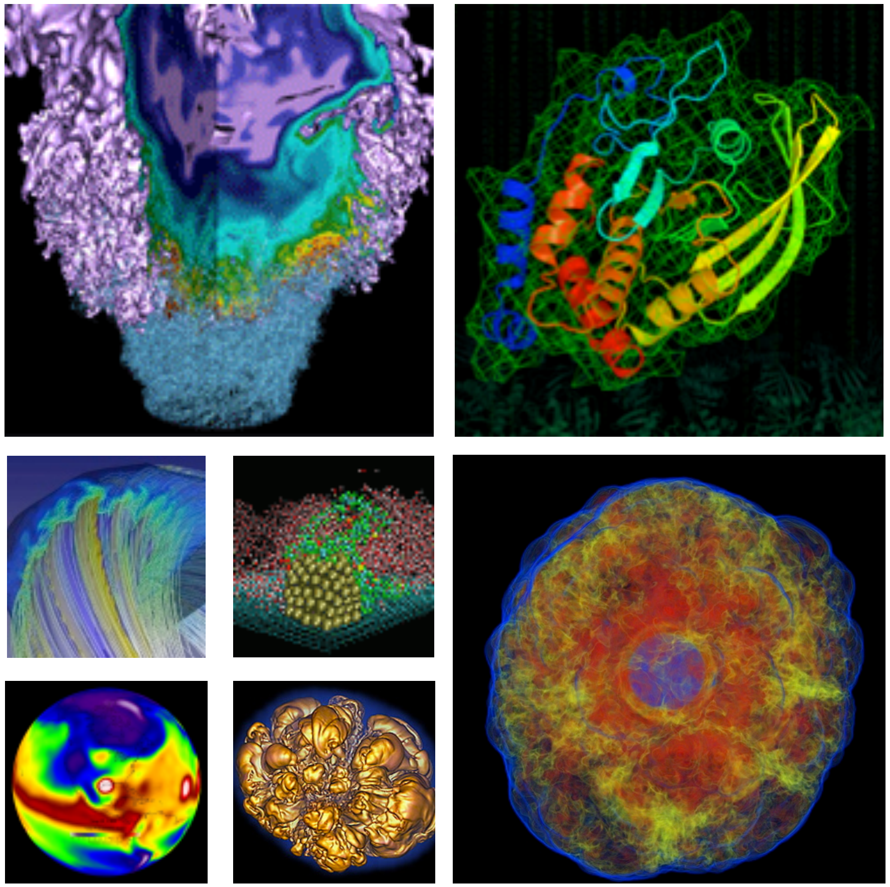
# Running Parallel Jobs



**Kirsten Fagnan**
NERSC User Services
February 15, 2013

## Outline

- Parallel Programming Models
- NERSC Systems and Sample Scripts
- Queues and Policies

## Parallel Computations

**Since you'll be learning about building jobs in the next talk, we'll focus on running applications in this one.**

## Parallel Programming Models

- **Message Passing**
  - In a message passing model, parallel tasks exchange data through passing messages to one another. These communications can be asynchronous or synchronous.
- **Shared Memory**
  - Parallel processes share memory and the threads will read and write to/from asynchronously.
- **Hybrid programming models**
  - MPI tasks are distributed across nodes with a number of threads being spawned per task.  This has benefits like lower memory requirements.

# What system is best for your job?

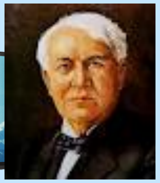
## Large-Scale Computing Systems

**Hopper (NERSC-6): Cray XE6**

- 6,384 compute nodes, 153,216 cores
- 144 Tflop/s on applications; 1.3 Pflop/s peak

**Edison (NERSC-7): Cray Cascade**

- To be operational in 2013
- Over 200 Tflop/s on applications, 2 Pflop/s peak



## Midrange

140 Tflops total

**Carver**

- IBM iDataplex cluster
- 9884 cores; 106TF

**PDSF (HEP/NP)**

- ~1K core cluster

**GenePool (JGI)**

- ~5K core cluster
- 2.1 PB Isilon File System

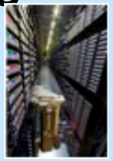
## NERSC Global Filesystem (NGF)

Uses IBM's GPFS

- 8.5 PB capacity
- 15GB/s of bandwidth

## HPSS Archival Storage

- 240 PB capacity
- 5 Tape libraries
- 200 TB disk cache



## Analytics & Testbeds

**Dirac** 48 Fermi GPU nodes

## Hopper - Cray XE6

- 153,408 cores, 6,392 nodes
- "Gemini" interconnect
- 2 12-core AMD 'MagnyCours' 2.1 GHz processors per node
- 24 processor cores per node
- 32 GB of memory per node (384 "fat" nodes with 64 GB)
- 216 TB of aggregate memory
- 1.2 GB memory / core (2.5 GB / core on "fat" nodes) for applications
- /scratch disk quota of 5 TB
- 2 PB of /scratch disk
- Choice of full Linux operating system or optimized Linux OS (Cray Linux)
- PGI, Cray, Pathscale, GNU compilers

## Hopper's Nodes

- **Compute Nodes**
  - The 6,384 compute nodes are dedicated to running scientific applications. A job is given exclusive access to each node it requests for the entirety of the job's run time. Since Hopper has 24 cores on each node, the minimum number of cores per job is 24.
- **Login Nodes**
  - Hopper's login nodes run a full Linux operating system and provide support services for the system. When you connect to Hopper with SSH, you land on the login nodes. These nodes are shared by many users; please do not run applications on the login nodes.
- **Job Host (MOM) Nodes**
  - MOM nodes are servers that execute batch job commands. These nodes are shared by many users and thus are not intended for compute- or memory-intensive applications.

## Sample Hopper Batch Script - MPI

```
#PBS -q debug
#PBS -l mppwidth=96
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -V

cd $PBS_O_WORKDIR
aprun -n 96 ./my_executable
```

MPI tasks

## Sample Hopper Batch Script - MPI

```
#PBS -q debug
#PBS -l mppwidth=192
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -V

cd $PBS_O_WORKDIR
aprun -n 96 –N 12 ./my_executable
```

MPI tasks

MPI tasks per node

## Hybrid OpenMP/MPI

```
#PBS –q regular
#PBS -l mppwidth=96
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -V
export OMP_NUM_THREADS=6
cd $PBS_O_WORKDIR

aprun -n 16 -d 6 -N 4 -S 1 -ss ./hybrid.x
```

threads per NUMA node

MPI tasks

MPI tasks per NUMA node

## Carver - IBM iDataPlex

- 3,200 compute cores
- 400 compute nodes
- 2 quad-core Intel Nehalem 2.67 GHz processors per node
- 8 processor cores per node
- 24 GB of memory per node (48 GB on 80 "fat" nodes)
- 2.5 GB / core for applications (5.5 GB / core on "fat" nodes)
- InfiniBand 4X QDR

- NERSC global /scratch directory quota of 20 TB
- Full Linux operating system
- PGI, GNU, Intel compilers

## Sample Carver Batch Script

```
#PBS -q debug
#PBS –l nodes=16:ppn=8
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -V
#PBS –l pvmem=20G

cd $PBS_O_WORKDIR
mpirun –n 128 ./myexecutable
```

↑ MPI tasks

## Sample Carver Batch Script

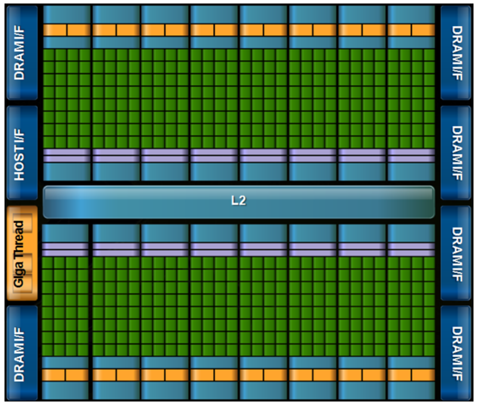
```
#PBS -q debug
#PBS –l nodes=1:ppn=512
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -V
#PBS –l pvmem=20G

export OMP_NUM_THREADS=512
cd $PBS_O_WORKDIR
./myexecutable
```

Number of threads

# Dirac – GPU Computing Testbed

- 50 GPUs
- 50 compute nodes
  - 2 quad-core Intel Nehalem 2.67 GHz processors
  - 24 GB DRAM memory
- 44 nodes: 1 NVIDIA Tesla C2050 (Fermi) GPU with 3GB of memory and 448 cores
- 1 node: 4 NVIDIA Tesla C2050 (Fermi) GPU's, each with 3GB of memory and 448 processor cores.
- InfiniBand 4X QDR



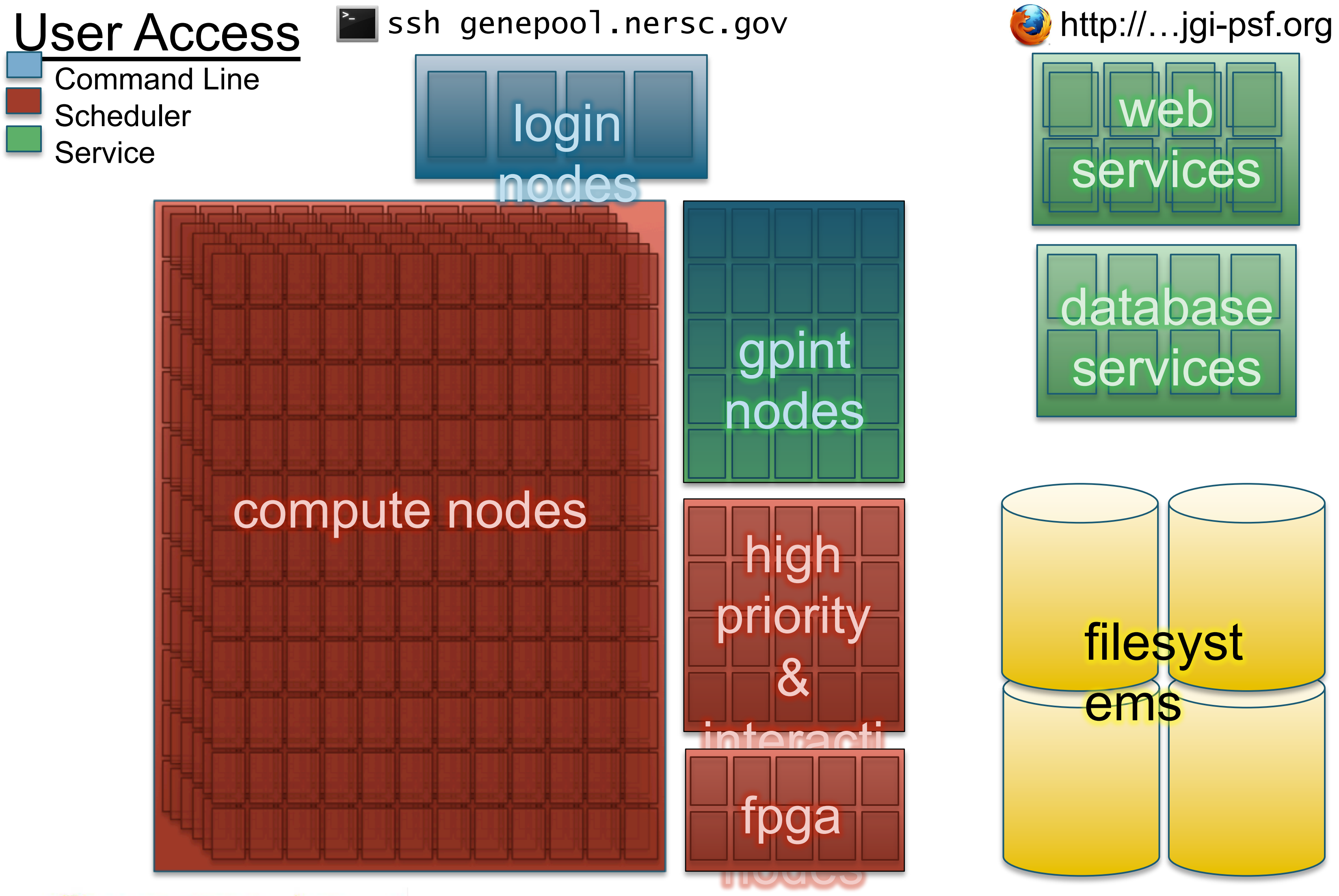
- CUDA 5.0, OpenCL, PGI and HMPP directives
- DDT CUDA-enabled debugger
- PGI, GNU, Intel compilers

## Sample Dirac Batch Script

```
#PBS -q dirac-reg
#PBS –l nodes=16:ppn=8:fermi(or tesla)
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS –V

cd $PBS_O_WORKDIR
mpirun –n 128./my_executable
```

# Genepool – JGI Compute Resource



## Sample Genepool Batch Script

```
#$ -pe pe_slots 8
#$ -l h_rt=00:10:00
#$ -N my_job
#$ -V
#$ -cwd

export OMP_NUM_THREADS=8
./my_executable
```

## How does my job get out on the nodes?

- `qsub my_batch_script` will submit the job to the cluster
- You can run parallel jobs interactively with qsub –I and the appropriate resource requests

## Launching Parallel Jobs

- A “job launcher” executes your code
  - Distributes your executables to all your nodes
  - Starts concurrent execution of N instances of your program
  - Manages execution of your application
  - On Crays: the job launcher is called “aprun”
  - On Carver: “mpirun”
- Only the job launcher can start your job on compute nodes
- You can’t run the job launcher from login nodes

## Interactive Parallel Jobs

- You can run small parallel jobs interactively for up to 30 minutes

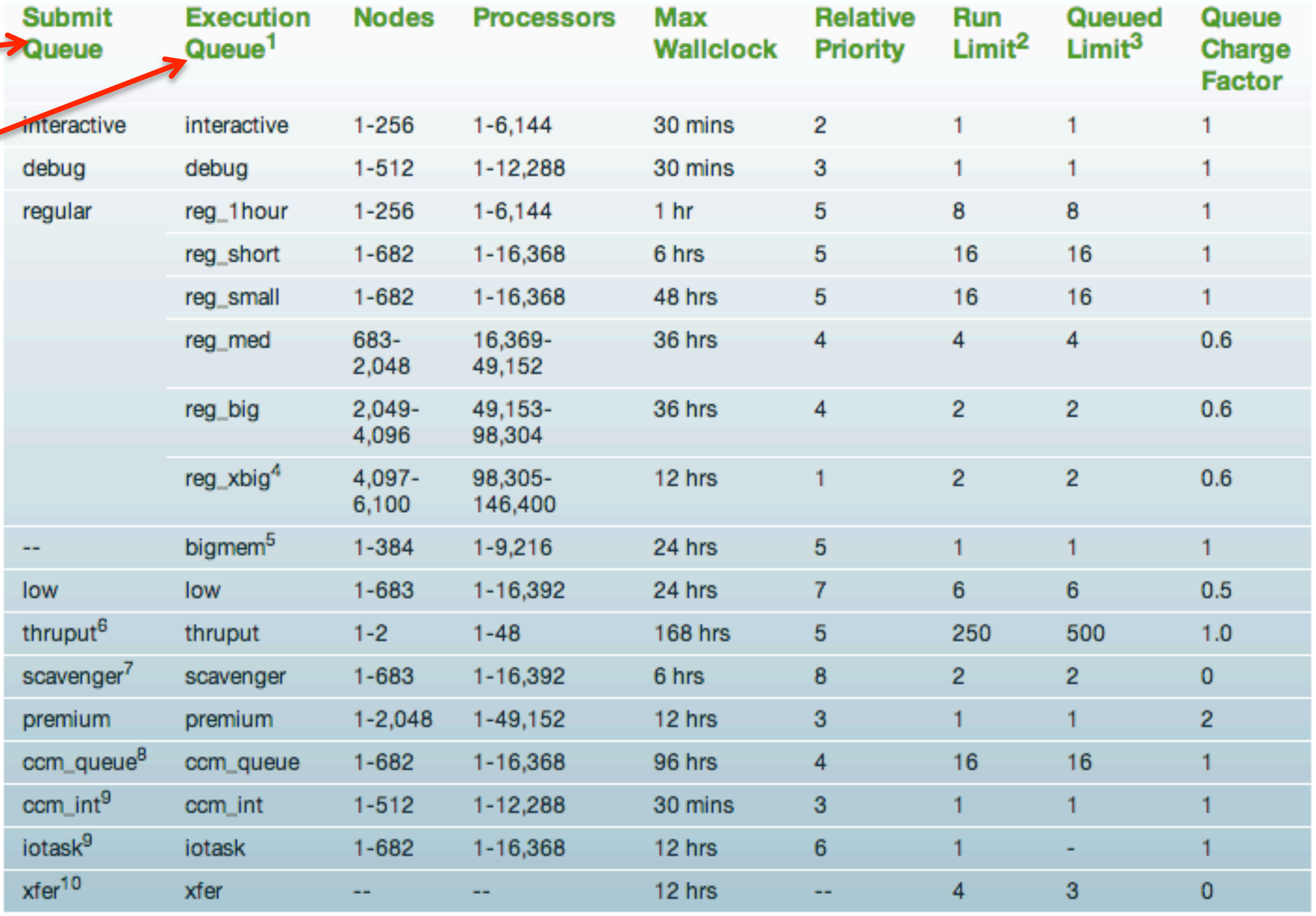
```
% qsub –I –V –lmppwidth=32
[wait for job to start]
% cd $PBS_O_WORKDIR
% aprun –n 32 ./mycode.x
```

# Job Limits

**There are per user, per machine job limits. See the NERSC web site for details. Here are the limits on Hopper as of Jan 16, 2013.**

Specify these queues (with –q queue_name)

NEVER these!



| Submit Queue | Execution Queue[1] | Nodes | Processors | Max Wallclock | Relative Priority | Run Limit[2] | Queued Limit[3] | Queue Charge Factor |
|---|---|---|---|---|---|---|---|---|
| interactive | interactive | 1-256 | 1-6,144 | 30 mins | 2 | 1 | 1 | 1 |
| debug | debug | 1-512 | 1-12,288 | 30 mins | 3 | 1 | 1 | 1 |
| regular | reg_1hour | 1-256 | 1-6,144 | 1 hr | 5 | 8 | 8 | 1 |
|  | reg_short | 1-682 | 1-16,368 | 6 hrs | 5 | 16 | 16 | 1 |
|  | reg_small | 1-682 | 1-16,368 | 48 hrs | 5 | 16 | 16 | 1 |
|  | reg_med | 683-2,048 | 16,369-49,152 | 36 hrs | 4 | 4 | 4 | 0.6 |
|  | reg_big | 2,049-4,096 | 49,153-98,304 | 36 hrs | 4 | 2 | 2 | 0.6 |
|  | reg_xbig[4] | 4,097-6,100 | 98,305-146,400 | 12 hrs | 1 | 2 | 2 | 0.6 |
| -- | bigmem[5] | 1-384 | 1-9,216 | 24 hrs | 5 | 1 | 1 | 1 |
| low | low | 1-683 | 1-16,392 | 24 hrs | 7 | 6 | 6 | 0.5 |
| thruput[6] | thruput | 1-2 | 1-48 | 168 hrs | 5 | 250 | 500 | 1.0 |
| scavenger[7] | scavenger | 1-683 | 1-16,392 | 6 hrs | 8 | 2 | 2 | 0 |
| premium | premium | 1-2,048 | 1-49,152 | 12 hrs | 3 | 1 | 1 | 2 |
| ccm_queue[8] | ccm_queue | 1-682 | 1-16,368 | 96 hrs | 4 | 16 | 16 | 1 |
| ccm_int[9] | ccm_int | 1-512 | 1-12,288 | 30 mins | 3 | 1 | 1 | 1 |
| iotask[9] | iotask | 1-682 | 1-16,368 | 12 hrs | 6 | 1 | - | 1 |
| xfer[10] | xfer | -- | -- | 12 hrs | -- | 4 | 3 | 0 |

## Tips for jobs

- Submit shorter jobs. If your application has the capability to checkpoint and restart, consider submitting your job for shorter time periods.
- On a system as large as Hopper there are many opportunities for backfilling jobs. Backfill is a technique the scheduler uses to keep the system busy. If there is a large job at the top of the queue the system will need to drain resources in order to schedule that job. During that time, short jobs can run. Jobs that qualify for reg_short are good candidates for backfill.
- Make sure the wall clock time you request is accurate. As noted above, shorter jobs are easier to schedule. Many users unnecessarily enter the largest wall clock time possible as a default.
- Run jobs before a system maintenance. A system must drain all jobs before a maintenance so there is an opportunity for good turn around for shorter jobs.

# How Your Jobs Are Charged

- **Your repository account is charged for each core your job was allocated for the entire duration of your job.**
  - The minimum allocatable unit is a node. Hopper has 24 cores/node, so your minimum charge on Hopper is 24*`walltime`.
  - e.g., `mppwidth=96` for 1 hour of run time is charged 96*1 hour = 96 MPP Hours (assuming the default setting of `mppnppn=24`)
  - You are charged for your actual run time, not the value of `walltime` in your batch script.
  - Serial jobs on Carver are charged for just a single core.
- **If you have access to multiple repos, pick which one to charge in your batch script**
  - `#PBS -A` *`repo_name`*

# Charge Factors & Discounts

- **Each machine has a "machine charge factor" (mcf) that multiplies the "raw hours" used**
  - Hopper has mcf=1.0
  - Carver has mcf=1.5
- **Queues have "priority charge factors" (pcf) and corresponding relative scheduling priorities**
  - Premium pcf=2.0
  - Low pcf=0.5
  - Everything else pcf=1.0
- **On Hopper only:**
  - reg_med, reg_big, reg_xbig jobs get a 40% discount
- **Storage and bandwidth are allocated and charged for HPSS**
  - Exhausting an HPSS allocation is rare
  - See the NERSC web site for details

Thank you for your attention!