Running Large Scale Jobs on a Cray XE6 System

Yun (Helen) He and Katie Antypas,
NERSC/LBNL

Cray User Group Meeting, May 2012

U.S. DEPARTMENT OF ENERGY | Office of Science

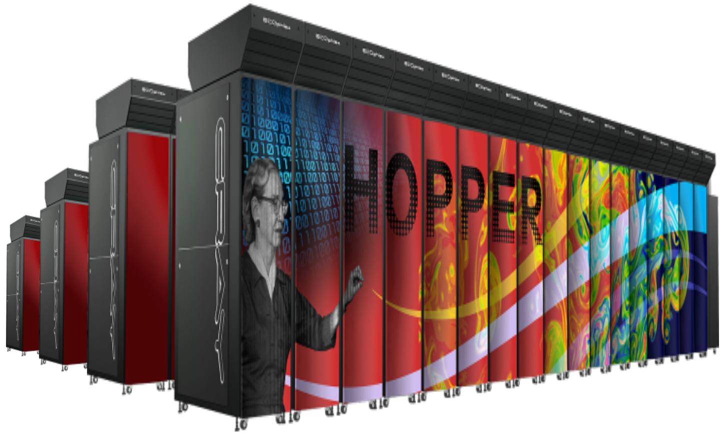
NERSC National Energy Research Scientific Computing Center

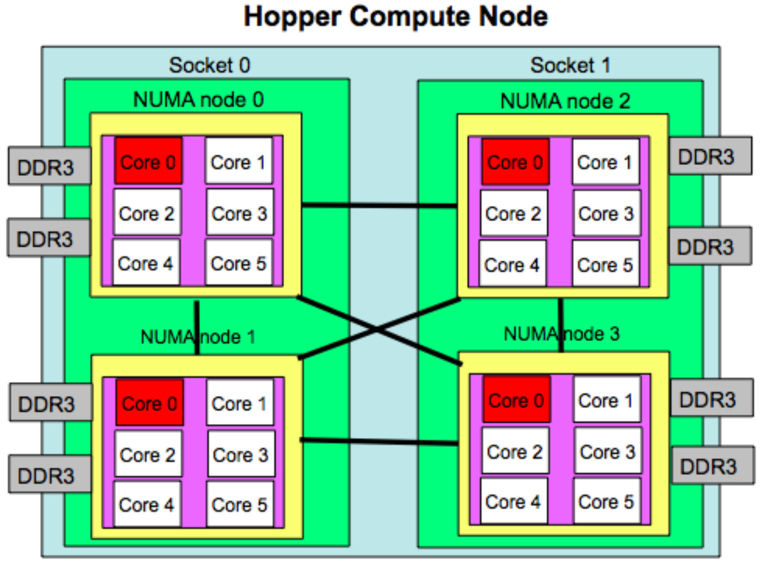
BERKELEY LAB Lawrence Berkeley National Laboratory

Outline

- Hopper Introduction
- NERSC User Survey on Running Large Jobs
- Successful Tuning Stories
- System Issues Affecting Large Jobs
  - Huge Pages Issue
  - Hung Jobs issue
- IO Tunings
- Summary

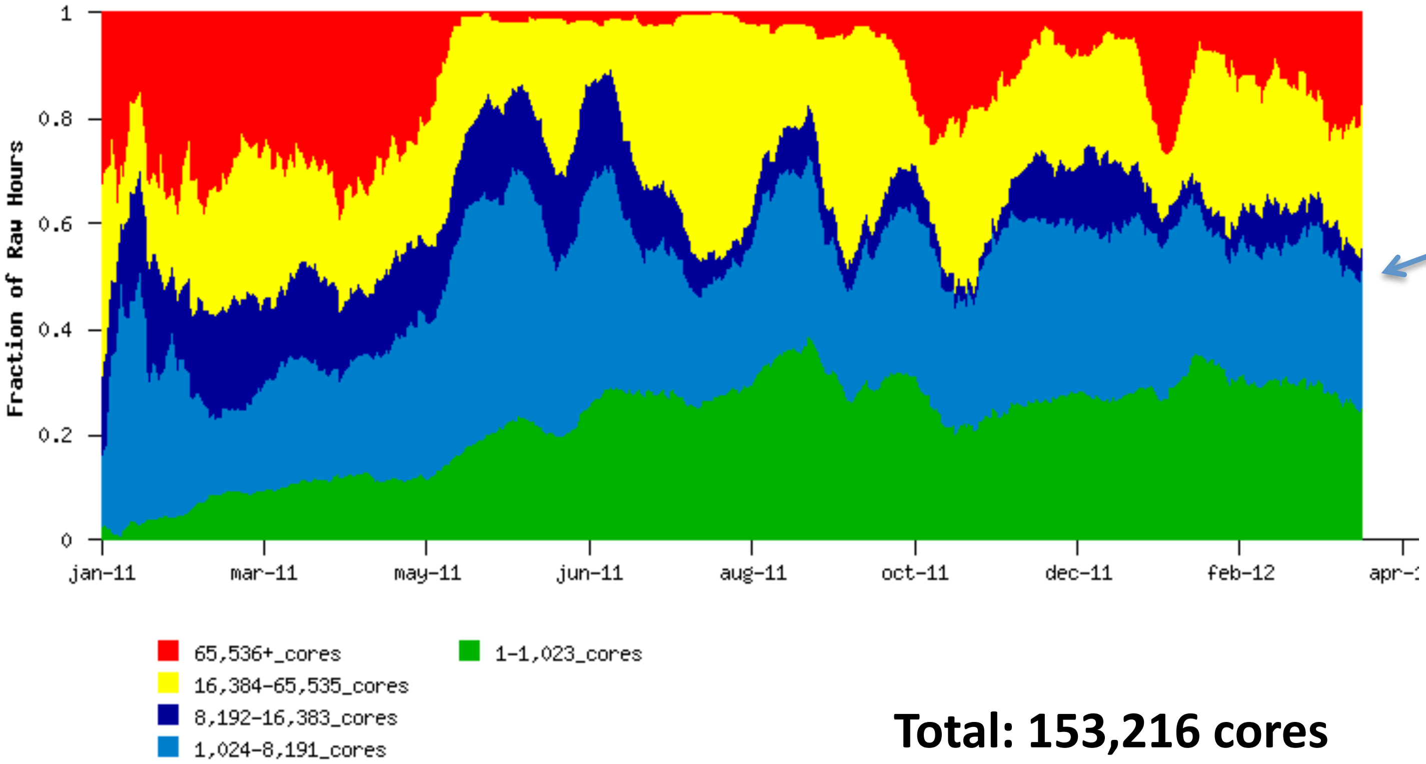
Hopper Configuration





- NERSC Cray XE6, 6,384 nodes, 153,126 cores.
- 1.28 PFlops/peak,~140 Tflops/sustained. 212 TB memory.
- Each node has 2 twelve-core AMD MagnyCours 2.1 GHz procs (2 sockets)
- 4 NUMA nodes per node, 6 cores per NUMA node.
- 32 GB per node, 1.33 GB/core. (some 64 GB, 2.67 GB/core)

Hopper Job Sizes Breakdown



8,192+

Total: 153,216 cores

- **About 40% of total compute nodes are used by jobs over 8k cores.**
- **More large jobs before charging started on May 1, 2011.**

User Survey on Running Large Jobs

- Sent to 50 users who run 682+ node jobs (16,368+ cores) routinely. Over 1/3 responded.
- Typical job sizes, code name and science area
  - From single node to the entire machine.
- Challenges faced
  - Biggest challenge is getting through the queue
- Tuning options used
  - Compilers and flags
    - They try various compilers! Many like gcc, some cray, some PGI.
  - Change default MPI env
    - Mostly not needed, unlike in XT with Portals.
  - Change default MPI ranking
    - Yes, simple yet effective

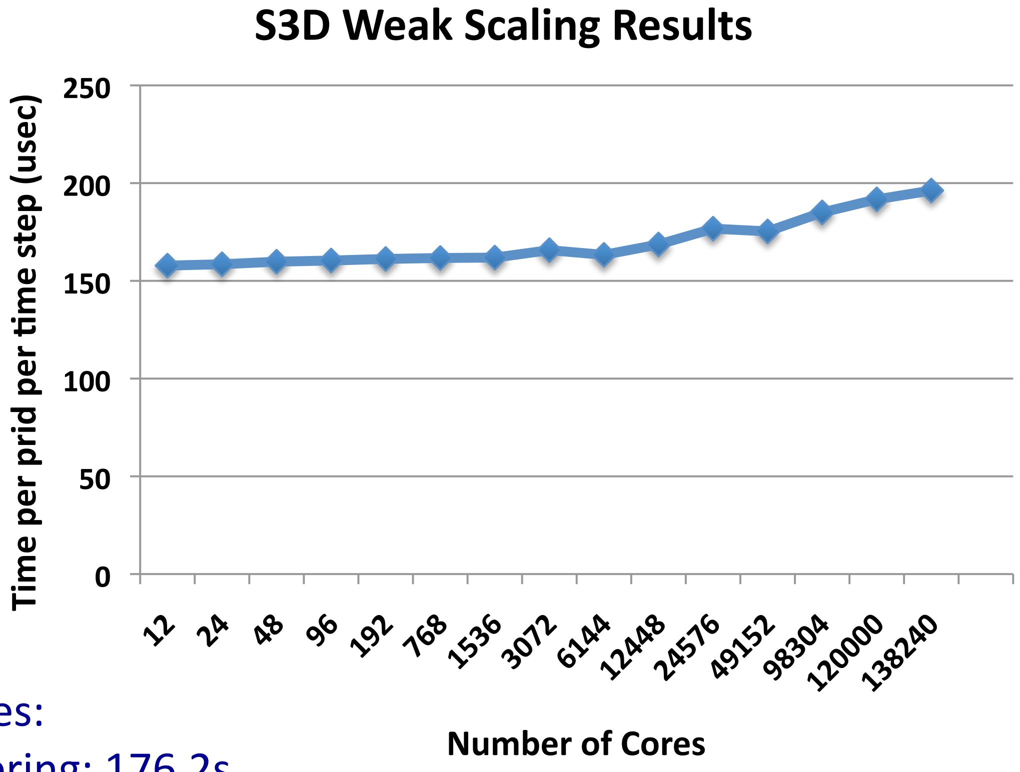
User Survey on Running Large Jobs (2)

- Tuning Options Used (cont'd)
  - Hybrid MPI/OpenMP. number of threads, "first touch"?
    - Mostly use 4 MPI tasks, 6 thread per node as suggested.
  - Advanced aprun options, affinity control
    - Yes, -cc, -S, -ss, -d …
  - Use fewer cores per node
    - Yes, both for using more memory per process and for bandwidth.
  - Huge pages
    - None. But MPI uses huge pages under the hood
  - Core specialization
    - None. My tests with two applications see no improvement.
  - IO tuning
    - MPI-IO, adjust striping counts, …

S3D: Rank Reordering

30x30x30 cube with c2h2 combustion chemistry
50 time steps, no IO

Ideal is flat; Lower is better

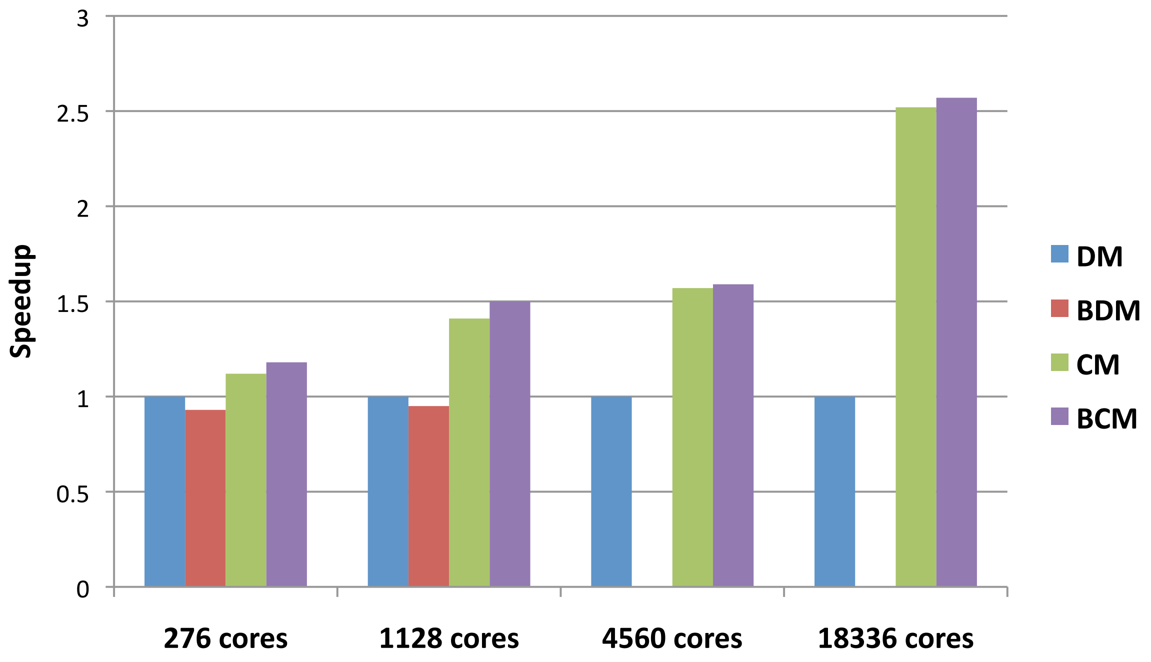


3,072 cores:
No reordering: 176.2s
With reordering: 165.7s

Courtesy of Hemanth Kolla and Evatt Hawkes

- **S3D: numerical simulations of turbulent combustion**
- **Very little global communication**
- **Almost all communication is among nearest neighbors in physical space**
- **Reorder MPI ranks to place ranks that are contiguous in physical space.**

MFDn: Rank Reordering



DM

1				
6	2			
10	7	3		
13	11	8	4	
15	14	12	9	5

CM

1				
2	6			
3	7	10		
4	8	11	13	
5	9	12	14	15

BDM

1			14	10
6	2			15
11	7	3		
	12	8	4	
		13	9	5

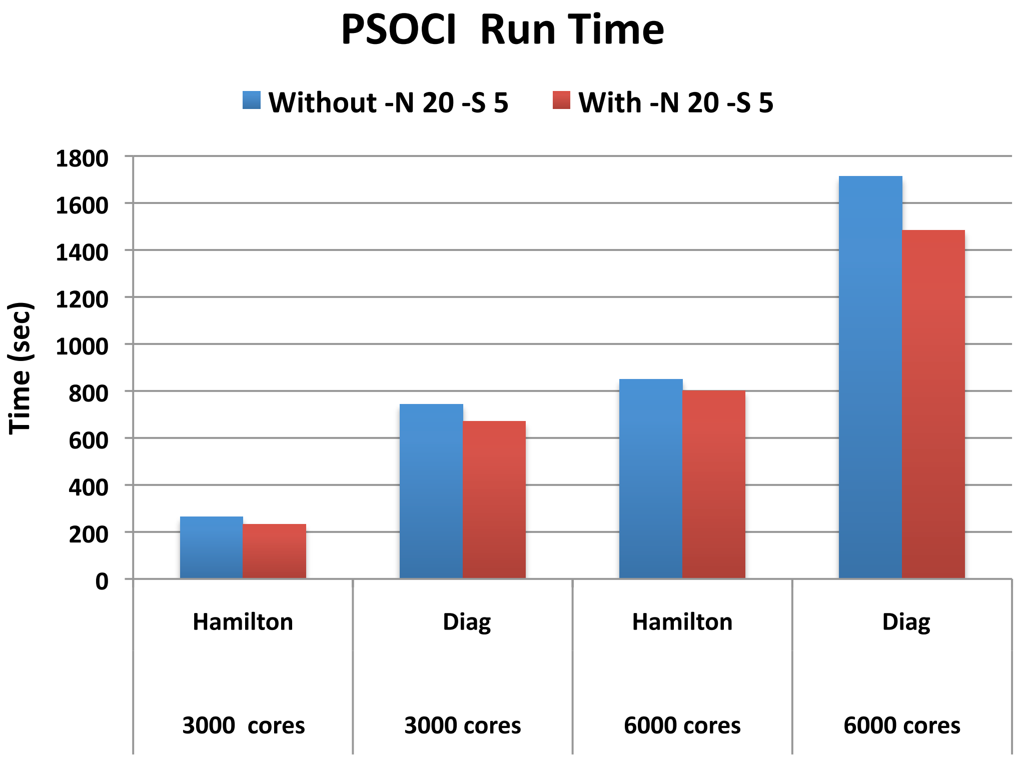
BCM

1			12	14
2	4			15
3	5	7		
	6	8	10	
		9	11	13

- **MFDn: a nuclear physics code**
- **Ranks reordered to avoid hot spot and congested links.**
- **The grid represents the 2D decomposition of the Hamiltonian H over processors for parallel processing.**
- **BCM order is the best.**

Courtesy of H.Metin Aktulga et al.

PSOCI: Use Fewer Cores Per Node



- **PSOCI: a chemistry code**
- **Global Arrays with ga++ bindings is used**
- **4 cores per node left free for local I/O, etc.**

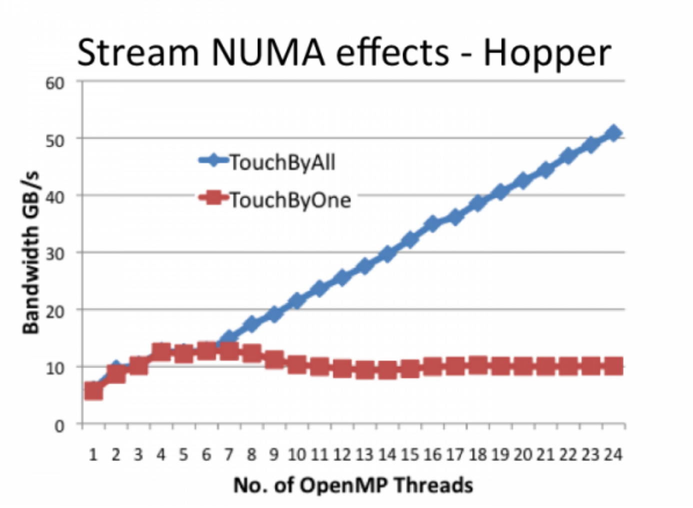
Courtesy of Jeffrey Tilson
3,000 core and 6,000 core runs used different problem sizes.

“First Touch” Memory

- Memory affinity is not decided by the memory allocation, but by the initialization. This is called “**first touch**” policy.
- Hard to do “perfect touch” for real applications. NERSC recommends do not use more than 6 threads per node to avoid NUMA effect.
- No real user applications reported used “first touch”.

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

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

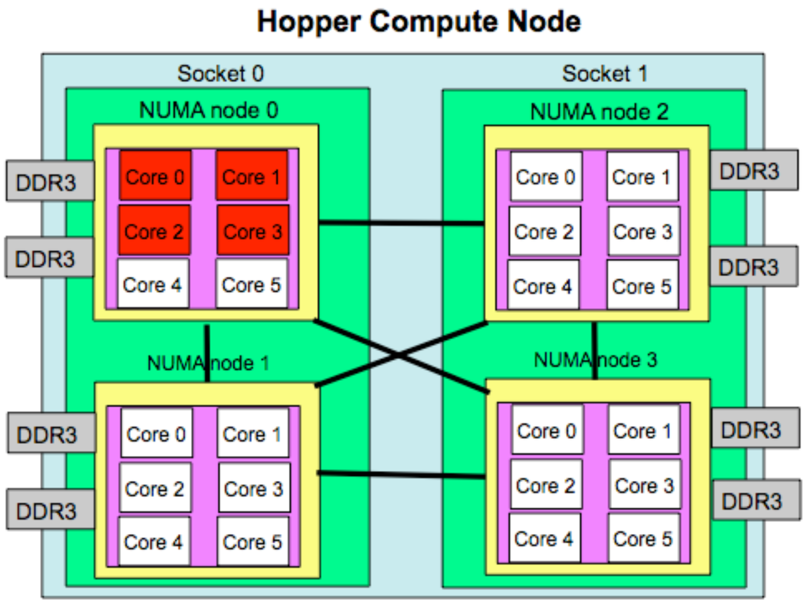


Courtesy of Hongzhang Shan

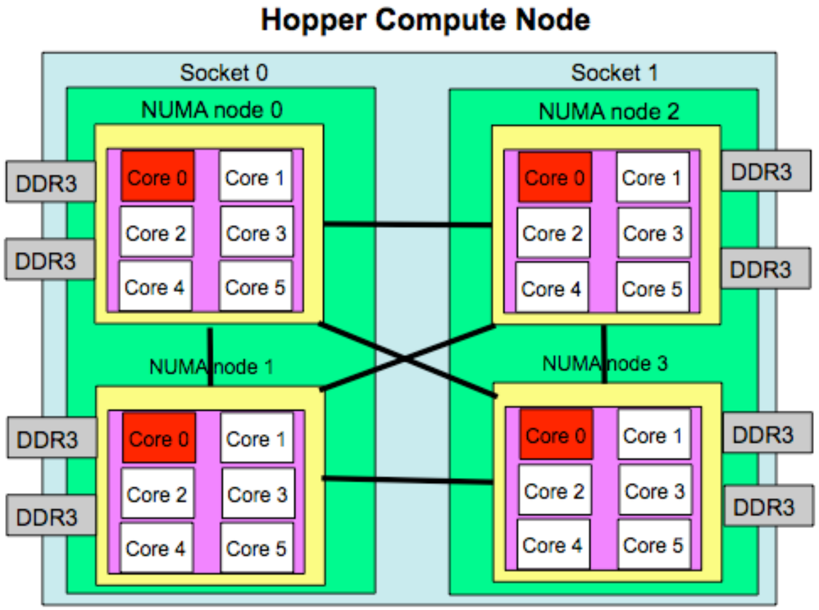
aprun “-S” option

- The “-S” option is especially important for hybrid MPI/ OpenMP applications, since it helps to spread the MPI tasks onto different NUMA nodes.

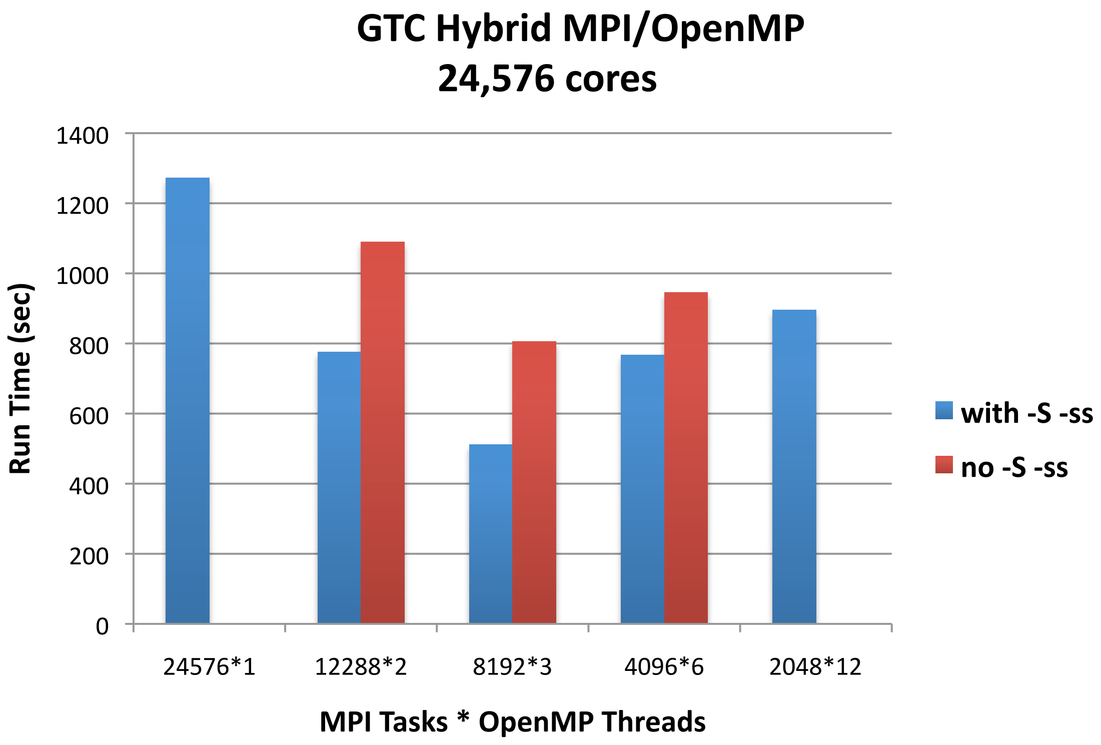
aprun –n 4 –d 6…



aprun –n 4 –S 1 –d 6 …

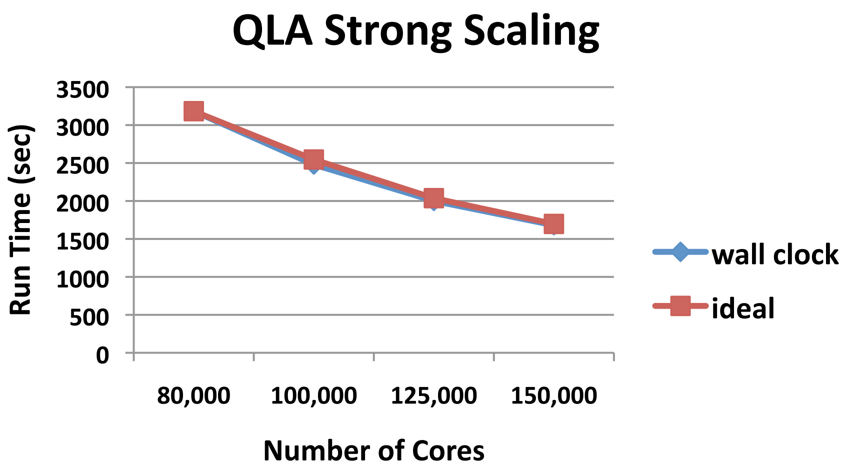


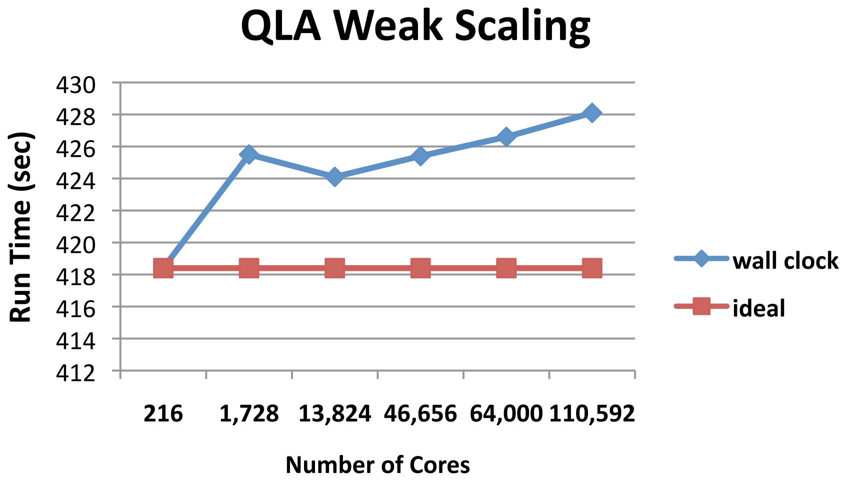
GTC: Hybrid MPI/OpenMP



- **GTC: 3D Gyrokinetic Toroidal Fusion Code.**
- **Sweet spot of 3 threads per MPI task.**
- **Large penalty without specifying "–S .. –ss" aprun options**
- **NUMA effects seen with 12 threads**

QLA: Code Tuning

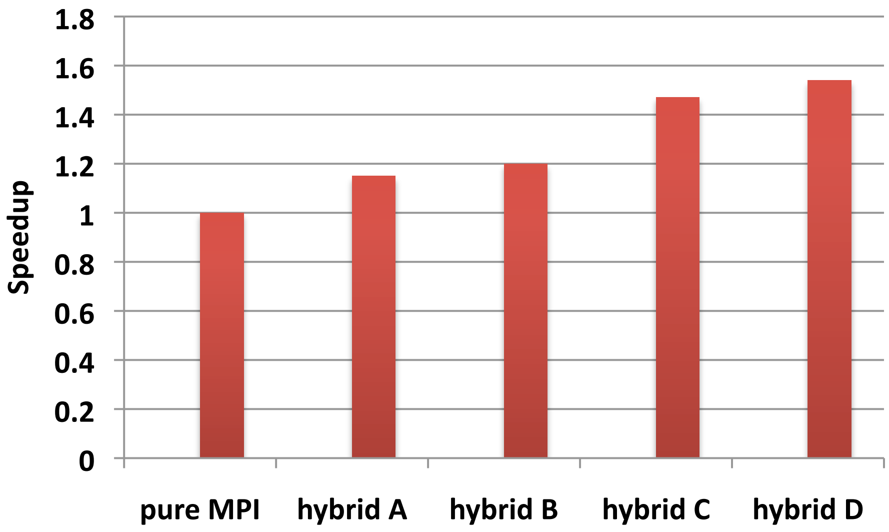




Courtesy of Min Soe

- QLA: a quantum turbulence code.
- Super-linear scaling to 150,000 cores!
- 3 major steps: unitary collide, stream, rotate.
- Combining first 2 steps resulted in 1.6x speedup.
- Hand tuning by simplifying expressions to eliminate redundant operations not recognized by the compilers: additional 1.4x speedup.
- Use non-blocking send and recv for comm and comp overlap.

MFDn: Overlap Comm and Comp in Hybrid MPI/OpenMP

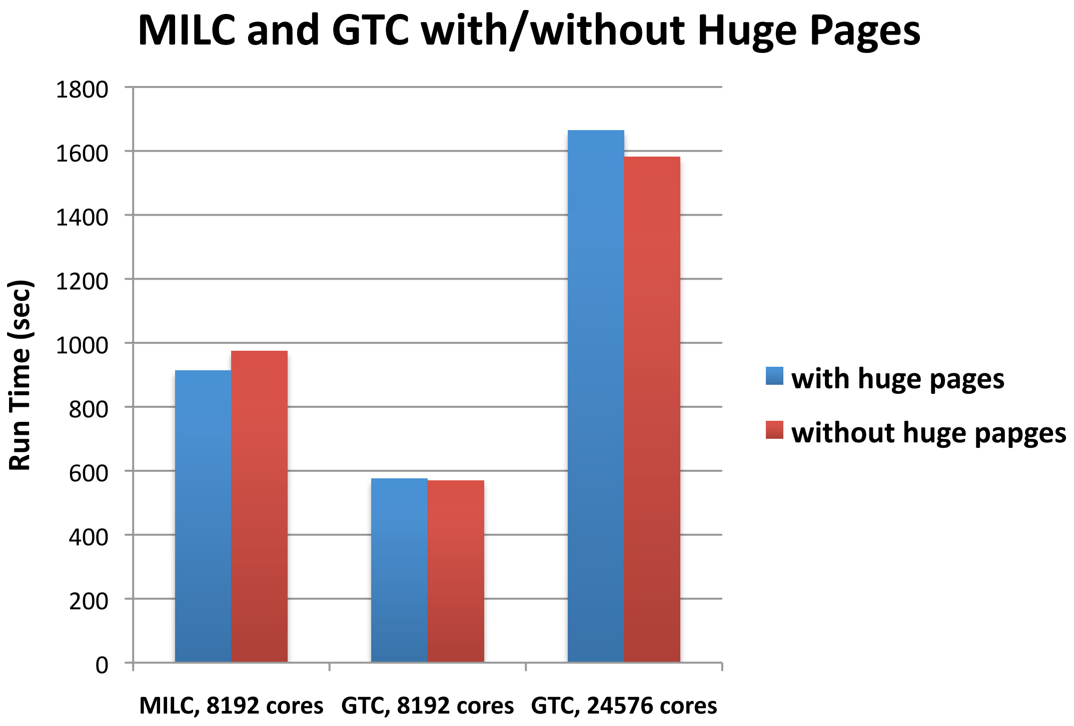


Pure MPI: 12,096 MPI tasks
Hybrid: 2,016 MPI tasks,
6 threads per task.

Courtesy of H. Metin Aktulga et al.

- **MFDn: a nuclear physics code. BCM rank order.**
- **Hybrid A: hybrid MPI/OpenMP**
- **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.**

MILC/GTC: Huge Pages



GTC 8,192 core and 24,576 core runs used different problem sizes.

- **MILC: Lattice Gauge Physics code.**
- **GTC: Fusion code.**
- **Huge pages can improve memory performance for common access patterns on large data sets.**
- **Huge pages effect is within production environment variations from these tests.**

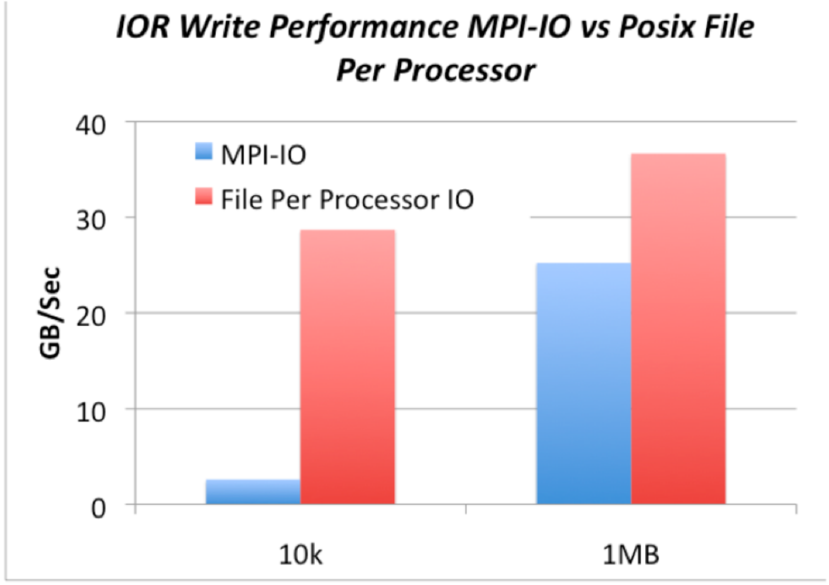
Huge Pages Issue

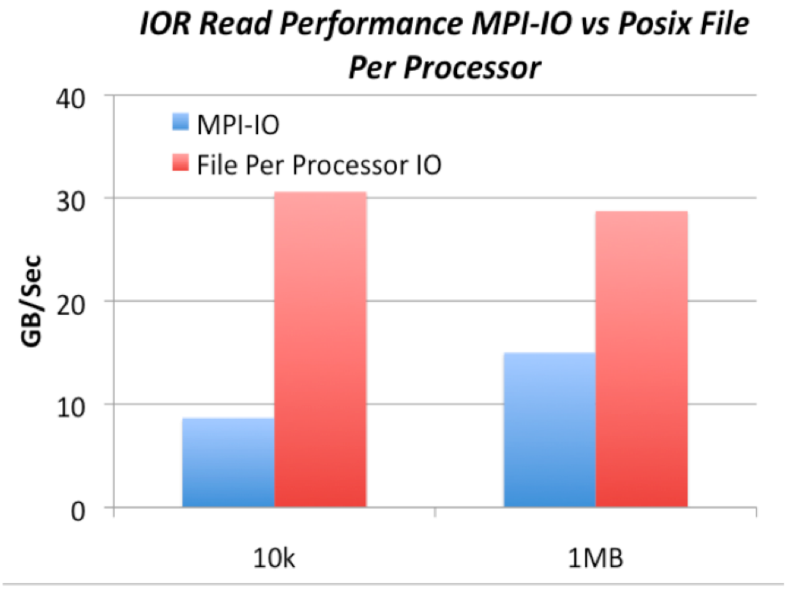
- Jobs explicitly use huge pages or large jobs which implicitly use huge pages by MPI are sometimes affected by not enough huge page memory error on the compute nodes.
- From June to Oct 2011, two NERSC benchmark applications that use huge pages only had ~35% success rate.
- Cray is actively pursing the bug about the compute node memory being gradually fragmented after a system reboot. Meanwhile, we modified the node health check script to identify these low huge memory nodes, and admin down and warmboot them manually. A bug in PMI library was also fixed.
- Jobs using huge pages are having much higher success rate.

Hung Jobs Issue

- Shortly after CLE4.0UP02 upgrade in Jan 2012, we received hung jobs report. Many users (50+) were affected, and huge amount of wasted compute hours (13.5M core hours) had to be refunded to the users.
- Cray and NERSC teams worked intensively and held daily progress meetings for about a month.
- 8 “bad” nodes nodes in a state that datagram packets cannot be received were identified. Rebooting these nodes helped the situation tremendously.
- Worked with users to test various MPI libraries and env settings before Cray provided two kGNI patches for the bug that attributed to the “bad” nodes.
- No more hung jobs …

Tuning Block Size for MPI-IO

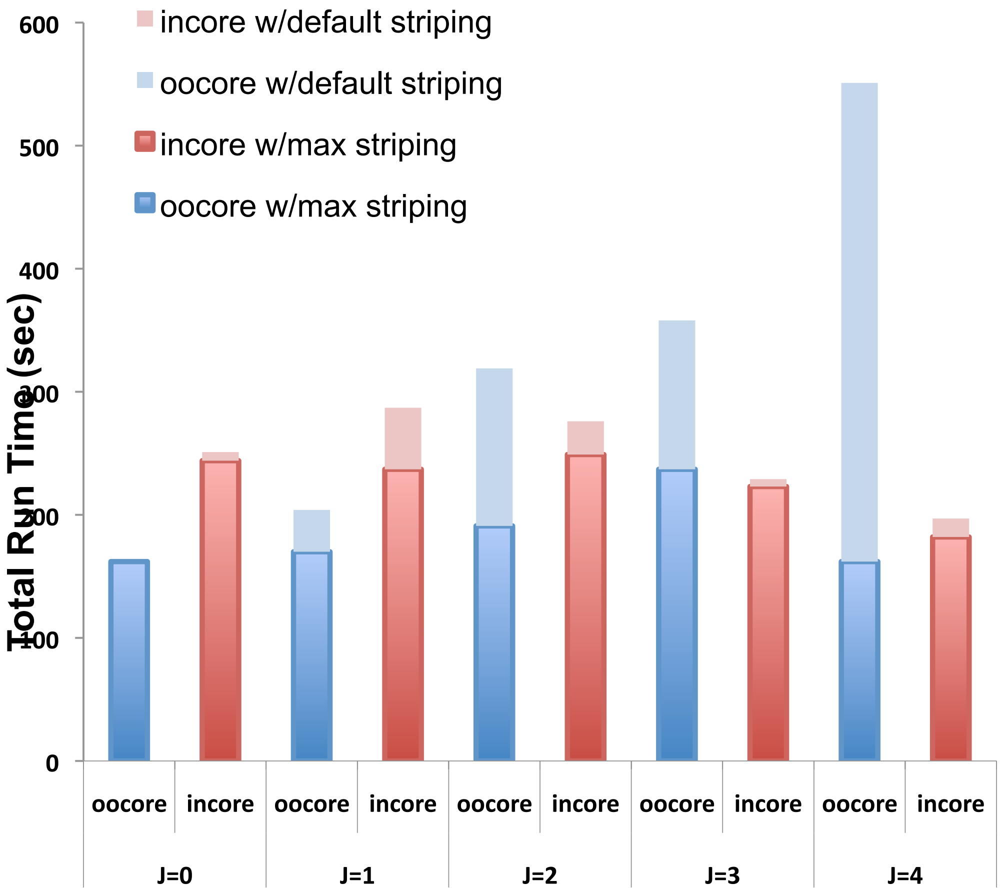

IOR Write Performance MPI-IO vs Posix File Per Processor


IOR Read Performance MPI-IO vs Posix File Per Processor

Courtesy of Yushu Yao

- **Block size has a large effect on MPI-IO performance, especially the write rate.**
- **Similar performance on DVS enabled GPFS file system, as on native Lustre file system.**
- **NERSC is working with Cray to address the problem.**

MFDn J-scheme: Lustre File Striping



- **MFDn Problem size:**
  ^{6}Li, Nmax=12, J=0 to 4
- **Total size of all data-blocks: 2.7~10 GB**
- **Number of blocks: 2.5x10^{5}**
- **Total number of block accesses: 1.1~1.6x10^{8}**
- **Default striping on Hopper = 2**
- **Max. striping on Hopper = 156**

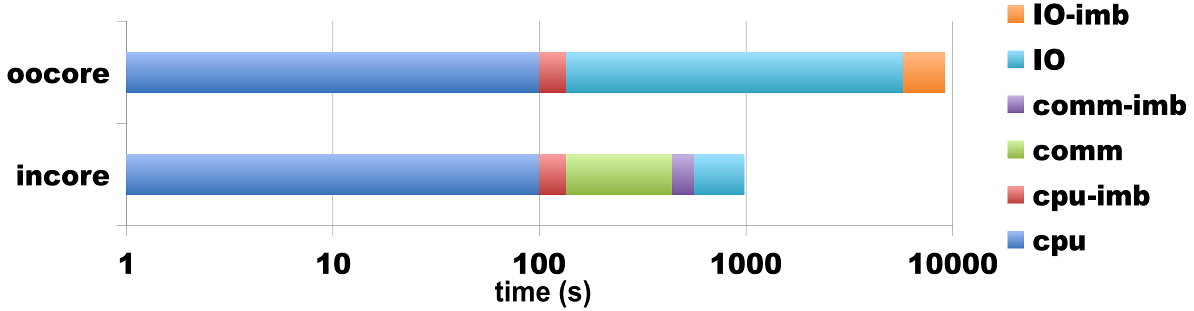
  % lfs setstripe -c -1

Courtesy of H. Metin Aktulga et al.

Much Larger Problem → Bigger Gain with Lustre File Striping

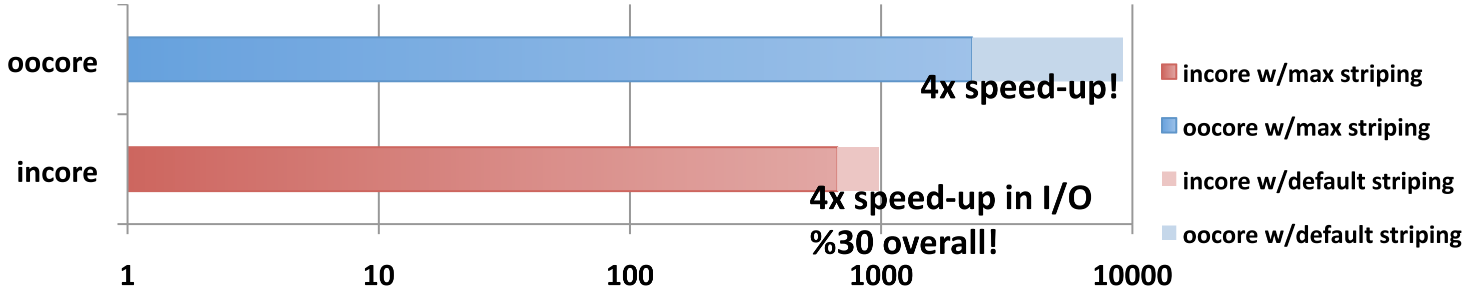
^{6}Li, N$_{max}$=14, J=3
Each block size: 67.1 GB,
Number of blocks: 7.4x10^5
Number of access: 7x10^8

Default Striping



Courtesy of H.Metin Aktulga et al.

Default and Max Striping



4x speed-up!

4x speed-up in I/O
%30 overall!

Summary

- Experiment with different compilers and compiler flags.
- MPI rank reordering is a simple and effective run time tuning method if you know your application's communication pattern well.
- Using fewer cores per node helps.
- Hybrid MPI/OpenMP is encouraged on Hopper since it also reduces the memory footprint. NERSC suggests not to use more than 6 threads on one node. Make sure to use the process (-S) and memory affinity options (-ss) for aprun.
- Consider overlapping comm and comp in hybrid MPI/OpenMP.
- Using huge pages is worth trying.
- Tune block sizes for MPI-IO, use different stripe sizes for Lustre files.

Further References

- NERSC Hopper web pages: https://www.nersc.gov/users/computational-systems/hopper
- Hopper Performance and Optimizations web page:

  http://www.nersc.gov/users/computational-systems/hopper/performance-and-optimization/
- Hopper Programming Tuning Options web page: http://www.nersc.gov/users/computational-systems/hopper/programming/tuning-options/
- Hopper Run Time Tuning Options web page:

  http://www.nersc.gov/users/computational-systems/hopper/running-jobs/runtime-tuning-options/

Acknowledgement

- Many NERSC users participated in Running Large Jobs on Hopper user survey and provided feedback on various aspects.
- Special thanks to Hemanth Kolla, Evatt Hawkes, Jeffrey Tilson, Min Soe , Hasan Metin Aktulga, and Pieter Maris for providing performance tuning stories and test cases.
- Thanks to Hongzhang Shan for the STREAM NUMA effect plot and Yushu Yao for the MPI-IO performance plot.
- Authors are supported by the Director, Office of Science, Advanced Scientific Computing Research, U.S. Department of Energy.