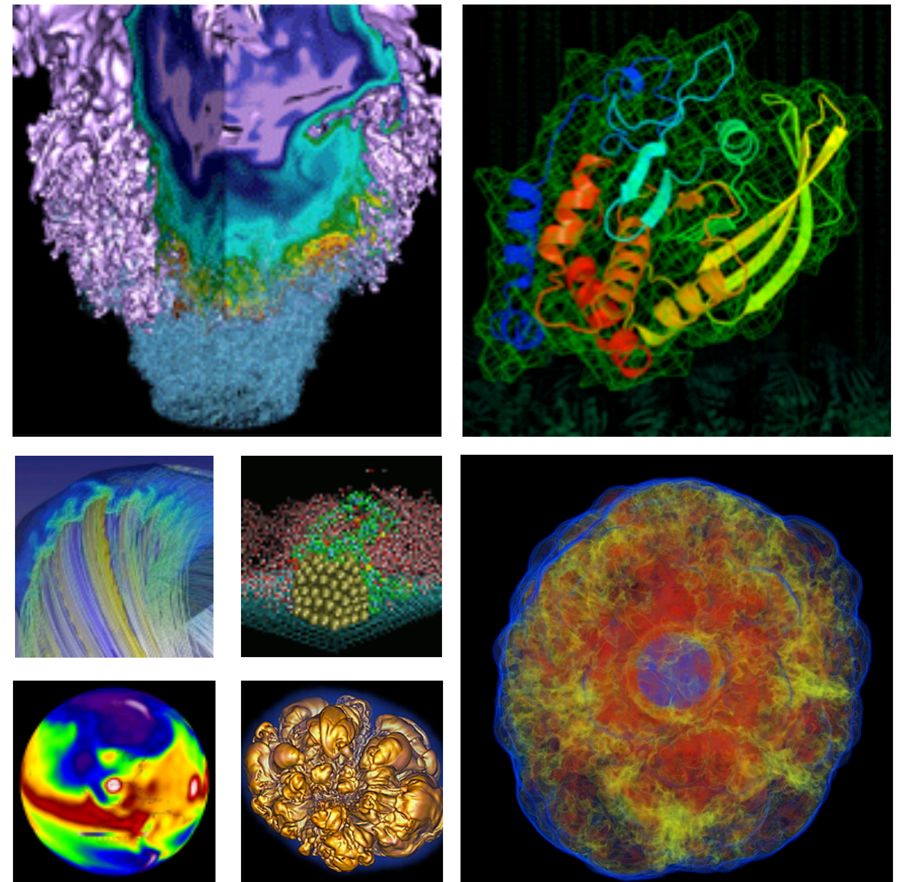
Using OpenMP at NERSC



Yun (Helen) He, Alice Koniges, Richard Gerber, Katie Antypas

OpenMPCon, Sept 28-30, 2015

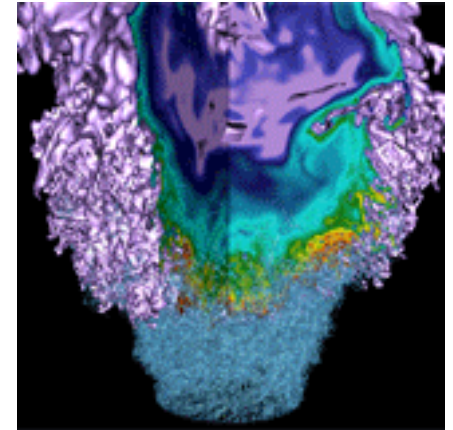
U.S. DEPARTMENT OF ENERGY | Office of Science

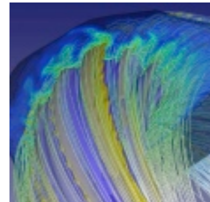
BERKELEY LAB
Lawrence Berkeley National Laboratory

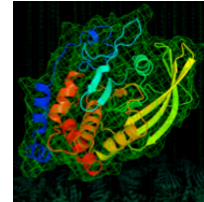
Outline

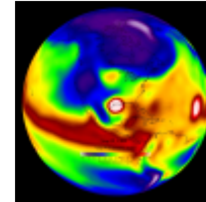
- **NERSC and our new system**
  - Why MPI + OpenMP is preferred
- **OpenMP usage at NERSC**
- **What do we tell users about OpenMP scaling**
  - Process and thread affinity
  - Scaling tips
  - Tools for OpenMP
- **Case studies of using and tuning MPI/OpenMP performance**

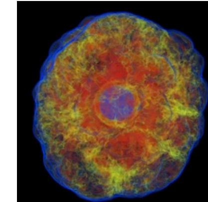
NERSC and our new system

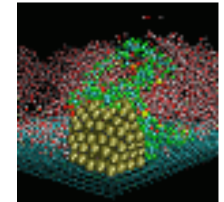












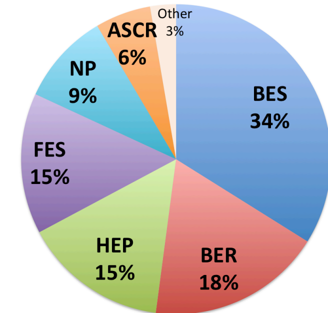
U.S. DEPARTMENT OF ENERGY | Office of Science

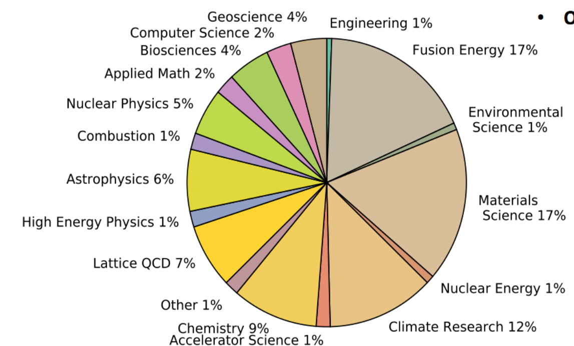
BERKELEY LAB

What is NERSC/LBNL

- **National Energy Research Scientific Computing Center (NERSC) is the primary computing facility for DOE Office of Science for its mission.**
  - 6,900 users, >850 projects, >600 codes.
- **Strong focus on Science**
  - A word-class resource to support world-class science.
  - 1,808 refereed journal publications, 22 journal covers (2014)
  - 4 NERSC users have won Nobel Prizes
- **NERSC collaborates with vendors to deploy advanced HPC and data resources**
  - Collaborate years before a system's delivery to influence hardware and software design
  - Hopper (N6) and Cielo (ACES) were the first Cray petascale systems with a Gemini
  - Edison (N7) is the first Cray petascale system with Intel processors, Aries interconnect and Dragonfly topology (serial #1)
  - Cori (N8) will be one of the first large Intel KNL systems and will have unique data capabilities
- **NERSC is a Division at the Lawrence Berkeley National Laboratory (LBNL) and one of three divisions in compute science areas.**

2014 Allocation Breakdown

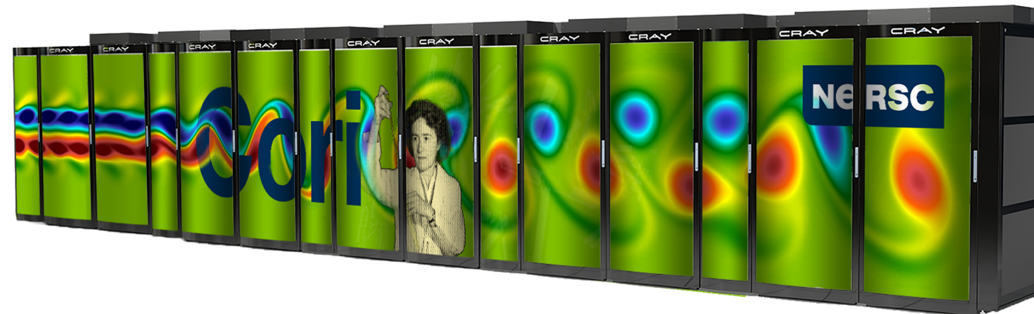




- **Over 6900 users**
- **Over 850 projects**

The Big Picture: KNL is Coming to NERSC

- **The next large NERSC production system "Cori" will be Intel Xeon Phi KNL (Knights Landing) architecture**
  - Energy efficient manycore system
  - > 9300 single socket nodes, multiple NUMA domains
  - Self-hosted, not an accelerator
  - 72 cores/node, 4 hardware threads per core. Total of 288 threads per node
  - AVX512, larger vector length of 512 bits (8 double-precision elements)
  - On package high-bandwidth memory (HBM)
  - Burst Buffer



Edison / Cori Quick Comparison

Edison (Ivy-Bridge)

NERSC Cray XC30 system

- 12 Cores Per CPU
- 24 Logical Cores Per CPU
- 2.4-3.2 GHz
- Vector length of 256 bits, 4 Double Precision Ops per Cycle (+ multiply/add)
- 2.5 GB of Memory Per Core
- ~100 GB/s Memory Bandwidth

Cori (Knights-Landing)

- 72 Physical Cores Per CPU
- 288 Logical Cores Per CPU
- Much slower GHz
- Vector length of 512 bits, 8 Double Precision Ops per Cycle (+ multiply/add)
- < 0.3 GB of Fast Memory Per Core
- < 2 GB of Slow Memory Per Core
- Fast memory has ~ 5x DDR4 bandwidth
- Burst Buffer for fast IO

Programming Considerations: Running on Cori

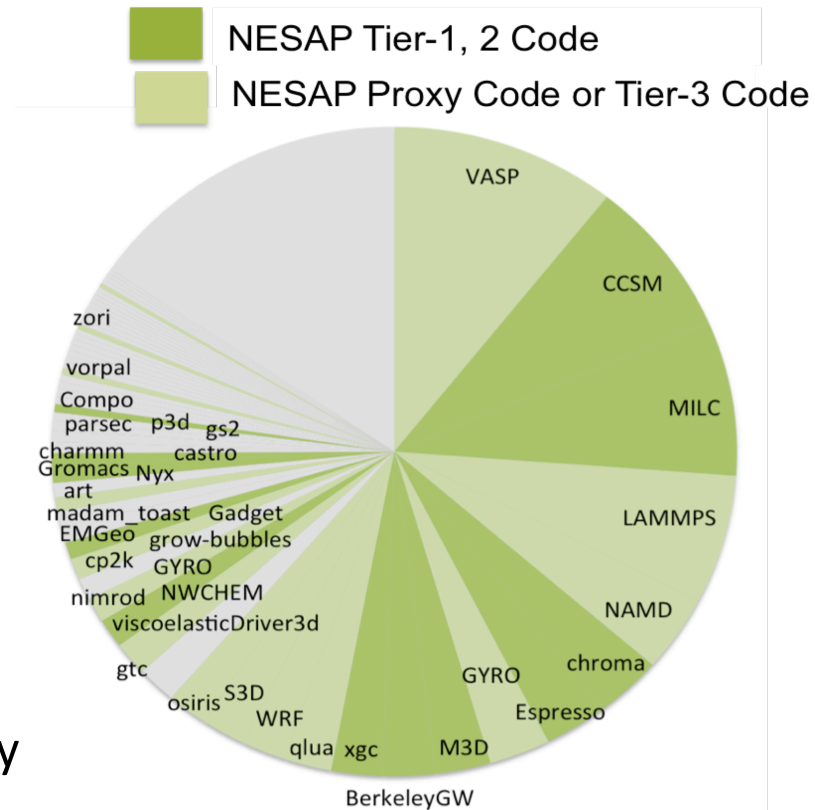
- Application is very likely to run on KNL with simple porting, but high performance is harder to achieve.
- Applications need to explore more on-node parallelism with thread scaling and vectorization, also to utilize HBM and burst buffer options.
- Many applications will not fit into the memory of a KNL node using pure MPI across all HW cores and threads because of the memory overhead for each MPI task.
- Hybrid MPI/OpenMP is the recommended programming model, to achieve scaling capability and code portability.
- Current NERSC systems (Edison/Hopper and Babbage) can help prepare codes for Cori.

Program Portability/Maintainability

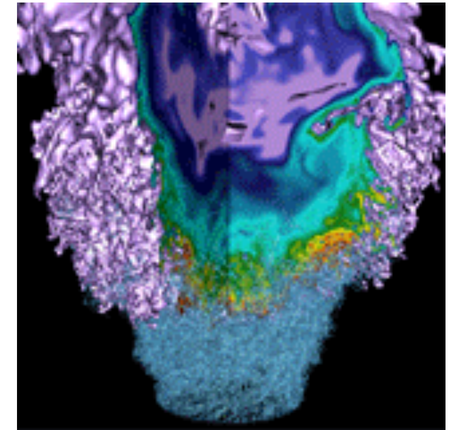
- **Many NERSC users are also users at other DOE labs. Program portability is important to help maintain single source version of an application.**
- **Avoid as much as possible: "#ifdef" for GPU/CPU, to use CUDA Fortran, OpenCL, OpenACC or OpenMP, and to use different compiler directives.**
- **Regardless of processor architecture, users will need to modify applications to achieve performance**
  – Expose more on-node parallelism in applications (OpenMP can help)
  – Increase application vectorization capabilities (OpenMP SIMD can help)
  – OpenMP is an industry standard that works on both CPU/GPU, promotes code portability

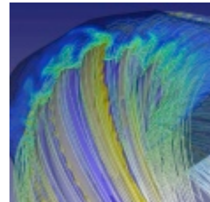
Application Readiness for Cori

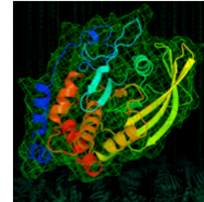
- **We begin to transition our workload to Cori manycore system**
  - 10 codes make up 45% of the workload
  - 25 codes make up 66% of the workload
- **NERSC Exascale Science Application Program (NESAP)**
  - 20 application code teams selected to work with Cray, Intel and NERSC
    - Some starts from adding OpenMP, then explore scaling
  - Close collaborations with other DOE facilities, vendors and science community
  - Trainings and lessons learned will be made available to all application teams and users.

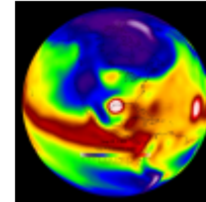


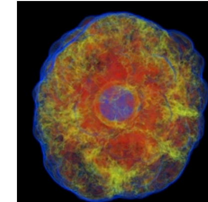
OpenMP Usage at NERSC

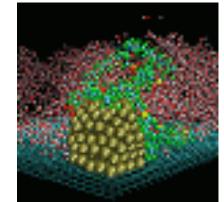








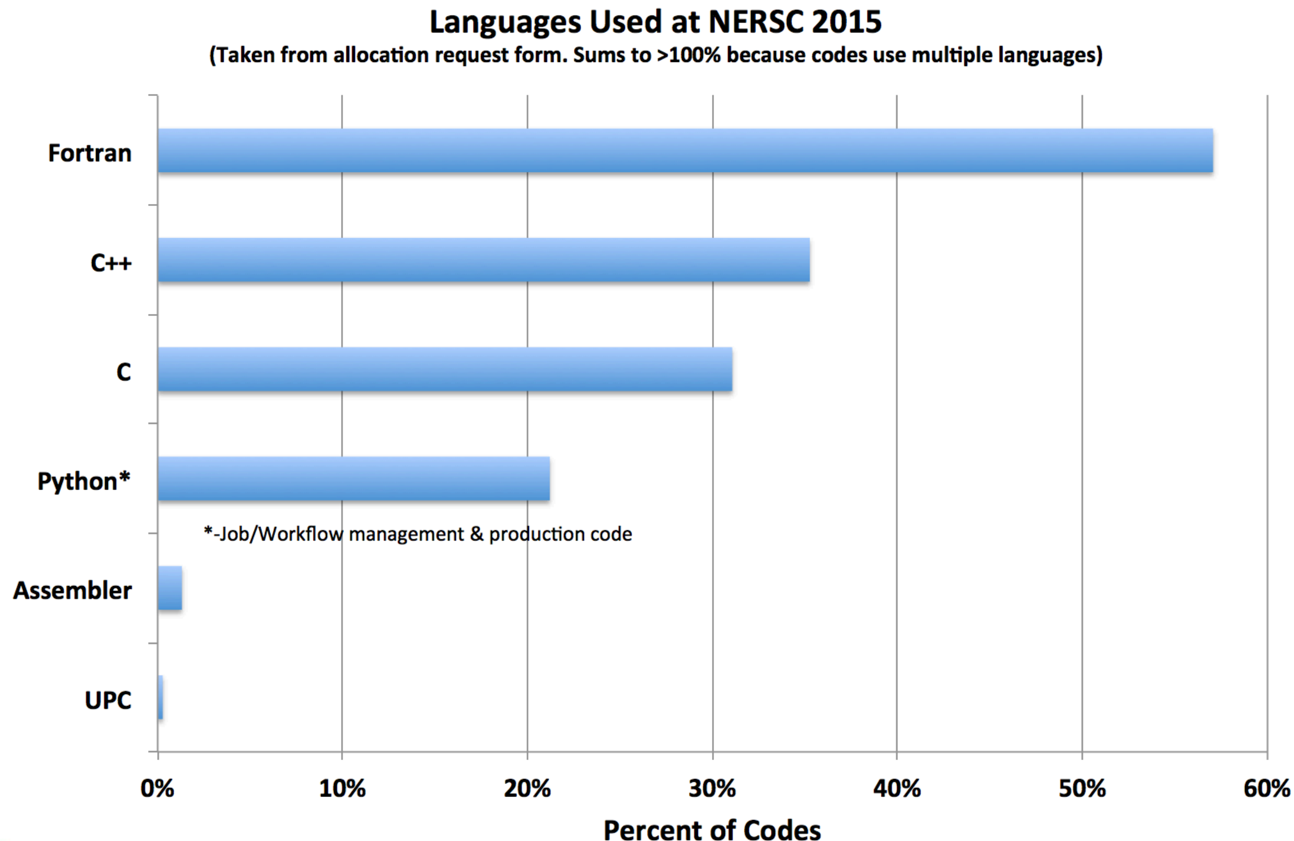




Languages Used at NERSC

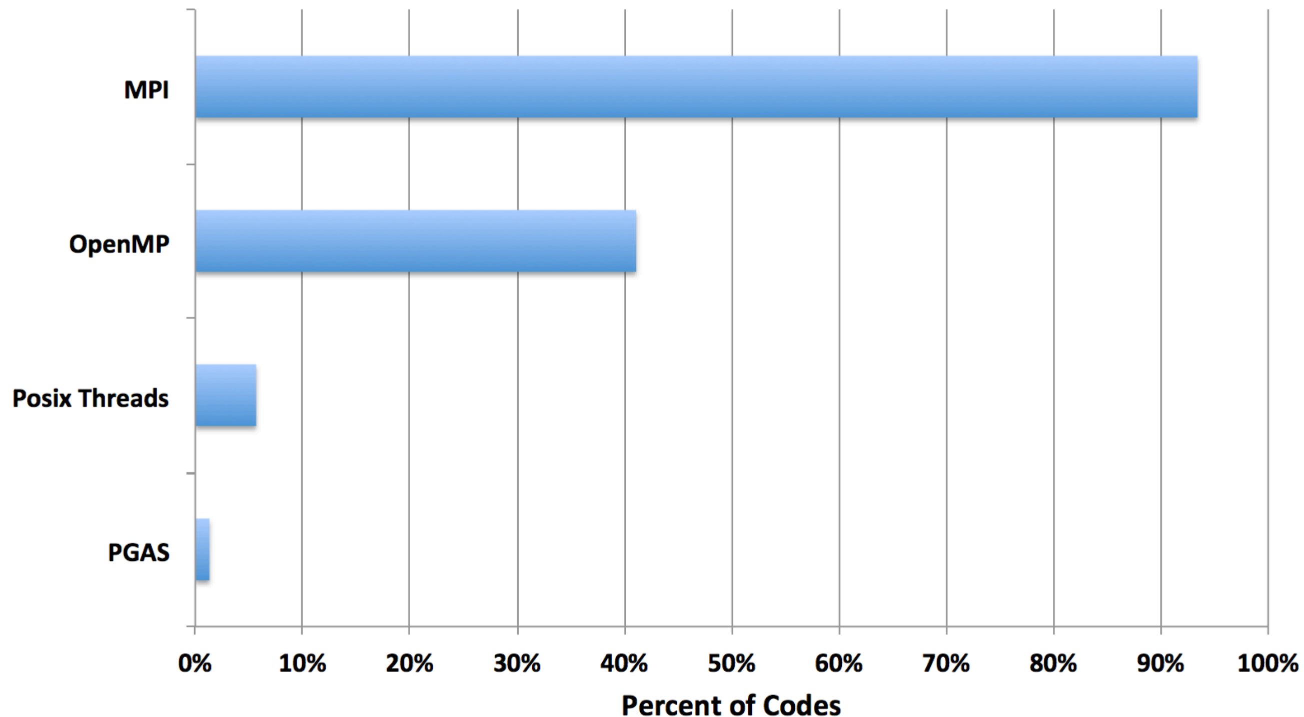
NERSC 40 YEARS at the FOREFRONT

- Here data are collected from all NERSC projects
- If by machine hours used, Fortran is even more popular: 23 out of 36 top codes primarily use Fortran



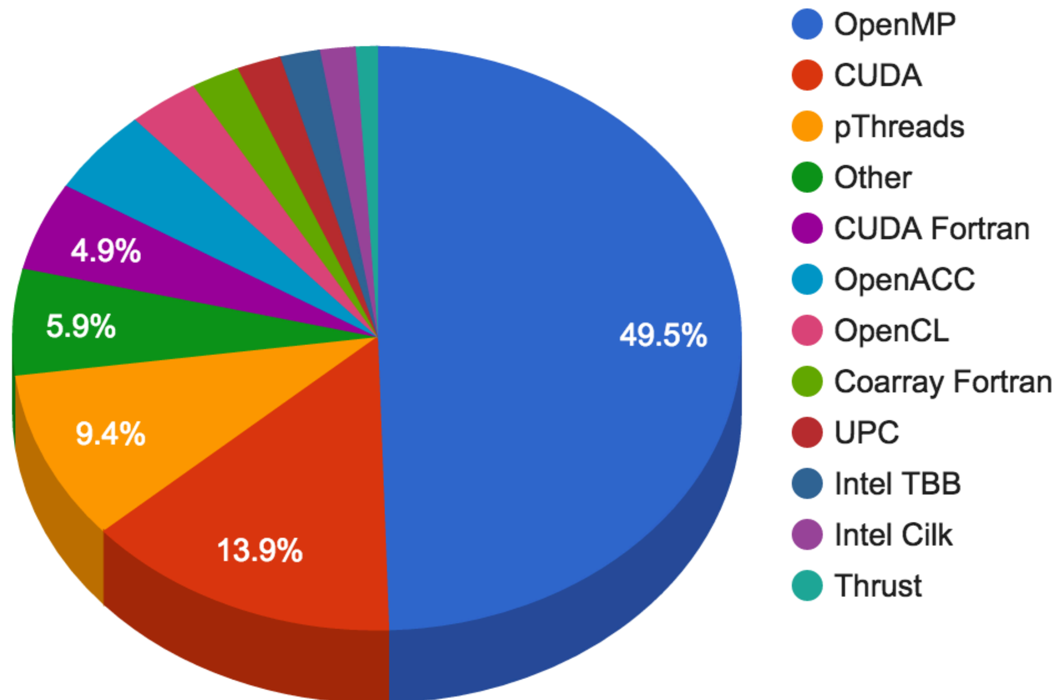
Programming Models Used at NERSC

- MPI dominates
- 40% of projects use OpenMP



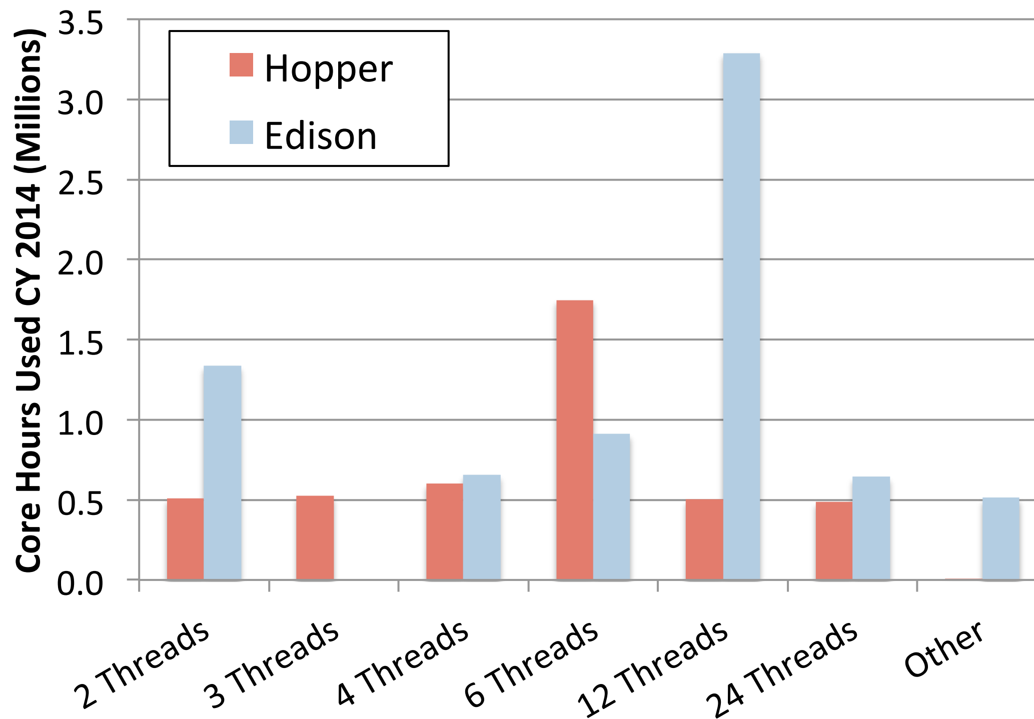
What is X if Use MPI+X at NERSC

OpenMP is about 50%, out of all choices of X



OpenMP Threads Usage at NERSC

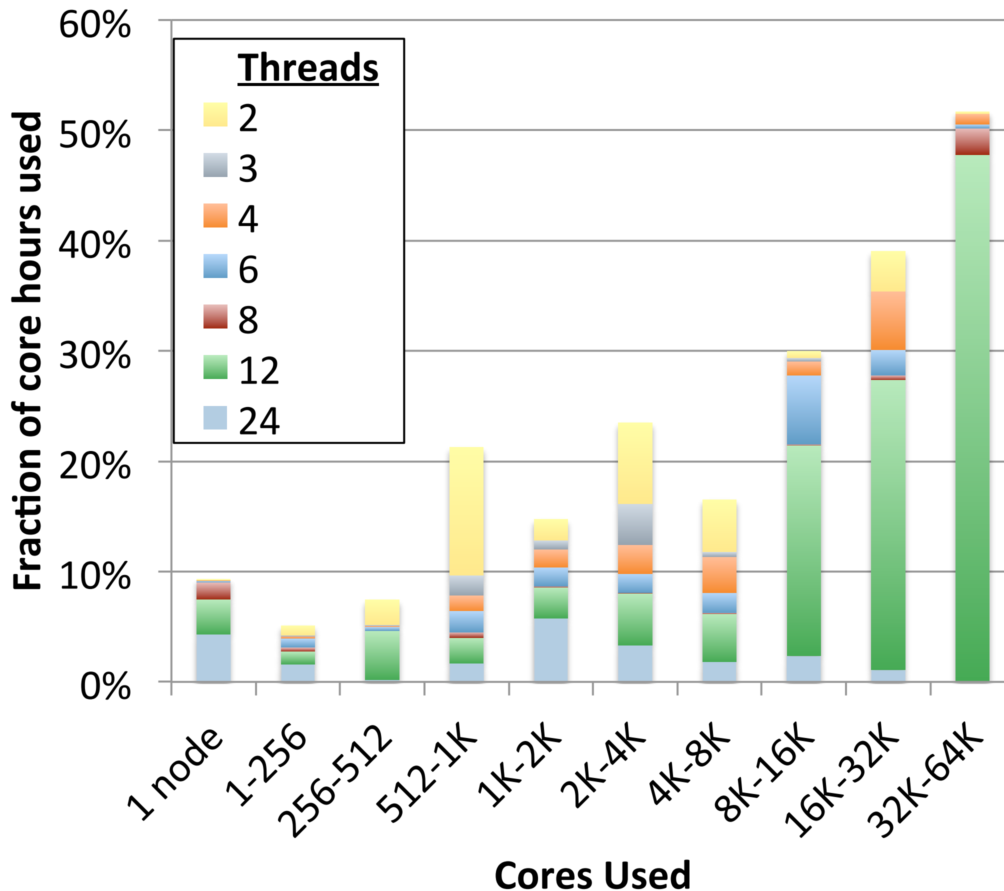
	MPP hours	Hopper	Edison
Fraction of hours using OpenMP	19%	14%	21%



- **Thread utilization is ~20% and increasing**
- **OpenMP adoption is not driven by memory capacity**
  - OpenMP usage is higher on Edison even though it has more memory per core.
- **Thread concurrency increases over generations**
  - Grows to match size of NUMA domains.

Brian Austin et. al., NERSC Workload Analysis

High Concurrency Jobs Use More Threads



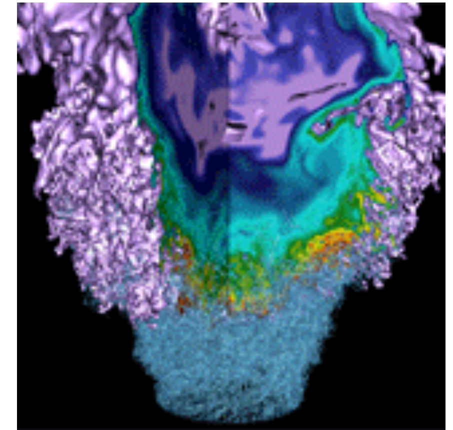
- **Thread utilization increases with node count**
  - More than half of the core hours using 2/3 of Edison are threaded. (not shown)
- **Thread concurrency increases with job size**
  - Jobs with 12 threads per process is dominate at higher concurrency.
- **Any OpenMP inefficiencies are outweighed by MPI scalability issues**

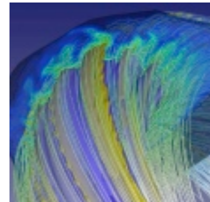
Brian Austin et. al., NERSC Workload Analysis

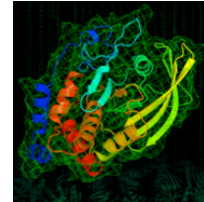
Adoption of Threads Varies Across Science Areas

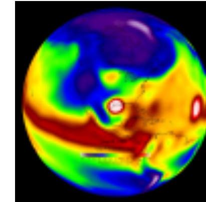
Brian Austin et. al., NERSC Workload Analysis

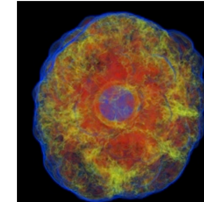
What do we tell users about OpenMP scaling

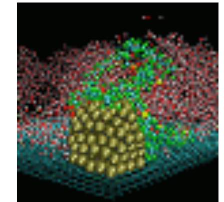








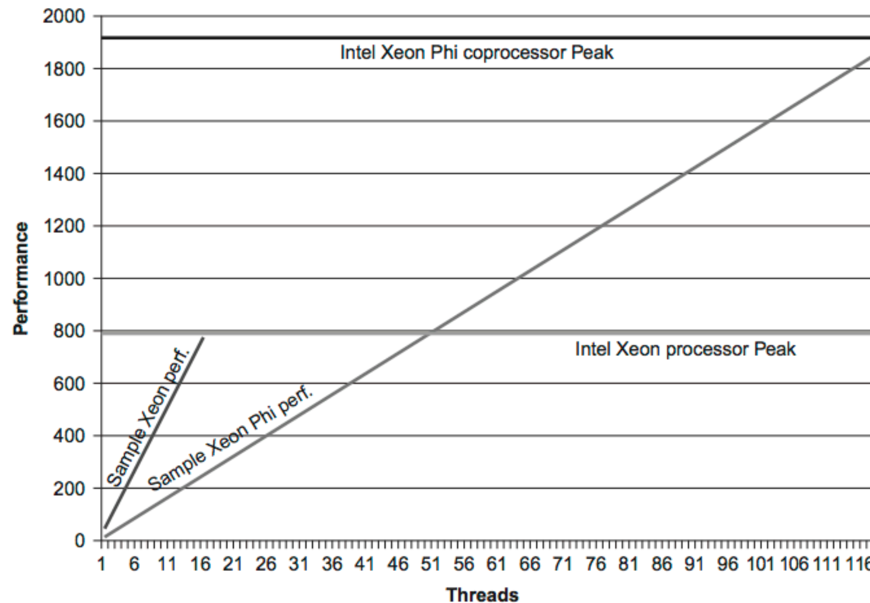




U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Why Scaling is So Important

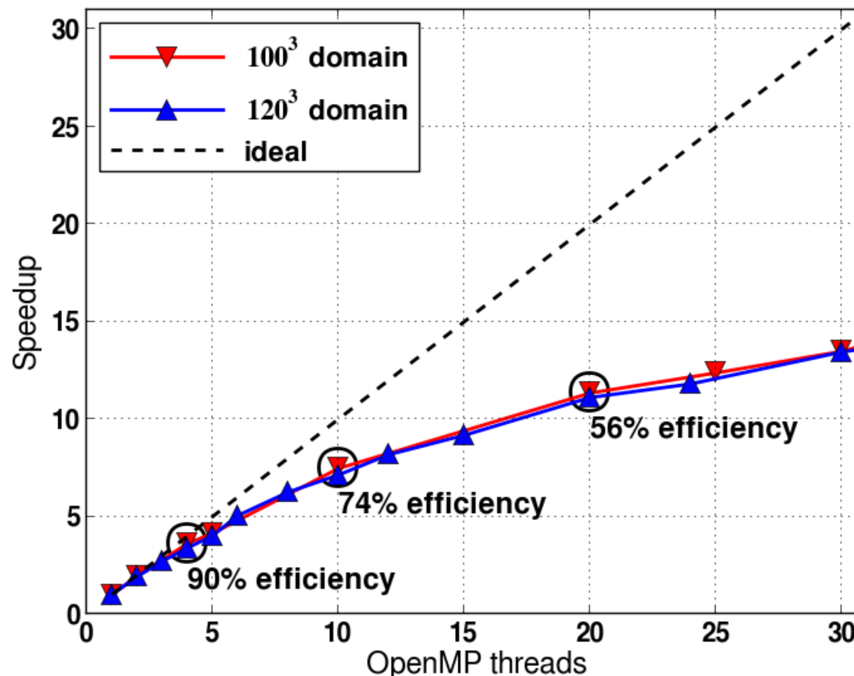


Courtesy of Jim Jeffers and James Reinders, Intel

- **Scaling of an application is important to get the performance potential on the Xeon Phi manycore systems.**
- **Does not imply to scale with "pure MPI" or "pure OpenMP"**
- **Does not imply the need to scale all the way to 240-way either**
- **Rather, should explore hybrid MPI/OpenMP, find some sweet spots with combinations, such as: 4 MPI tasks * 15 threads per task, or 8*20, etc.**

OpenMP Scaling Analysis

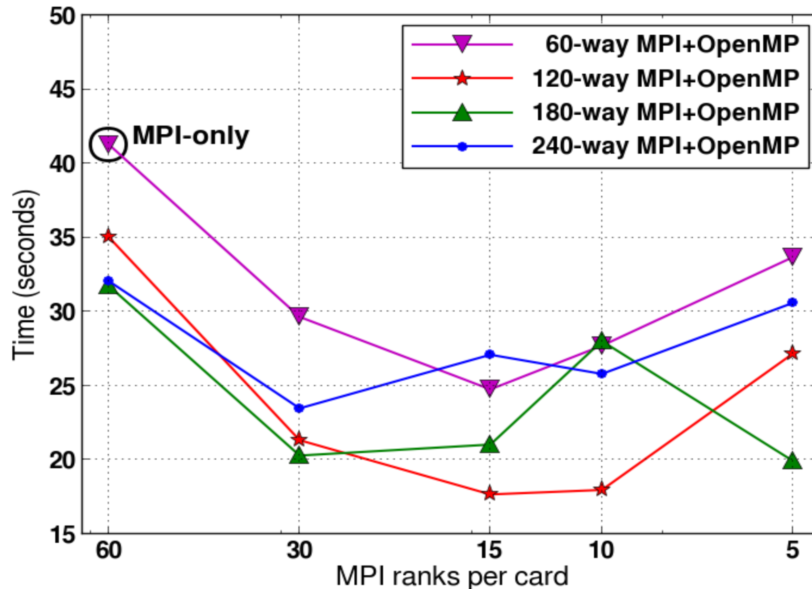
- **For the optimal rank vs. thread balance, assess the (relative) efficiency of the OpenMP implementation**
  - Hold number of ranks fixed, varying the number of threads



Courtesy of Chris Daley, NERSC

- **Helps to guide to choose optimal number of threads.**

MPI *vs.* OpenMP Scaling Analysis

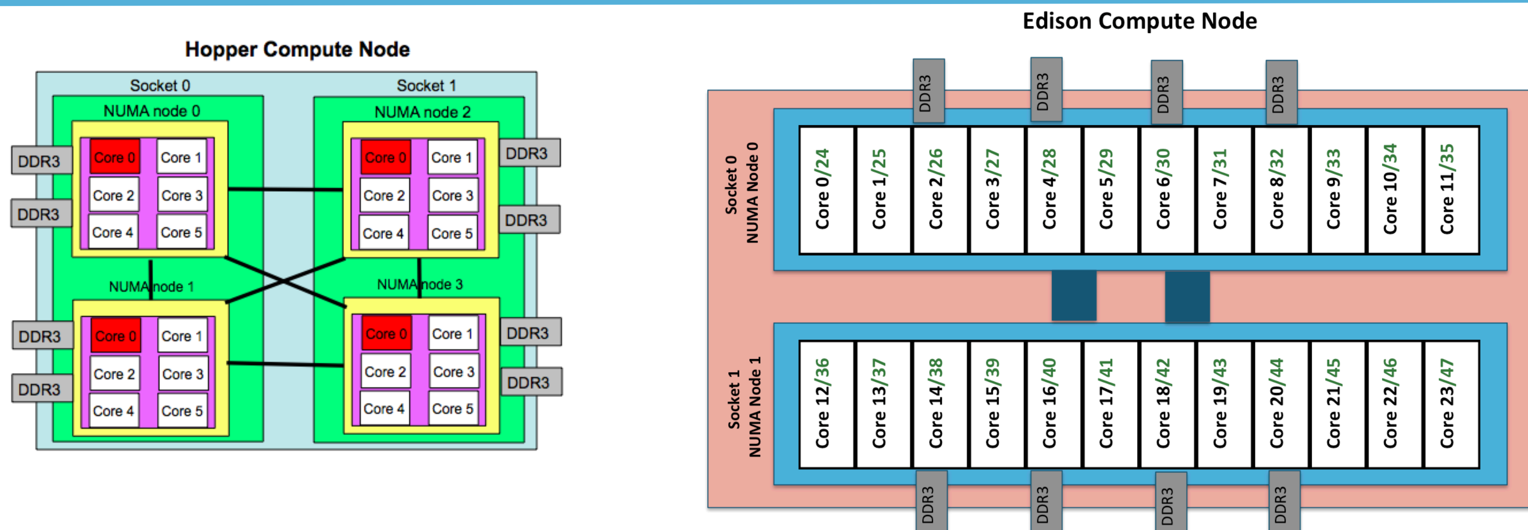


Courtesy of Chris Daley, NERSC

- Each line represents multiple runs using fixed total number of cores = #MPI tasks x #OpenMP threads/task.
- Scaling may depend on the kernel algorithms and problem sizes.
- In this test case, 15 MPI tasks with 8 OpenMP threads per task is optimal.

- **Understand your code by creating the MPI *vs.* OpenMP scaling plot, find the sweet spot for hybrid MPI/OpenMP.**
- **It can be the base setup for further tuning and optimizing on Xeon Phi.**

NERSC Systems: Hopper and Edison

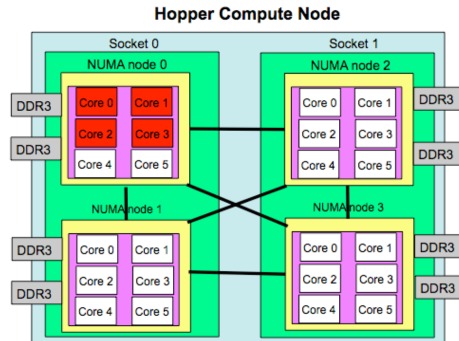


- Hopper: NERSC Cray XE6, 6,384 nodes, 153,126 cores.
  - 4 NUMA domains per node, 6 cores per NUMA domain.
- Edison: NERSC Cray XC30, 5,576 nodes, 133,824 cores.
  - 2 NUMA domains per node, 12 cores per NUMA domain.
    2 hardware threads per core.
- Memory bandwidth is non-homogeneous among NUMA domains.
- Edison can be used for exploring OpenMP thread parallelism and vecotorization.

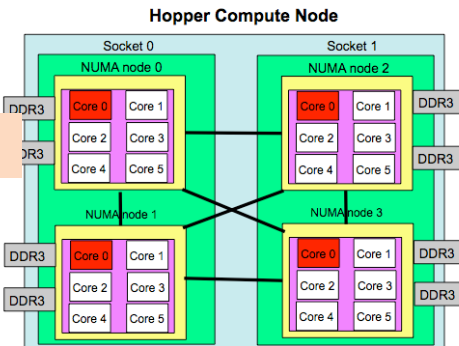
MPI Process Affinity: aprun "-S" Option

- **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.**

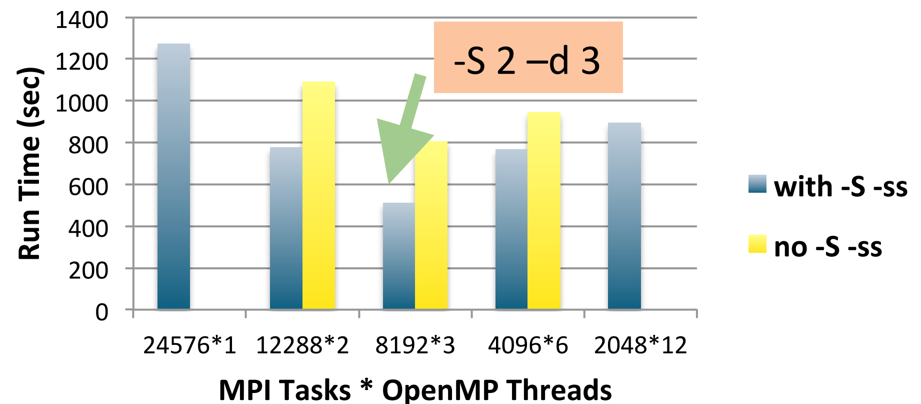
aprun –n 4 –d 6



aprun –n 4 **–S 1**–d 6



GTC Hybrid MPI/OpenMP on Hopper, 24,576 cores



Sweet spot is 8 MPI tasks per node, and 3 thread per MPI task

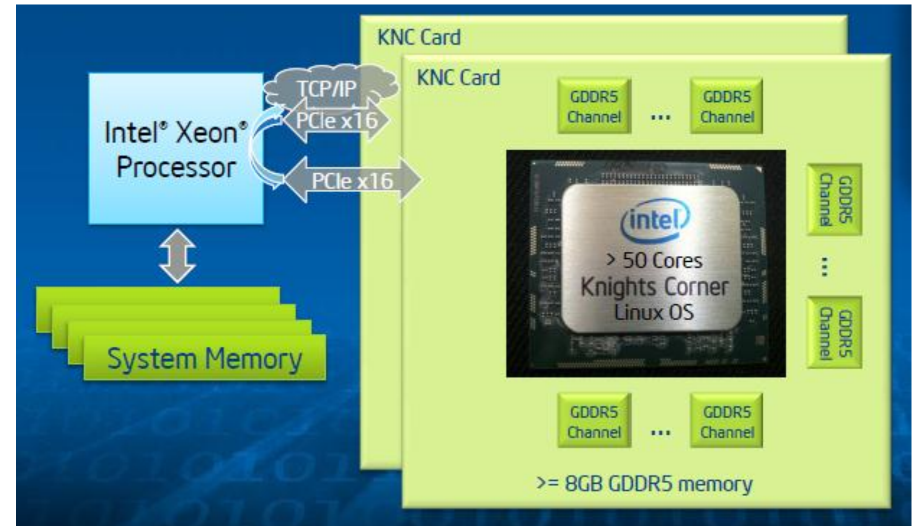
Thread Affinity: aprun "-cc" Option

- **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 40 YEARS at the FOREFRONT

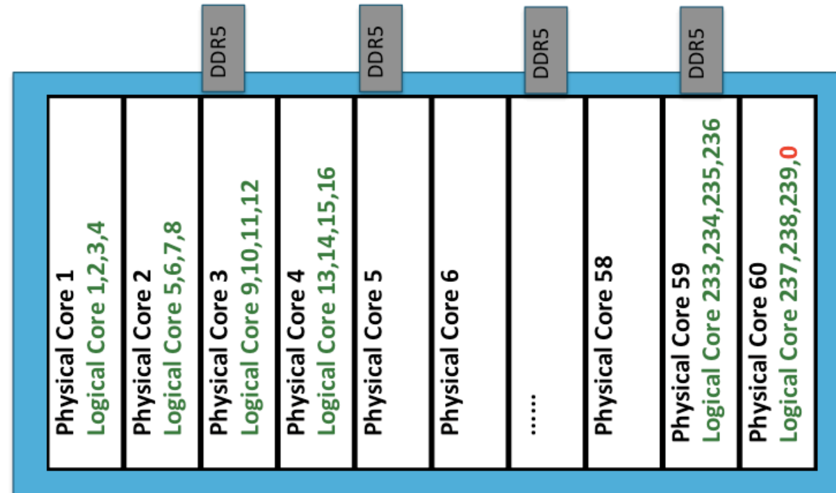
- **NERSC Intel Xeon Phi Knights Corner (KNC) testbed**
- **45 compute nodes, each has:**
  - Host node: 2 Intel Xeon Sandybridge processors, 8 cores each.
  - 2 MIC cards each has 60 native cores and 4 hardware threads per core.
  - MIC cards attached to host nodes via PCI-express.
  - 8 GB memory on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

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

Memory Affinity: "First Touch" Memory

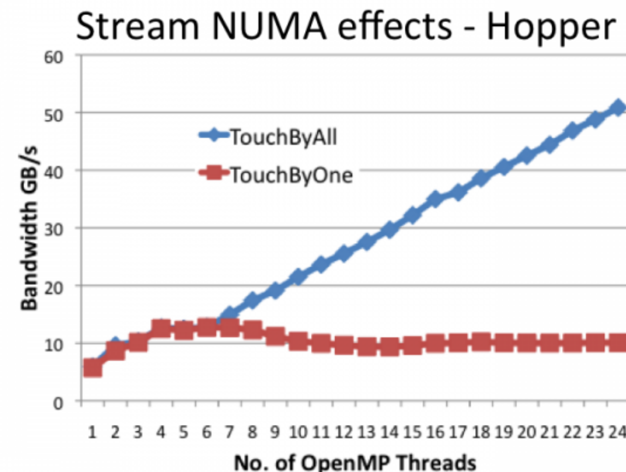
- **Memory affinity: allocate memory as close as possible to the core on which the task that requested the memory is running.**
- **Memory affinity is not decided by the memory allocation, but by the initialization. Memory will be local to the thread which initializes it. This is called "first touch" policy.**
- **Hard to do "perfect touch" for real applications. Instead, use number of threads few than number of cores per NUMA domain.**

Initialization

```
#pragma omp parallel for
for (j=0; j<VectorSize; j++) {
a[j] = 1.0; b[j] = 2.0; c[j] = 0.0;}
```

Compute

```
#pragma omp parallel for
for (j=0; j<VectorSize; j++) {
a[j]=b[j]+d*c[j];}
```



Courtesy of Hongzhang Shan, LBNL

Nested OpenMP

```
#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

Nested OpenMP

- **Beneficial to use nested OpenMP to allow more fine-grained thread parallelism.**
- **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
```

- **Refer to NERSC "Nested OpenMP" web page**
  - https://www.nersc.gov/users/computational-systems/edison/running-jobs/using-openmp-with-mpi/nested-openmp/

OpenMP 4 SIMD

A year ago:

- To get vector code, you had to use intrinsics, pray the compiler chose to vectorize a loop, or use compiler specific directives.

Today:

```
#pragma omp simd reduction(+:sum) aligned(a:64)
for(i=0; i<num; i++){
 a[i]=b[i]*c[i];
 sum=sum+a[i];
}
```

Warning: Using OpenMP 4 SIMD bypasses the compiler analysis

- Incorrect results possible!
- Poor performance possible!
- Memory errors possible!

Slide of Jack Deslippe, NERSC

OpenMP 4 SIMD

- **Parallelize and Vectorize:**
  - Fortran: !$OMP do simd *[clauses]*
  - The loop is first divided across a thread team, then subdivide loop chunks to fit a SIMD vector register.
- **SIMD Functions:**

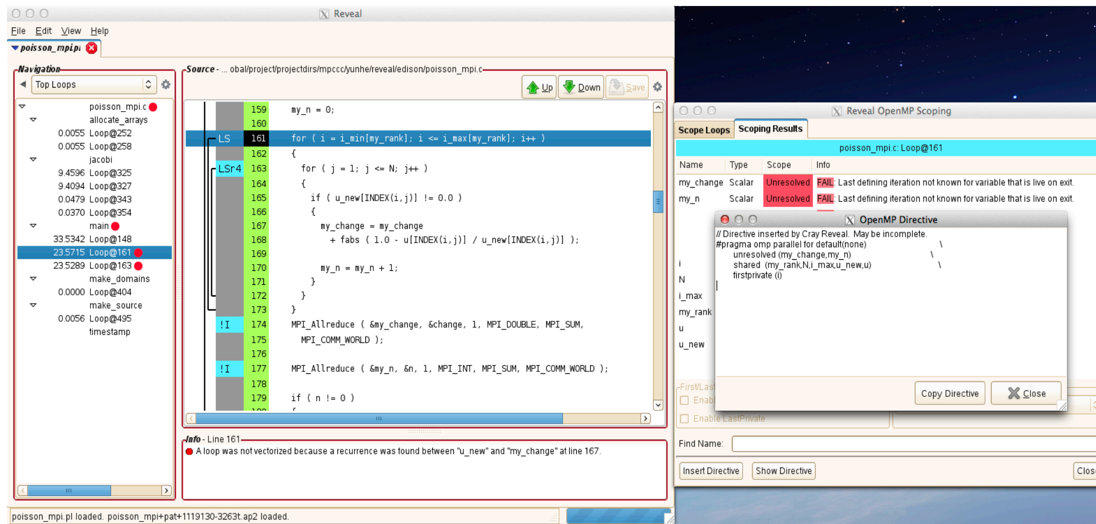
C/C++:

```
#pragma omp declare simd
float min (float a, float b) {
    return a<b ? a:b;
}
```

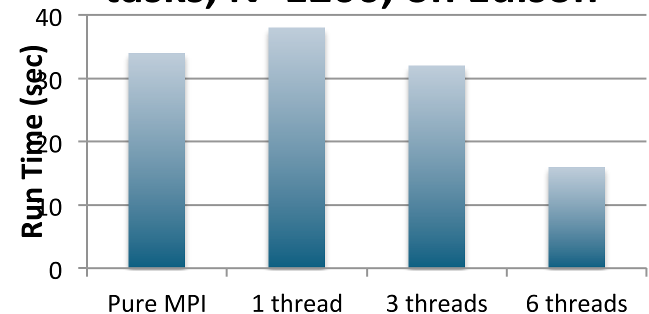
  - Compilers may not be able to vectorize and inline function calls easily.
  - Compilers #pramga declare simd tells compiler to generate SIMD function
  - Useful to use “declare simd” for elemental functions that are called from within a loop, so compilers can vectorize the function.

Adding OpenMP to Your Program

- **On Hopper/Edison, under Cray programming environment, Cray Reveal tool helps to perform scope analysis, and suggests OpenMP compiler directives to a pure MPI or serial code.**
  - Based on CrayPat performance analysis
  - Utilizes Cray compiler source code analysis and optimization information



poisson_mpi_omp, 4 MPI tasks, N=1200, on Edison



- **On Babbage, Intel Advisor tool helps to guide threading design options.**

Performance Analysis And Debugging

- **Performance Analysis**
  - Hopper/Edison:
    - Cray Performance Tools
    - VTune (on Edison)
    - IPM
    - Allinea MAP, perf-reports
    - TAU
  - Babbage:
    - VTune
    - Intel Trace Analyzer and Collector
    - HPCToolkit
    - Allinea MAP
- **Debugging**
  - Hopper/Edison: DDT, Totalview, LGDB, Valgrind
  - Babbage: Intel Inspector, GDB, DDT

Programming Tips for Adding OpenMP

- **Choose between fine grain or coarse grain parallelism implementation.**
- **Use profiling tools to find hotspots. Add OpenMP and check correctness incrementally.**
- **Parallelize outer loop and collapse loops if possible.**
- **Minimize shared variables, minimize barriers.**
- **Decide whether to overlap MPI communication with thread computation.**
  - Simplest and least error-prone way is to use MPI outside parallel region, and allow only master thread to communicate between MPI tasks.
  - Could use MPI inside parallel region with thread-safe MPI.
- **Consider OpenMP Tasking.**

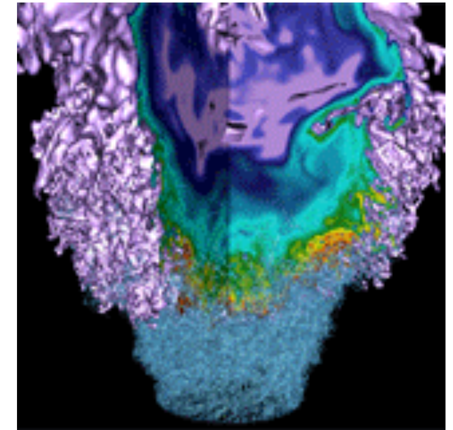
Why Hybrid MPI/OpenMP Code is Sometimes Slower Than Pure MPI?

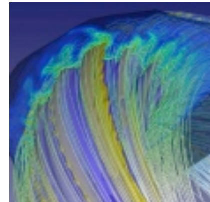
- **Serial code sections are not parallelized.**
- **Thread creation and synchronization overhead**
- **Cache coherence and false sharing.**
- **Data placement, NUMA effects.**
- **Natural one level parallelism problems.**
- **Not enough work for each thread.**
- **Load imbalance among threads.**
- **All threads are idle except one while MPI communication.**
  - Need overlap comp and comm for better performance.
  - Critical Section for shared variables.
- **Pure OpenMP code performs worse than pure MPI within node.**
- **Lack of optimized OpenMP compilers/libraries.**

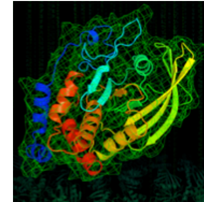
If a Routine Does Not Scale Well

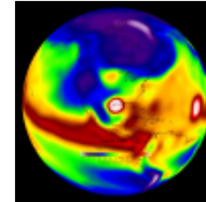
- Examine code for serial/critical sections, eliminate if possible.
- Reduce number of OpenMP parallel regions to reduce overhead costs.
- Perhaps loop collapse, loop fusion or loop permutation is required to give all threads enough work, and to optimize thread cache locality. Use NOWAIT clause if possible.
- Pay attention to load imbalance. If needed, try dynamic scheduling or implement own load balance scheme.
- Experiment with different combinations of MPI tasks and number of threads per task. Less MPI tasks may not saturate inter-node bandwidth.
- Test different process and thread affinity options.
- Leave some cores idle on purpose, for memory capacity or bandwidth capacity.
- Refer to Improving OpenMP Scaling web page:
  - https://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/improving-openmp-scaling

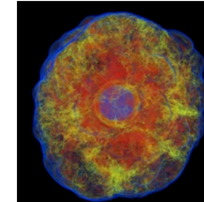
Case Studies

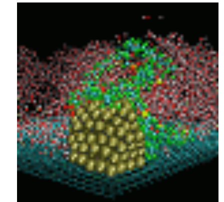










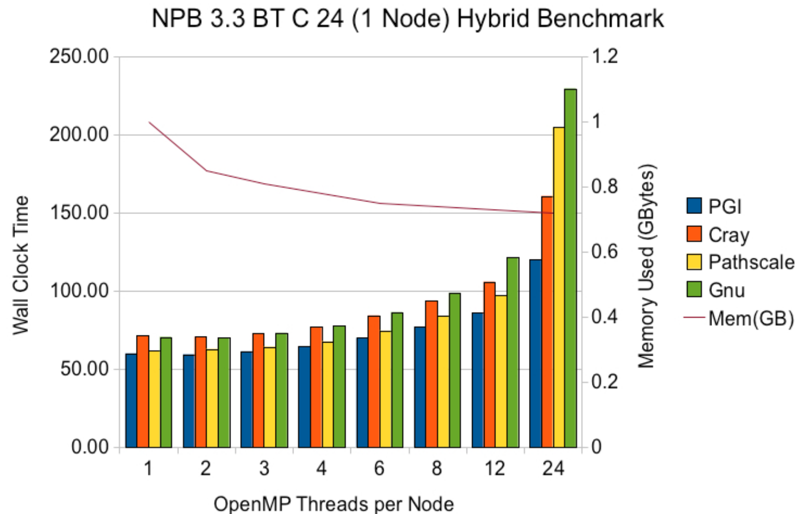


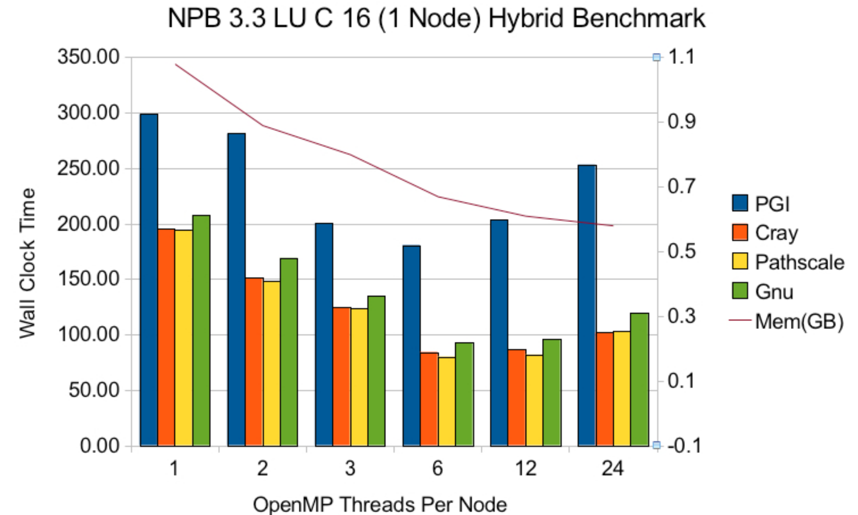
NERSC 40 YEARS at the FOREFRONT 1974-2014

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NPB: Hybrid MPI/OpenMP on Hopper





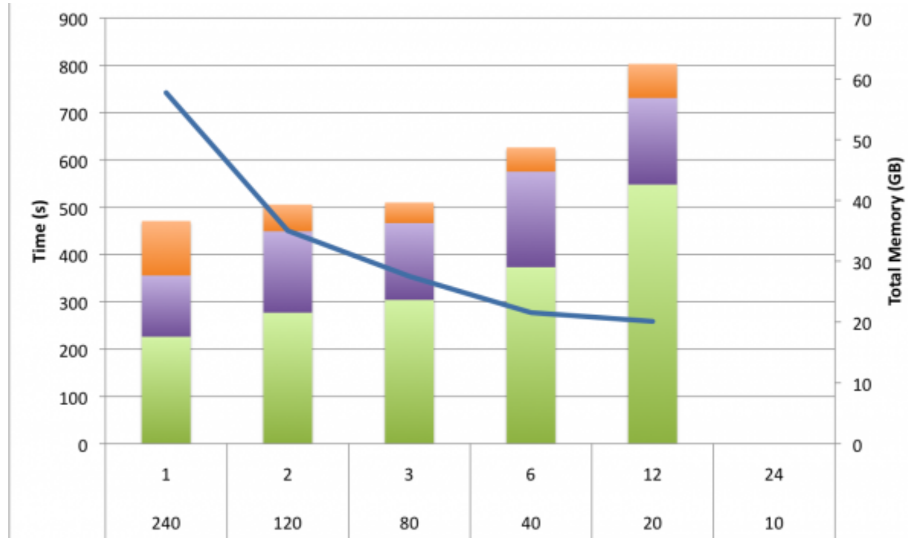
Courtesy of Mike Stewart, NERSC

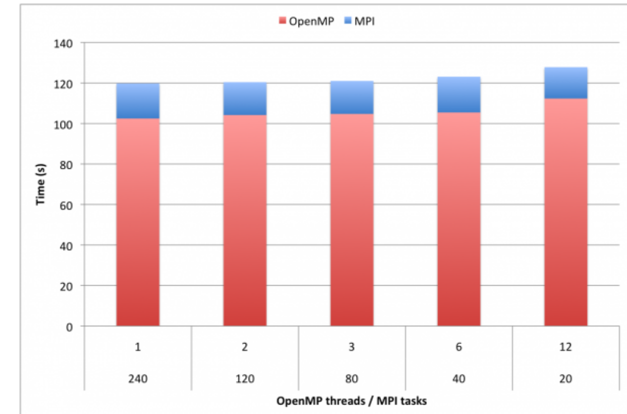
On a single node, hybrid MPI/OpenMP NAS Parallel Benchmarks:

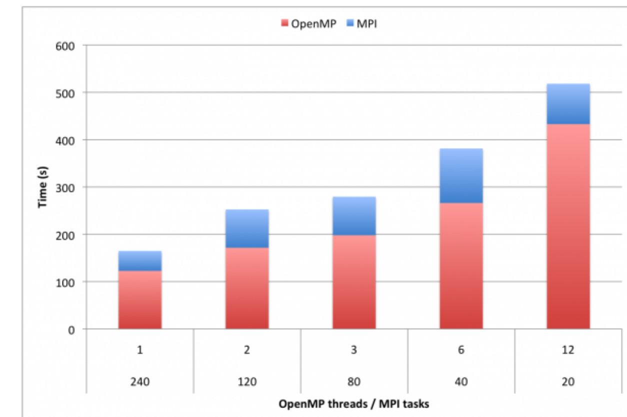
- **Reduced memory footprint with increased OpenMP threads.**
- **Hybrid MPI/OpenMP can be faster or comparable to pure MPI.**
- **Try different compilers.**
- **Sweet spot: BT: 1-3 threads; LU: 6 threads.**

fvCAM: Hybrid MPI/OpenMP on Hopper

NERSC 40 YEARS at the FOREFRONT







Community Atmospheric Model:

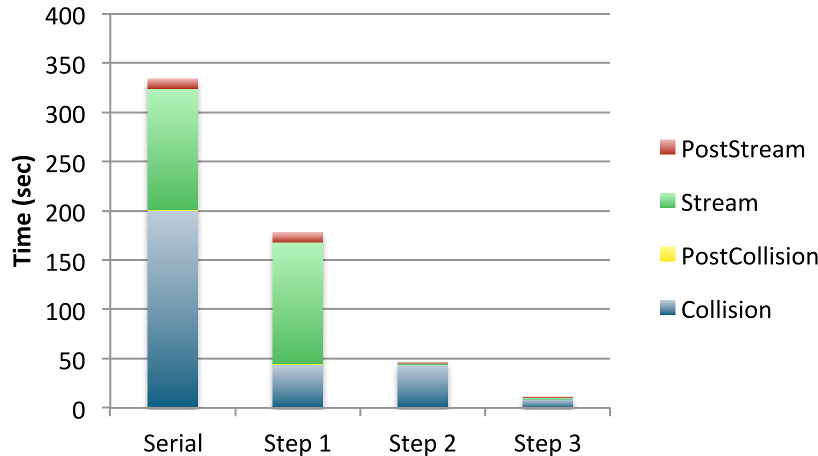
- **Memory reduces to 50% with 3 threads but only 6% performance drop.**
- **OpenMP time starts to grow from 6 threads.**
- **Load imbalance in "Dynamics" OpenMP**

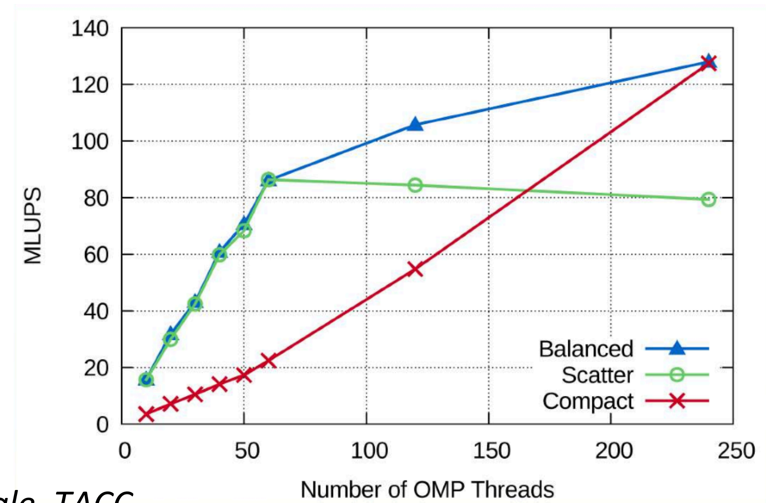
Courtesy of Nick Wright, et. al, NERSC/Cray Center of Excellence

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

LBM: Add OpenMP Incrementally

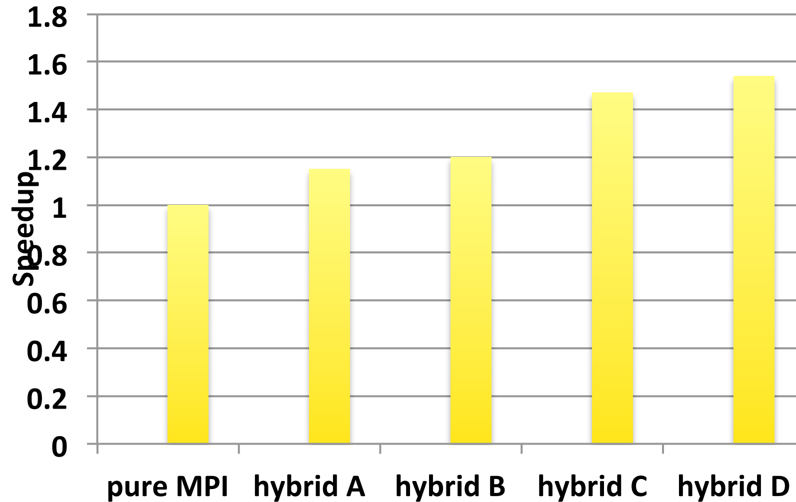




Courtesy of Carlos Rosale, TACC

- **Lattice Boltzmann Method: a Computational Fluid Dynamics Code.**
- **Actual serial run time for Collision > 2500 sec** ***(plotted above as 200 sec only for better display)*****, > 95% of total run time.**
- **Step 1: Add OpenMP to hotspot Collision. 60X Collision speedup.**
- **Step 2: Add OpenMP to the new bottleneck, Stream and others. 89X Stream speedup.**
- **Step 3: Add vectorization. 5X Collision speedup.**
- **Balanced provides best performance overall.**

MFDn: Overlap Comm and Comp



Courtesy of H. M. Aktulga et. al.

```
!$OMP PARALLEL
    if (my_thread_rank < 1) then
        call MPI_xxx(…)
    else
        do some computation
    endif
!$OMP END PARALLEL
```

- Need at least **MPI_THREAD_FUNNELED**.
- While master or single thread is making MPI calls, **other threads are computing**!
- Must be able to separate codes that can run before or after halo info is received. **Very hard**!
- Lose compiler optimizations.

- **MFDn: a nuclear physics code.**
- **Hopper. Pure MPI: 12,096 MPI tasks.**
- **Hybrid A: hybrid MPI/OpenMP, 2,016 MPI* 6 threads.**
- **Hybrid B: hybrid A, plus: merge MPI_Reduce and MPI_Scatter into MPI_Reduce_Scatter, and merge MPI_Gather and MPI_Bcast into MPI_Allgatherv.**
- **Hybrid C: Hybrid B, plus: overlap row-group communications with computation.**
- **Hybrid D: Hybrid C, plus: overlap (most) column-group communications with computation.**

MPAS-O: Tasking

- **MPAS-O model uses unstructured meshes, data stored in memory is unstructured. Next contiguous element in an array may not be a neighbor of the previous element. Elements are decomposed into blocks.**
- **Threaded Block Loops: OpenMP Tasking**

```
block => domain % blocklist

do while (associated(block))

   call compute_block(block)

   block => block % next

end do
```

```
block => domain % blocklist
do while (associated(block))
   block_d = block
   !$omp task
   firstprivate(block_d)
   call compute_block(block_b)
   !$omp end task
   block => block % next
end do
!$omp task wait
```

Courtesy of Douglas Jacobsen et. al., NCAR Multi-Core 2015 Workshop

MPAS-O: Threaded Element Loops

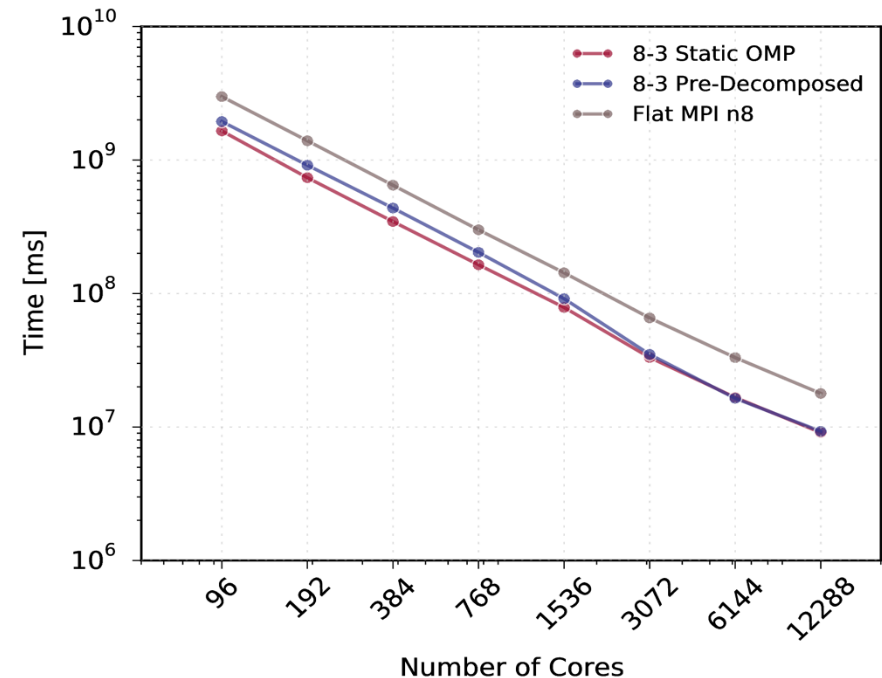
- **Use Pre-computed decompositions (SPMD)**

```
eleStart = get_ele_start(iThread)
eleEnd = get_ele_end(iThread)

do iElement = eleStart, eleEnd
   … compute on elements ...
end do
```

- **Use OpenMP Directives (loop parallelism)**

```
!$omp do private(...)
do iElement = 1, nElements
   … compute on elements ...
end do
!$omp end do
```

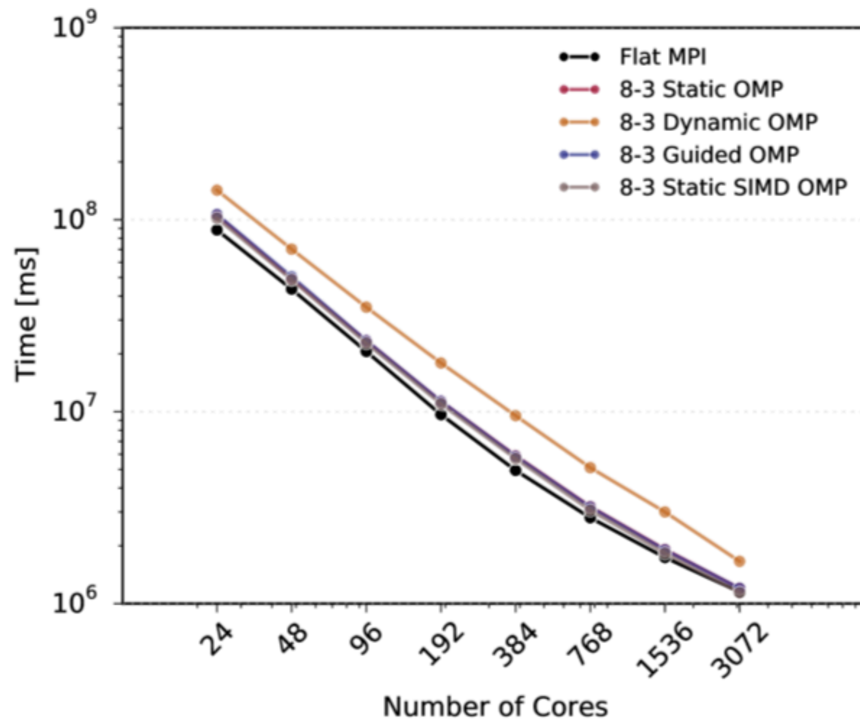


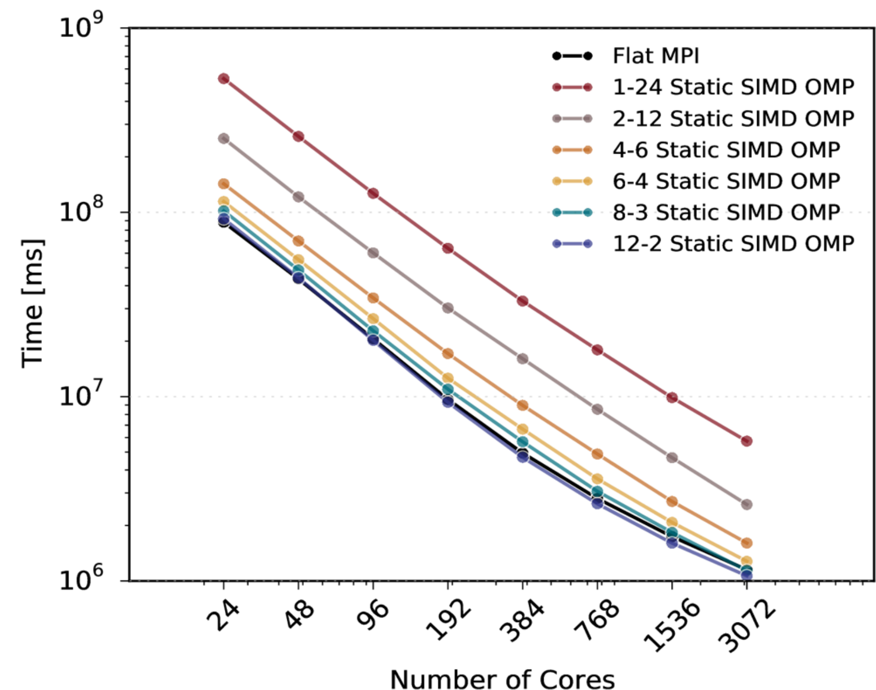
- **Loop parallelism better than SPMD**
- **Both better than pure MPI**

Courtesy of Douglas Jacobsen et. al., NCAR Multi-Core 2015 Workshop

U.S. DEPARTMENT OF ENERGY | Office of Science
BERKELEY LAB
NERSC 40 YEARS at the FOREFRONT

MPAS-O: Compare Schedules and SIMD

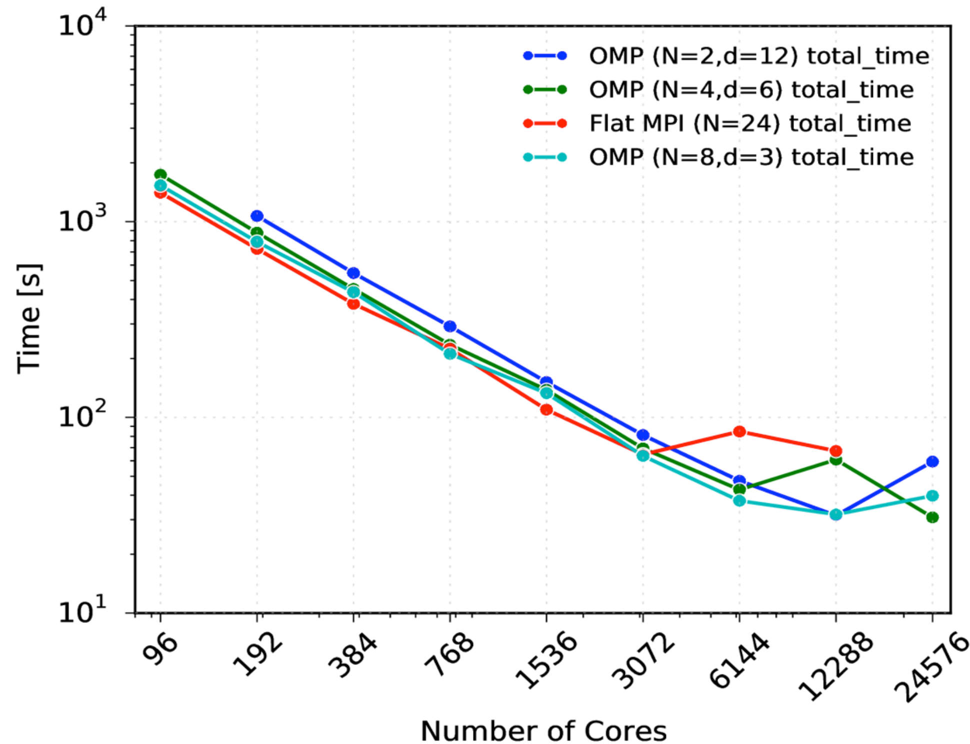




- **Good to explore different OpenMP schedules**
- **Good to experiment with different combinations of MPI tasks and OpenMP threads to find a sweet spot. "2-12 Static SIMD OMP" is the best in this case.**
- **SIMD directive helps a little, to vectorize loops compilers can not auto-vectorize.**

Courtesy of Douglas Jacobsen et. al., NCAR Multi-Core 2015 Workshop

MPAS-O: Strong Scaling with Full Code



- **OpenMP helps scaling for larger core counts**
- **"OMP (N=8,d=3)" is the best in this case**

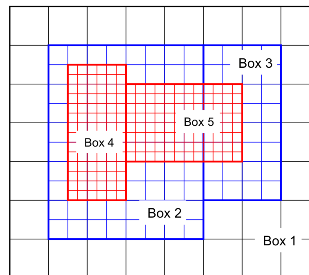
Courtesy of Douglas Jacobsen et. al., NCAR Multi-Core 2015 Workshop

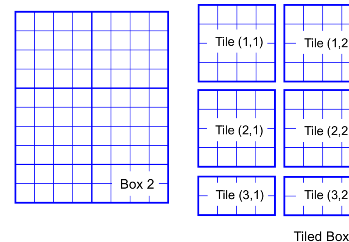
BoxLib: Tiling Threading Model

- **Traditional threading model**
  - Domain decomposed into *N* boxes, distributed among *M* MPI tasks, each with *m* threads.
  - Load imbalance especially in AMR applications due to uneven workload among MPI tasks
  - Fine grain OpenMP
- **Tiling threading model**
  - Iteration space within each box is divided into smaller "tiles", which are distributed among threads
  - Better load balancing
  - Tile size can be tuned for optimal cache reuse.
  - Coarse grain OpenMP

Many such regions as below:

```
!$OMP parallel do private (i)
do j=1,15
    do i =1,16
    … some work here …
    end do
end do
!$OMP end parallel do
```

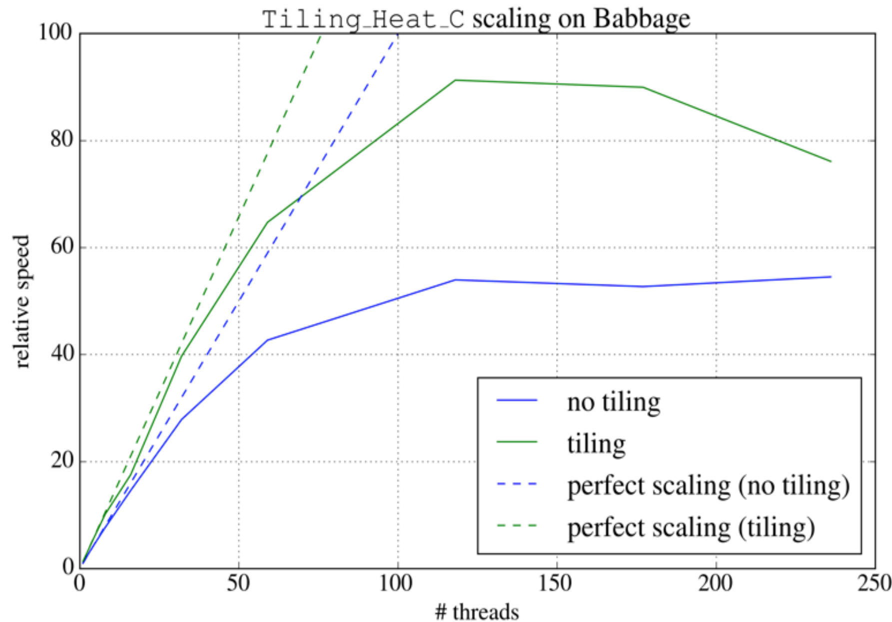





```
!$OMP parallel
loop over tiles
    get tile box
    … some work here …
end loop over tiles
!$OMP end parallel
```

Courtesy of Brian Friesen, NERSC and Jessica Kawana, Williamette University

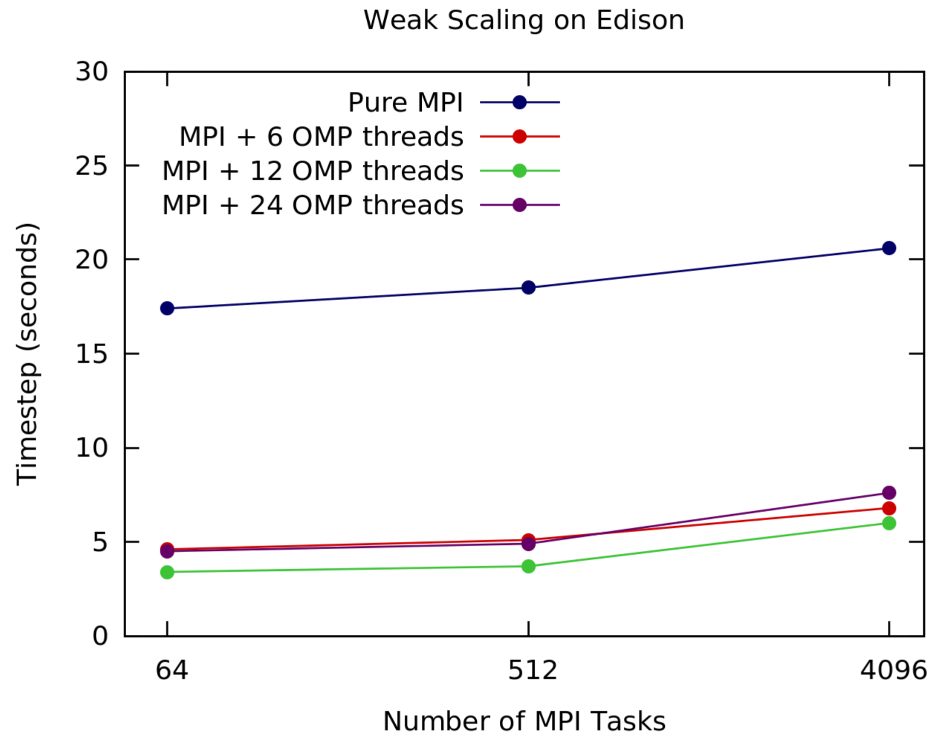
BoxLib: OpenMP Scaling with Tiling



- **Tiling implementation strong scales efficiently up to ~120 threads on Babbage.**
- **For simple operators, tiling is always a win.**
- **For complicated operators (e.g., low Mach number flows), results are mixed.**

Courtesy of Brian Frieson., NERSC

BoxLib: Hybrid MPI/OpenMP Scaling with Tiling



- **Best performance with MPI+12 OpenMP threads**

Courtesy of Andrew Nonaka, LBNL

XGC1: "-heap-arrays 64" Compiler Flag

NERSC 40 YEARS at the FOREFRONT

- This Intel compiler flag puts automatic arrays and temp of size 64 kbytes or larger on heap instead of stack.
- Surprisingly it slows down kernels by >6X.
- Allocation and access of private copies on the heap are very expensive.
- Does not affect explicit-shape arrays.
- Remove this flag, and set OMP_STACKSIZE to a large value: run time improves from 348 sec to 43 sec.
- Alternative: use !$OMP THREADPRIVATE.
- An Intel compiler bug has been filed to fix the slowness caused by the compiler flag.

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.

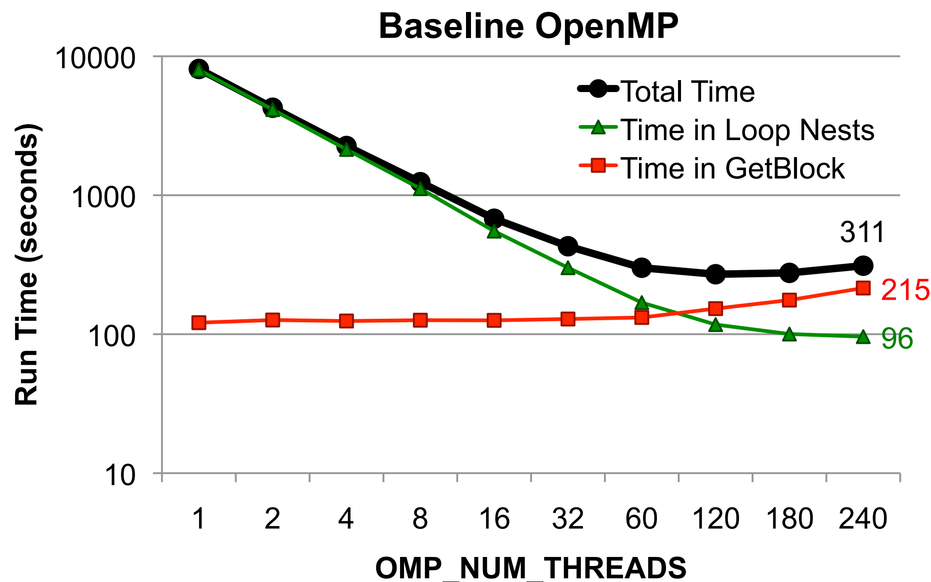
```
export OMP_NUM_THREADS=24
export OMP_NESTED=TRUE
export OMP_STACKSIZE=8000000
aprun -n 200 -d 24 -N 2 -S 1 -j 2 -cc numa_node ./xgca
```

- **Will try:**

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

- **Use num_threads clause in source code to set threads for regions not using the same number of threads for most other regions. For other regions, use OMP_NUM_THREADS env for simplicity and flexibility.**

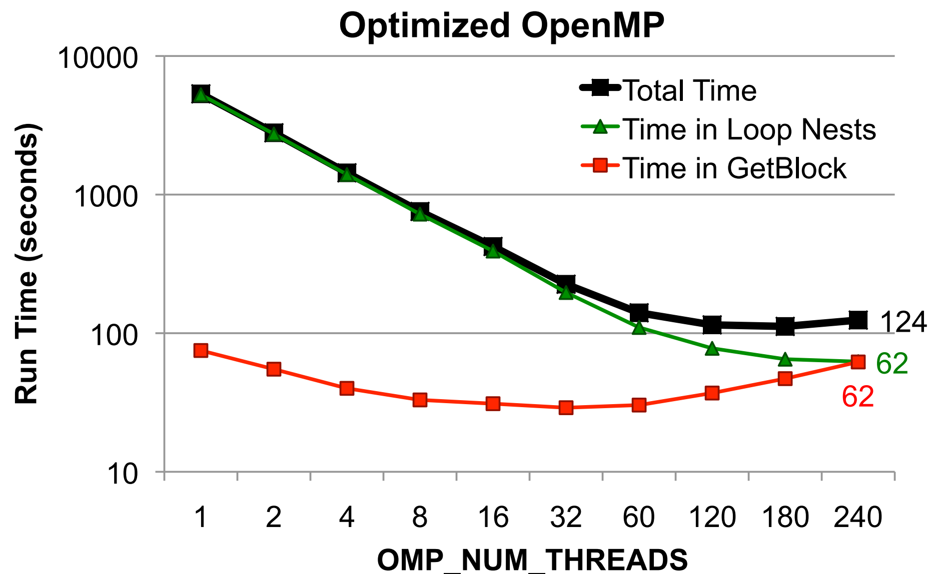
NWChem CCSD(T): Baseline OpenMP



- Due to memory limitation, can only run with 1 MPI process per MIC.
- OpenMP added at the outermost loops of hotspots: Loop Nests. Scales well up to 120 threads.
- GetBlock is not parallelized with OpenMP. Hyper-threading hurts performance.
- Total time has perfect scaling from 1 to 16 threads. Best time at 120 threads.
- Balanced affinity gives best performance.

Courtesy of Hongzhang Shan et al., LBNL

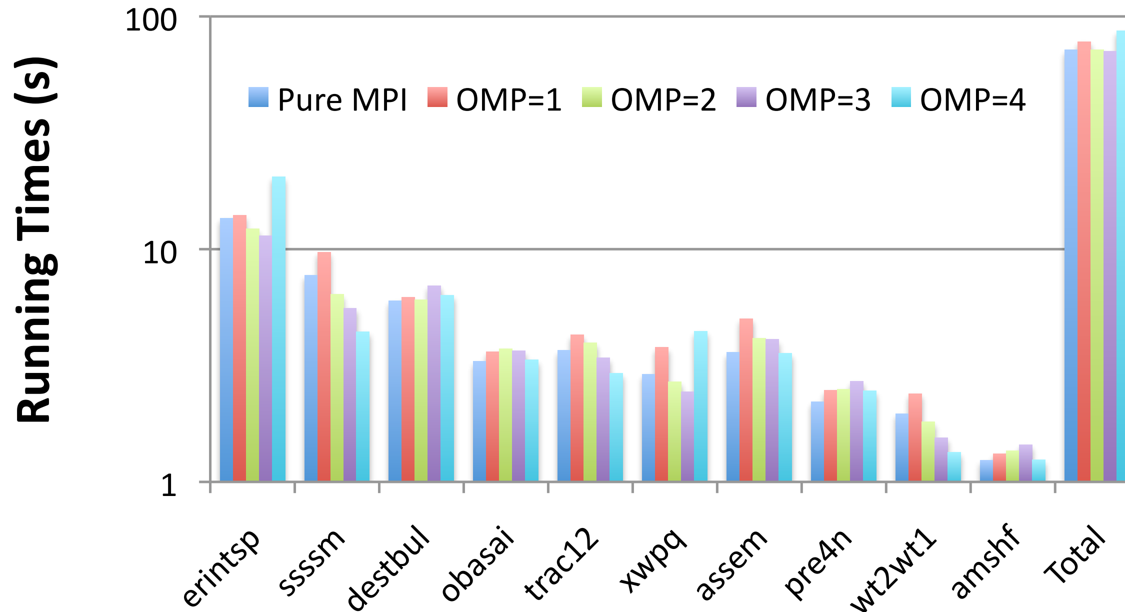
NWChem CCSD(T): OpenMP Optimizations



- GetBlock optimizations: parallelize sort, loop unrolling.
- Reorder array indices to match loop indices.
- Merge adjacent loop indices to increase number of iterations.
- Align arrays to 64 bytes boundary.
- Exploit OpenMP loop control directive, provide complier hints.
- Total speedup from base is 2.3x.

Courtesy of Hongzhang Shan et al., LBNL

NWChem FMC: Add OpenMP to HotSpots (OpenMP #1)



- Total number of MPI ranks=60; OMP=N means N threads per MPI rank.
- Original code uses a shared global task counter to deal with dynamic load balancing with MPI ranks
- Loop parallelize top 10 routines in TEXAS package (75% of total CPU time) with OpenMP. Has load-imbalance.
- OMP=1 has overhead over pure MPI.
- OMP=2 has overall best performance in many routines.

Courtesy of Hongzhang Shan et al., LBNL

NERSC 40 YEARS at the FOREFRONT
U.S. DEPARTMENT OF ENERGY | Office of Science
BERKELEY LAB

NWChem FMC: OpenMP Task Implementation (OpenMP #3)

Fock Matrix Construction — OpenMP Task Implementation

```
c$OMP parallel
myfock() = 0
c$OMP master
current_task_id = 0
mytid = omp_get_thread_num()
My_task = global_task_counter(task_block_size)
for ijkl = 2*ntype to 2 step −1 do
    for ij = min(ntype, ijkl − 1) to max(1, ijkl − ntype) step −1 do
        kl = ijkl − ij
        if (my_task .eq. current_task_id) then
            c$OMP task firstprivate(ij,kl) default(shared)
            create_task(ij,kl, ...)
            c$OMP end task
            my_task=global_task_counter(task_block_size)
        end if
        current_task_id = current_task_id + 1
    end for
end for
c$OMP end master
c$OMP taskwait
c$OMP end parallel
Perform Reduction on myfock to Fock matrix
```

- OpenMP task model is flexible and powerful.
- The task directive defines an explicit task.
- Threads share work from all tasks in the task pool.
- Master thread creates tasks.
- The taskwait directive makes sure all child tasks created for the current task finish.
- Helps to improve load balance.

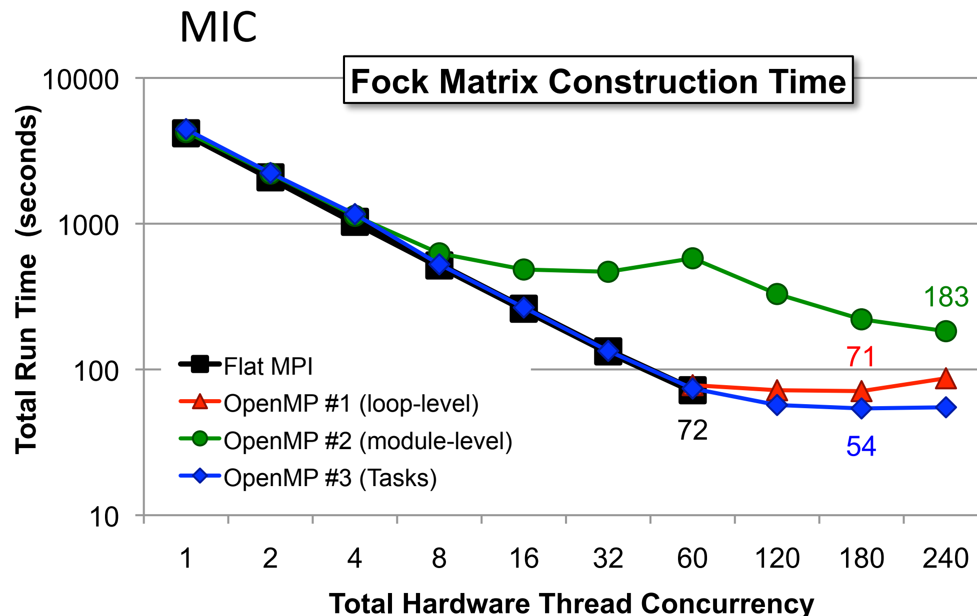
- **Use OpenMP tasks.**
- **To avoid two threads updating Fock matrix simultaneously, a local copy is used per thread. Reduction at the end.**

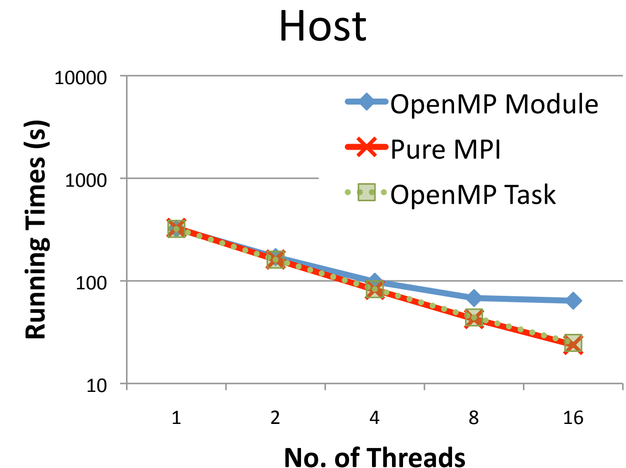
U.S. DEPARTMENT OF ENERGY | Office of Science

Courtesy of Hongzhang Shan et al., LBNL

BERKELEY LAB

NWChem FMC: Various OpenMP Optimizations

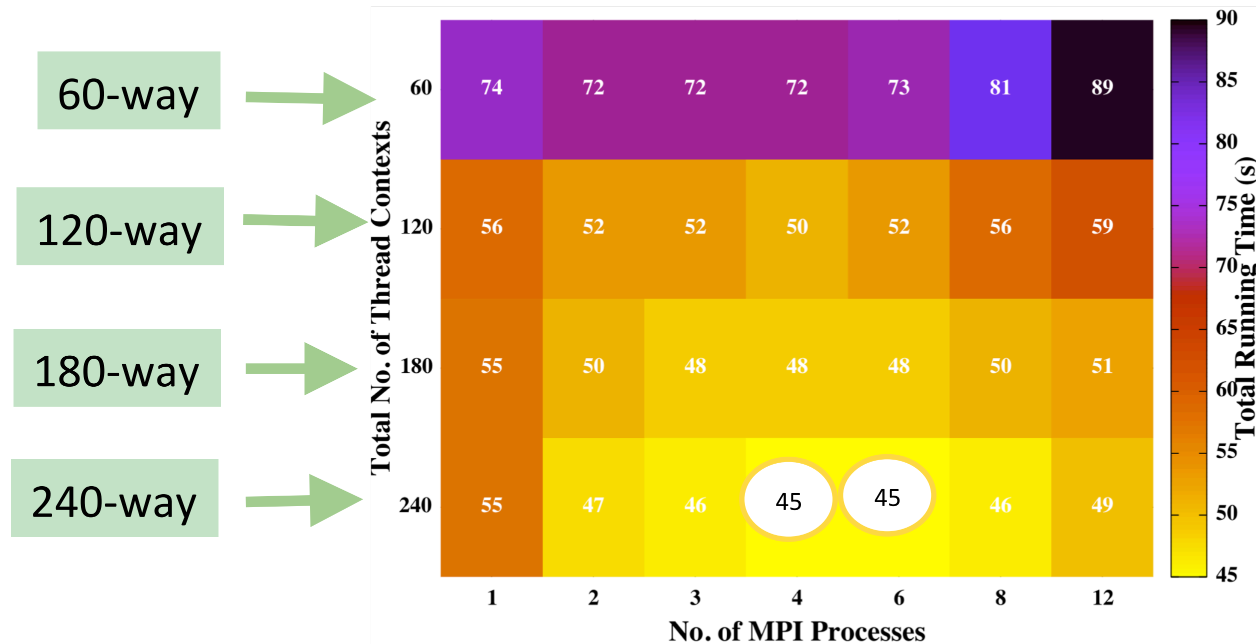




- Flat MPI is limited to a total of 60 ranks due to memory limitation.
- OpenMP #1 uses flat MPI up to 60 MPI processes, then uses 2, 3, and 4 threads per MPI rank.
- OpenMP #2 and #3 are pure OpenMP.
- OpenMP #2 module-level parallelism saturates at 8 threads (critical and reduction related). Then when over 60 threads, hyper-threading helps.
- OpenMP #3 Task implementation continues to scale over 60 cores. 1.33x faster (with 180 threads) than pure MPI.
- The OpenMP Task implementation benefits both MIC and Host.

Courtesy of Hongzhang Shan et al., LBNL

NWChem FMC: MPI/OpenMP Scaling and Tuning



- Another way of showing scaling analysis result.
- Sweet spot is either 4 MPI tasks with 60 OpenMP threads per task, or 6 MPI tasks with 40 OpenMP threads per task.
- 1.64x faster than original flat MPI.
- 22% faster than 60 MPI tasks with 4 OpenMP threads per task.

Courtesy of Hongzhang Shan et al., LBNL

NWChem: OpenMP “Reduce” Algorithm

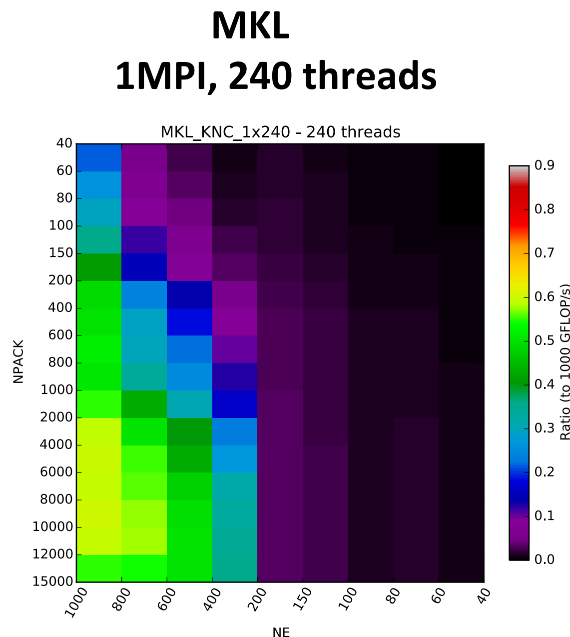
- **Plane wave Lagrange multiplier**
  - Many matrix multiplications of complex numbers, C = A x B
  - Smaller matrix products: FFM, typical size 100 x 10,000 x 100
  - Original threading scaling with MKL not satisfactory
- **OpenMP “Reduce” or “Block” algorithm**
  - Distribute work on A and B along 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

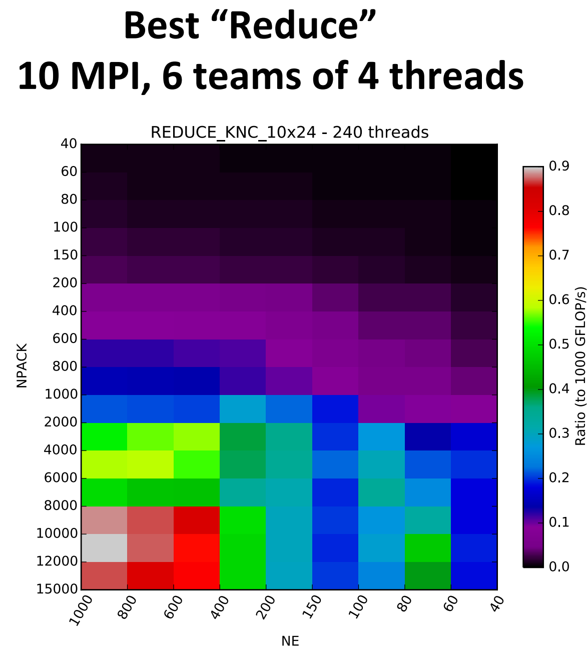
NWChem: OpenMP "Reduce" Algorithm

- 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

MKL
1MPI, 240 threads



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



Courtesy of Mathias Jacquelin, LBNL

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
```

“OMP target device” Works on Babbage

```
program test
use omp_lib
write(*,*) 'cpu max threads:',omp_get_max_threads()
!$omp target device(0)
write(*,*) 'mic max threads:',omp_get_max_threads()
!$omp parallel
!$omp master
write(*,*) 'mic nbr threads:',omp_get_num_threads()
!$omp end master
!$omp end parallel
!$omp end target

!$omp target device(0)
!$omp teams num_teams(1)
write(*,*) 'team', omp_get_team_num(), ' mic max
threads:',omp_get_max_threads()
!$omp parallel
!$omp master
write(*,*) 'team',omp_get_team_num(),' mic nbr
threads:',omp_get_num_threads()
!$omp end master
!$omp end parallel
!$omp end teams
!$omp end target
end program test
```

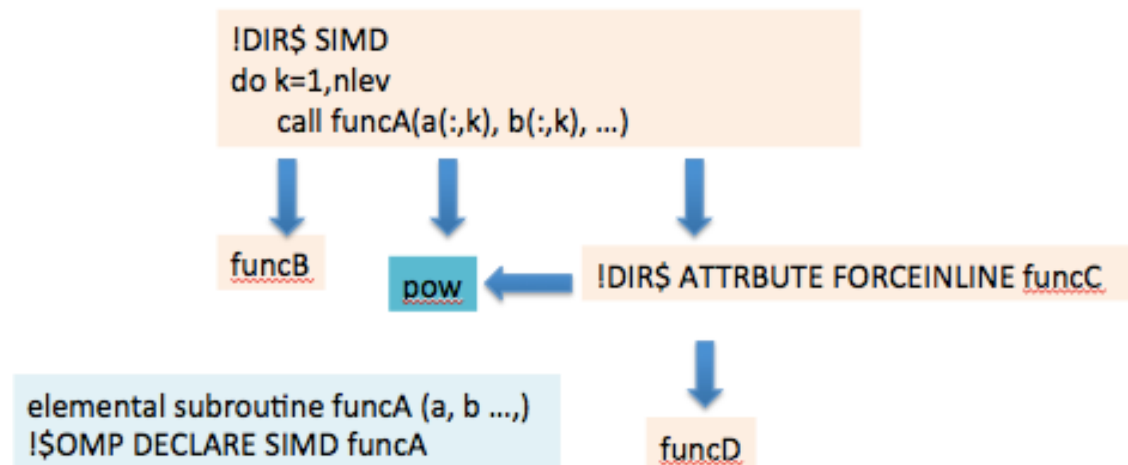
```
export KMP_AFFINITY=balanced
export OMP_NUM_THREADS=1
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=60
```

```
% cat myjob.host.2680.out
cpu max threads: 1
mic max threads: 60
mic nbr threads: 60
team 0 mic max threads: 60
team 0 mic nbr threads: 236
```

Not recommended for preparing for Cori, but it is good to know that it works ☺

CESM MG2: Vectorization Prototype

- **Use compiler report to check and make sure key functions are vectorized (and all functions on the call stack are vectorized too)**
  - Elemental functions need to be inlined
  - "-qopt-report=5" reports highest level of details.
  - "-ipo" is needed if functions are in different source codes.
- **Add !$OMP DECLARE SIMD and !DIR$ ATTRIBUTE FORCEINLINE when needed.**



CESM MG2: OMP SIMD ALIGNED

- **Align data on specific byte boundaries; directive based approach with OMP directive:**
  - Portable solution: !$OMP SIMD ALIGNED (….)
    - Tells the compiler that the arrays are aligned
    - Asserts that there are no dependencies
    - Requires to use PRIVATE or REDUCTION clauses to ensure correctness
    - Forces the compiler to vectorize, whether or not it thinks if it is a good idea or not
  - As compared to: !DIR$ VECTOR ALIGNED
    - Tells the compiler that the arrays are aligned
    - Intel compiler specific, not portable
- **!$OMP SIMD ALIGNED is independent of vendor, however it can be overly intrusive in code**

CESM MG2: OMP SIMD ALIGNED

- **Using the "ALIGNED" attribute achieved 8% performance gain when the list is explicitly provided.**
- **However, the process is tedious and error-prone, and often times impossible in large real applications.**
  - !$OMP SIMD ALIGNED added in 48 loops in MG2 kernel *(by Christopher Kerr)*, many with list of 10+ variables

!$OMP SIMD ALIGNED	!$OMP SIMD	!dir$ VECTOR ALIGNED	-align array64byte	-openmp	Time per iteration (usec) on Edison
x			x	x	**444**
x				x	446
	x		x	x	**484**
	x			x	482
		x	x		452
		x			456
					473

U.S. DEPARTMENT OF ENERGY | Office of Science
NERSC 40 YEARS at the FOREFRONT
BERKELEY LAB

CESM MG2: OMP SIMD ALIGNED

- **How can compilers know better which arrays are aligned so users do not have to specify?**
  - A variable can be declared as aligned
  - A variable can be set to aligned with a compiler flag
  - When in scope, hopefully complier should know
- **Inquired with Fortran Standard:**
  - Equivalent of "!$DIR ATTRIBUTES ALIGNED: 64 :: A"
    - C/C++ standard: float A[1000] __attribute__((aligned(64)));
    - Not in Fortran standard yet
  - Equivalent of the "-align array64byte" compiler flag
    - Exist in Intel (Fortran only) and Cray compilers
    - What about other compilers?

Srinath Vadlamani's testSIMD Suite

Compiler and language options	Run Time
None	**4.05**
-xavx	**3.29**
!$omp declare simd(init)	40.02
!$omp declare simd(init) uniform(n)	40.00
!$omp declare simd(init) simdlen(4) uniform(n)	37.83
!$omp declare simd(init) simdlen(4)	37.71
!$omp declare simd(init) **aligned(a:32)**	**4.26**
!$omp declare simd(init) **aligned(a:32)** uniform(n)	4.30
!$omp declare simd(init) simdlen(4**) aligned(a:32)**	4.26
!$omp declare simd(init) simdlen(4) **aligned(a:32)** uniform(n)	4.28

- **Python script to test which SIMD options are able to get close to AVX performance.**
- **Tests ran on Edison. Use "ifort" native compiler (15.0.1.133), default "-O2" optimization: not completely "–no-vec".**
- **"aligned" is essential to get good performance. Although none is as good as "-xavx".**

SIMD ALIGNED Restrictions

NERSC 40 YEARS at the FOREFRONT

- **Restriction for "OMP SIMD ALIGNED" in Fortran: "The type of list items appearing in the aligned clause must be C_PTR or Cray pointer, or the list item must have the POINTER or ALLOCATABLE attribute."**
- **Could this be relaxed to allow aligned static arrays to be included in the list? A static array can be specified as aligned either with compiler directives or by using alignment compiler flags (in certain compiler implementations).**
- **We have a kernel code written in Fortran, with a manually specified aligned list.**
  - One compiler could not compile since it conforms to the specification.
  - Another compiler compiles it anyway, and achieves speedup.

Users Like to Have

- **Threaded libraries**
  - MKL has threaded libraries. Has mechanisms to use with 1 or more thrds.
  - Users need other thread-safe libraries, such as PETSc.
  - Should we work with libraries developers to always inquiry if they are in threaded region first? (must be OpenMP aware at run time)
  - Or is it better for libraries to provide separate APIs for single-thread and multi-thread versions?
- **More consistent behaviors**
  - Example: default OMP_MAX_ACTIVE_LEVELS
  - Example: default OMP_NUM_THREADS
    - Had discussions on whether set to 1 or max
    - In some situations, it is better to set to 1, e.g. library calls within threaded region for thread safety
    - Most cases compilers are already using max available, however hard to reach an agreement on what is the max available cores (and hardware threads), especially when MPI affinity choices are considered. Possible oversubscription.
    - Decided to ask users to always set environment variable OMP_NUM_THREADS explicitly ☺

Summary

- OpenMP is a fun and powerful language for shared memory programming.
- Hybrid MPI/OpenMP is recommended for many next generation architectures (Intel Xeon Phi for example), including NERSC-8 system, Cori.
- Keep portability in mind, use portable programming models.
- Optimizations targeted for one architecture can help performance for other architectures.
- Keep promoting and working with users on OpenMP usage at NERSC.

Thank you.