Hopper Phase-2 Migration

Harvey Wasserman
User Services Group

Nick Wright
Advanced Technologies Group

NUG Training

October 2010
The NERSC Hopper System

- Cray XT6, 6,392 nodes, 153,408 cores, 2.1-GHz AMD Magny-Cours Opteron processor
- Cray Gemini Interconnect
- 1.25 Petaflops peak performance
- 2-PB disk Lustre filesystem
Part 1

PROGRAMMING
What is Different About Hopper?

• The new Hopper Phase-2 system will have 24 cores per node.

• Franklin has only four.

• The way that you use the new Hopper system may have to change as a result.
What Else is Different?

• Less memory per core: 1.33 GB vs. 2.0 GB
  – 8 GB per node (Franklin);
  – 32 GB per node (Hopper, 6,008 nodes)

• “OOM killer terminated this process” error
  OOM = Out of Memory

• (Hopper will have 384 larger-memory nodes 64 GB.)
Why Less Memory Per Core?

- Technology trends:
  - Memory density 2X every 3 yrs; processor logic every 2 yrs
  - Storage costs ($/MB) drops more gradually than logic costs

- NERSC optimized the Hopper system for a diverse workload
  - fixed budget; memory cost is already a significant portion.

*Source: David Turek, IBM*
What Else is Different?

- “Deeper” Memory Hierarchy in Hopper

Memory Hierarchy Levels

1000s of Bytes
~2 ns per access

MBytes
~10s ns per access

Gbytes
~100ns ns per access

PBytes
~ms per access

Registers

Caches

Memory

Memory

Memory

Disk

Faster

Larger
What Else is Different?

• “Deeper” Memory Hierarchy
  – NUMA: Non-Uniform Memory Architecture
  – All memory is transparently accessible but...
  – Longer memory access time to “remote” memory
  – A process running on NUMA node 0 accessing NUMA node 1 memory can adversely affect performance.

2xDDR1333 channel
21.3 GB/s

3.2GHz x16 lane HT
12.8 GB/s bidirectional

3.2GHz x8 lane HT
6.4 GB/s bidirectional
Franklin: Uniform Memory Architecture

Hopper: Non-Uniform Memory Architecture
What About the Future?

• The technology trends point to
  – Little or no gain in clock speed or performance per core;
  – Rapidly increasing numbers of cores per node;
  – Decreased memory capacity per core (possible slight increase per node)
  – Decreased memory bandwidth per core
  – Decreased interconnect bandwidth per core
  – Deeper memory hierarchy

• Hopper is the first example at NERSC but surely not the last
Will My Existing Code Run?

• Probably, yes, your MPI code will run.

• But the decrease in memory available per core may cause problems ...
  — May not be able to run the same problems.
  — May be difficult to continue “weak” scaling (problem size grows in proportion to machine size).

• (and your MPI code might not use the machine most effectively.)

• Time to consider alternative programming models?
What is NERSC Doing About All This?

- NERSC-Cray “Programming Models Center of Excellence”
- Close ties to UCB and LBNL Computing Research Division
- Investigation of Advanced Programming Models
- Study of application software that NERSC provides
  - OpenMP ready?
  - OpenMP capable?
What Does NERSC Recommend?

• NERSC recognizes the huge investment in MPI.

• But given the technology trends...

• We suggest a move towards programming models other than pure MPI

• A good place to start: MPI + OpenMP (“Hybrid”)
  - MPI for domain decomposition and OpenMP threads within a domain
  - Suggested primarily to help with memory capacity
Isn’t This the Same as Clusters of SMPs (ca 2002)?

- SMP: Symmetric Multiprocessor
  - aka clusters, Networks of Workstations, CLUMPS, ...
  - SGI Origin, ASCI Q/Blue Mountain, Berkeley NOW, IBM SP, ...

- In some ways the issues are the same:
  - Memory architecture is the key

- But chip multiprocessors have vastly improved inter-core latencies and bandwidth.

- With today’s trends we have no choice.
What are the Basic Differences Between MPI and OpenMP?

**Message Passing Model**
- Program is a collection of processes.
  - Usually fixed at startup time
  - Single thread of control plus private address space -- NO shared data.
- Processes communicate by explicit send/receive pairs
  - Coordination is implicit in every communication event.
- MPI is most important example.

**Shared Address Space Model**
- Program is a collection of threads.
  - Can be created dynamically.
- Threads have private variables and shared variables
  - Threads communicate implicitly by writing and reading shared variables.
  - Threads coordinate by synchronizing on shared variables
- OpenMP is an example

K. Yelick, CS267 UCB
Why are MPI-only Applications Memory Inefficient?

• MPI codes consist of \( n \) copies of the program

• MPI codes require application-level memory for messages
  - Often called “ghost” cells

• MPI codes require system-level memory for messages
  - Assuming the very common synchronous/blocking style
Why Does Hybrid/OpenMP Help?

- Reduced Memory Usage:
  - Fewer instances of your program on the node
  - Eliminate some ghost cell memory

Figures from Kaushik Datta, Ph.D. Dissertation, UC Berkeley, 2009
Why Does Hybrid/OpenMP Help?

- Send larger MPI messages
  - small messages are expensive
- No intra-node messages

\[
\text{Time} \\
\text{Slope} = \text{cost per word} = 1/BW \\
\{ t_s = \text{startup cost} \}
\]
Why Does Hybrid/OpenMP Help?

• There may be scalability limits to domain decomposition

• OpenMP adds fine granularity (larger message sizes) and allows flexibility of dynamic load balancing.

• Some problems have two levels of parallelism
What are the Benefits of OpenMP?

- Uses less memory per node
- At least equal performance
- Additional parallelization may fit algorithm well
  - especially for applications with limited domain parallelism
- Possible improved MPI performance and load balancing
  - Avoid MPI within node
- OpenMP is a standard so code is portable
- Some OpenMP code can be added incrementally
  - Can focus on performance-critical portions of code
- Better mapping to multicore architecture
What are the Disadvantages of OpenMP?

• Additional programming complexity
• Can be difficult to debug race conditions
• Requires explicit synchronization
• Additional scalability bottlenecks:
  – thread creation overhead, critical sections, serial sections for MPI
• Cache coherence problems (false sharing) and data placement issues
  – Memory locality is key...
  – but OpenMP offers no direct control
Are There Additional Solutions?

- Sometimes it may be better to leave cores idle
  - Improves memory capacity and bandwidth
  - Improves network bandwidth

- However, you are charged for all cores
Typical OpenMP Program

- Execution begins with a single “Master Thread”
- Threads “fork” at each parallel region, join at end
Can I Debug OpenMP and Hybrid Codes?

- Difficult because of *race conditions* – imprecise and non-reproducible ordering of memory read/store operations
- Common bugs:
  - incomplete or misplaced synchronization
  - improper scoping
  - occur often as a result of converting serial code
Screenshot of a Totalview debugging session with a hybrid MPI / OpenMP code.
Can I Analyze OpenMP Performance?

Yes: Use CrayPat Tool

module load xt-craypat
cd $SCRATCH/...
make (e.g., ftn -o my.exe mycode.f)
pat_build -g omp
qsub ...
aprun -n #_cores my.exe+pat
pat_report datafile.xf > out
What are the Disadvantages of OpenMP?

- Additional programming complexity
- Can be difficult to debug race conditions
- Requires explicit synchronization
- Additional scalability bottlenecks:
  - thread creation overhead, critical sections, serial sections for MPI
- Cache coherence problems (false sharing) and data placement issues
  - Memory locality is key...
  - but OpenMP offers no direct control
What’s All This About Locality?

- Remember: All memory accesses on the node happen transparently
  - but remote access takes longer

- Need NUMA control - memory and process affinity
  - Improve performance
  - Eliminate performance variability
  - Avoid resource contention

Where do processes, threads, and their memory go on the Hopper node?
Memory Affinity via “First Touch”

- Memory is mapped to the NUMA node containing the core that first touches that memory.
- “Touch” means write (not allocate)
- Solution (Golden Rule): have each thread initialize the points that it will later be processing
  - Initialize memory immediately after allocating it
  - Initialize memory in parallel regions, not in serial code

Recommended: Tutorial M16 at SC10
$S_i = AX_i + Y_i$  Performance

Measurements by Hongzhang Shan (CRD)
<table>
<thead>
<tr>
<th>Time (PDT)</th>
<th>Topic</th>
<th>Presenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00-8:30am</td>
<td>Arrive at NERSC</td>
<td></td>
</tr>
<tr>
<td>8:30am-noon</td>
<td>• Single processor optimizations for Magny Cours.</td>
<td>Cray</td>
</tr>
<tr>
<td></td>
<td>• Tips and tricks for running on the Hopper XE6 system</td>
<td></td>
</tr>
<tr>
<td>noon-1pm</td>
<td>Lunch + hands on sessions. Ask questions to Cray and NERSC staff</td>
<td></td>
</tr>
<tr>
<td>1-2pm</td>
<td>Best Practices for Hybrid OpenMP/MPI Programming on Hopper</td>
<td>Nick Wright</td>
</tr>
</tbody>
</table>
Part 2

RUNNING ON HOPPER
Running on Hopper

- Submit a job to the batch system requesting resources
  - Interactive
  - Batch
  - #PBS -l mppwidth = Total_Number_of_cores_needed

- Launch executable with aprun
  - Need to ensure that aprun command is consistent with batch resources requested
Running on Hopper

• You must recompile

  – Franklin and Hopper Phase 1 binaries include SeaStar
  – Hopper Phase 2 binaries need Gemini

  => you must recompile
**aprun: Example 1**

- Pure MPI application, using all cores in a node: 32 MPI tasks on 32 cores

```bash
#PBS -l mppwidth=32
aprun -n 32 a.out
```

**Franklin**

8 nodes, fully-populated, (32 cores charged against allocation)

**Hopper**

2 nodes, not fully-populated

NOTE: you are charged for all the cores allocated
(48 cores charged against allocation)
(8 cores on one node, 24 on other node is default)

On Hopper, you can request actual number needed; batch system will allocate required number of nodes. **NOT RECOMMENDED!!**
Request full nodes (#PBS -l mppwidth=48)
Important Note About Defaults

• Non-local Hopper NUMA node memory is **not** available unless your combination of #PBS directives and aprun command request it.

• Example: If you use 1/2 the cores in the node, and all are on two NUMA nodes only 1/2 the Hopper node memory is available.

• If you don’t fully populate the node be sure to spread your cores over all NUMA nodes
• Underpopulate nodes by 1/2 to save memory, 48 MPI tasks

Requirements:
- Requires 48 tasks ÷ 2 tasks per node X 4 cores per node = 96 cores (24 nodes * 4 cores per charged against allocation)

```
#PBS -l mppwidth=96
aprun -n 48 -N 2 a.out
```

- Requires 48 tasks ÷ 12 tasks per node X 24 cores per node = 96 cores (4 nodes * 24 cores per charged against allocation)

```
#PBS -l mppwidth=96
aprun -n 48 -N 12 -S 3 a.out
```
aprun NUMA options

• Important to ensure that MPI tasks are assigned separate NUMA nodes when underpopulating the node

- Cores per NUMA node; 1-6, default 6;
  aprun –S cores

- NUMA nodes per Hopper node; 1-4, no default:
  aprun –sn nodes

- NUMA node list; 0,1,2,3 comma or hyphen delimited:
  aprun –sl node-list
• Underpopulate nodes by 1/2 to save memory, 48 MPI tasks

Requires 48 tasks ÷ 12 tasks per node X 24 cores per node
= 96 cores (4 nodes * 24 cores per charged against allocation)

Hopper

#PBS –l mppwidth=96
aprun –n 48 –N 12 –S 3 a.out  

#PBS –l mppwidth=96
aprun –n 48 –N 12 –S 4 a.out  

optimal

avoid
aprun NUMA options

CPU affinity: Bind processes / threads
- to each core within a NUMA node, or
- to any core within a NUMA nodes or
- don’t bind at all;
-cc is the default for MPI codes

`aprun -cc [ cpu | numa_node | none ]`

Allocate memory only local to the NUMA node; do not use if underpopulating

`aprun -ss`
• Use **both** the `OMP_NUM_THREADS` environment variable + `aprun --n --d` options

• `aprun --n` # option specifies # of MPI processes

• `aprun --d` # option specifies number of threads per MPI task.
  – each of the “-n” MPI processes creates “-d” threads
aprun: Example 3

- Hybrid OpenMP / MPI

Franklin

92 MPI tasks, 4 OpenMP threads each:
Total cores = 92 tasks / 1 MPI task per node X 4 cores per node = 368 (92 nodes)

```
#PBS mppwidth=368
export OMP_NUM_THREADS=4
aprun -n 92 -N 1 -d 4 a.out
```

Hopper

92 MPI tasks, 6 OpenMP threads each:
Total cores = 92 tasks / 4 MPI tasks per node X 24 cores per node = 552 (23 nodes)

```
#PBS mppwidth=552
setenv OMP_NUM_THREADS 6
aprun -n 92 -N 4 -S 1 -d 6 a.out
```
Some Error Messages

• Claim exceeds reservation's node-count
  — On Franklin usually caused by requesting fewer cores (#PBS –l mppwidth=#) than aprun needs
  — On Hopper may result from improperly spreading processes and threads over NUMA nodes

• Claim exceeds reservation's memory
  — On Hopper; happens because having a compute node reserved for your job does not guarantee that you can use all NUMA nodes.
“Prediction is difficult - especially for the future.”
- Y. Berra

“The future will be just like the present - only more so.”
- Groucho Marx

Part 3
PERFORMANCE OF HOPPER
### What Performance Should I Expect on Hopper Phase-2?

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cores</th>
<th>Frequency (GHz)</th>
<th>Peak (GFLOPS) per Core</th>
<th>Peak (GFLOPS) per Processor</th>
<th>Bandwidth (GB/s)</th>
<th>Balance (Bytes/Flop)</th>
<th>Hyper-Transport Technology</th>
<th>Memory Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barcelona (Cray XT4)</td>
<td>4</td>
<td>2.3</td>
<td>9.2</td>
<td>36.8</td>
<td>12.8</td>
<td>0.34</td>
<td>3x 2GT/s</td>
<td>2x DDR2 667 per 4 cores</td>
</tr>
<tr>
<td>Magny Cours (Cray XT6)</td>
<td>12</td>
<td>2.1</td>
<td>8.4</td>
<td>100.8</td>
<td>42.6</td>
<td>0.42</td>
<td>4x 6.4 GT/s</td>
<td>2x DDR3 1333 per 6 cores</td>
</tr>
</tbody>
</table>
What Performance Should I Expect on Hopper Phase-2?

PERFORMANCE DATA ARE PROPRIETARY – NOT TO BE PUBLISHED IN ANY FORM

Cray XT6 PERFORMANCE DATA ARE FROM AN EARLY VERSION OF THE SYSTEM
What Performance Should I Expect on Hopper Phase-2?

PRELIMINARY PERFORMANCE DATA: PROPRIETARY – NOT TO BE PUBLISHED IN ANY FORM
## NERSC Application Benchmarks

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM</td>
<td>F77</td>
<td>Community Atmosphere Model, “D” grid</td>
</tr>
<tr>
<td>GAMESS</td>
<td>F77</td>
<td>Quantum Chemistry RHF gradient MP2</td>
</tr>
<tr>
<td>GTC</td>
<td>F90</td>
<td>Particle in Cell – Fusion turbulence</td>
</tr>
<tr>
<td>IMPACT-T</td>
<td>F90 + FFTW</td>
<td>Particle in Cell – Accelerator design</td>
</tr>
<tr>
<td>MAESTRO</td>
<td>F90 (C)</td>
<td>Low Mach number flow astrophysical</td>
</tr>
<tr>
<td>MILC</td>
<td>C</td>
<td>Lattice QCD</td>
</tr>
<tr>
<td>PARATEC</td>
<td>Fortran + FFTW + BLAS</td>
<td>Plane Wave Density Functional Theory</td>
</tr>
<tr>
<td>PMEMD</td>
<td>F90</td>
<td>Particle Mesh Ewald Molecular Dynamics</td>
</tr>
</tbody>
</table>
Application Benchmark Times
(run times in seconds)

(This slide intentionally left blank in published version of the slides)
Summary

• Hopper is performing well.

• Even for codes performing well you would be well advised to consider an alternative to MPI-only programming.

• The key to success is likely to be careful consideration of locality.

• NERSC can help.
FILESYSTEMS

Home directories are global, meaning
common across all NERSC systems.
Accessible from login and compute nodes.
Quota is 40 GB. Refer to your home space in
scripts as $HOME.

Scratch directories are configured for parallel
I/O, accessible from login and compute nodes,
in /scratch1/scratchdirs/username
and /scratch2/scratchdirs/username. Refer to
these in scripts as $$SCRATCH1
and $$SCRATCH2. Quota is 2 TB but purging will
occur. Use the NERSC web form if more
space is temporarily needed.

Project directories are available via the
NERSC Global Filesystem (NGF); use these to
store data across NERSC platforms or amongst
users in a project. Use a NERSC web
form to request (under Global File System).

HIGH PERFORMANCE
STORAGE SYSTEM (HPSS)
Use HPSS to back up all your code and data.
Access is via ftp, pftp, htar, or hsi to
archive.nersc.gov. Use NIM password or
generate a token (stored in .netrc) then no
password required. The hsi utility uses syntax
similar to Unix for most commands; additional
commands include add, cdls, cput, dump,
get, mkdir, mdelete, mget, mput, put,
replace, rename, save, send, store.

Several ways of using hsi:
- From a command line: Just type hsi, wait for
HPSS prompt, type commands, & exit to
- Multiple commands at once: hsi "mkdir
foo; cd foo; put data file"
- From input file: hsi "in input file"
- From or to standard input or output:
tar cvf - | hsi put - | d.tar
hsi get - | d.tar | tar xvf -

NERSC CONSULTING

Left to right, top: Katie Antypas (User Services
Group Leader), Richard Gerber, Helen He, Woo-
Sun Yang, Harvey Wasserman; bottom: Zhengji
Zhao, Viraj Paropkari, Mike Stewart, David
Turner, Eric Hjort

consult@nersc.gov
510-486-8611 or
800-66-NERSC, menu option 3
http://www.nersc.gov

NERSC ACCOUNT MANAGEMENT
(Passwords, New Users)

Clayton Bagwell, Mark Heer
510-486-8612 or
800-66-NERSC, menu option 2
800-66-NERSC, menu option 2
Online Account Management for
all users: http://nim.nersc.gov

NERSC

QUICK REFERENCE CARD

NERSC Hopper

Complete documentation available
on http://www.nersc.gov

Hopper OVERVIEW

The NERSC Hopper system, in full service early
2011, will have 1.25 Petaflops peak performance;
6,392 nodes, each with two 2.1-GHz Opteron 12-
core processors, 153,408 total cores, 24 cores
per node, 1.33 GB/core; Gemini interconnect in
3-D torus topology; operating system is the Cray
Linux Environment, with full Linux on the eight
login nodes and Compute Node Linux micro-
kernel on the compute nodes.

HOW TO LOG IN
Log in to hopper with:

ssh [-l user] hopper.nersc.gov

Use -l user only if user ID on your local system
is different from your user ID on Hopper. Note: use
hopp2.nersc.gov prior to January 2011.
Passwords must not be shared. Login privileges
are disabled with three login failures; call Account
Management to clear login failures.

Hopper NODE ARCHITECTURE
Note the Non-Uniform Memory Architecture
(NUMA) with six cores per NUMA node:
Low swirl burner combustion simulation. Image shows flame radical, OH (purple surface and cutaway) and volume rendering (gray) of vortical structures. Red indicates vigorous burning of lean hydrogen fuel; shows cellular burning characteristic of thermodiffusively unstable fuel. Simulated using an adaptive projection code. Image courtesy of John Bell, LBNL.

Hydrogen plasma density wake produced by an intense, right-to-left laser pulse. Volume rendering of current density and particles (colored by momentum orange - high, cyan - low) trapped in the plasma wake driven by laser pulse (marked by the white disk) radiation pressure. 3-D, 3,500 Franklin-core, 36-hour LOASIS experiment simulation using VORPAL by Cameron Geddes, LBNL. Visualization: Gunther Weber, NERSC Analytics.

Numerical study of density driven flow for CO₂ storage in saline aquifers. Snapshot of CO₂ concentration after convection starts. Density-driven velocity field dynamics induces convective fingers that enhance the rate by which CO₂ is converted into negatively buoyant aqueous phase, thereby improving the security of CO₂ storage. Image courtesy of George Pau, LBNL.

False-color image of the Andromeda Galaxy created by layering 400 individual images captured by the Palomar Transient Factory (PFT) camera in February 2009. NERSC systems analyzing the PTF data are capable of discovering cosmic transients in real time. Image courtesy of Peter Nugent, LBNL.

The exciton wave function (the white isosurface) at the interface of a ZnS/ZnO nanorod. Simulations performed on a Cray XT4 at NERSC, also shown. Image courtesy of Lin-Wang Wang, LBNL.

Simulation of a global cloud resolving model (GCRM). This image is a composite plot showing several variables: wind velocity (surface pseudocolor plot), pressure (b/w contour lines), and a cut-away view of the geodesic grid. Image courtesy of Professor David Randall, Colorado State University.