BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Running Jobs on Blue Waters

Greg Bauer

NCSA

NSF

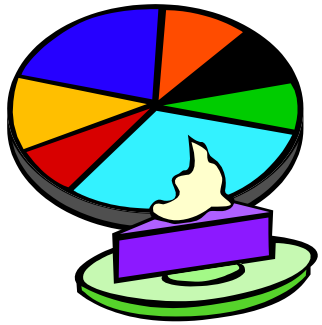
GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

CRAY

- Policies and Practices
- Placement
- Checkpointing
- Monitoring a job
- Getting a nodelist
- Viewing the torus

Resource and Job Scheduling Policies

- Runtime limits expected to be 12 to 24 hours.
- Fairshare within teams, and between teams.
- We will be implementing a QOS structure



QOS name	priority	charge factor
debug	1	tbd
high	2	tbd
(regular)	3	1
low	4	tbd

Specify with

```
#PBS –l qos=…
```

Run from …

- Scratch is the largest and fastest (1440 OSTs).
- All three filesystems are visible from compute nodes.
- Purging of scratch will be threshold based with a 30 day age based policy.
- Current issue limits stripe count to 160 OSTs.
- For a single OST, there is a 2 TB file limit.
- More in the I/O talk later in the workshop.

Blue Waters Job Submission Specifics

- Preferred

```
#PBS –l nodes=X:ppn=32:xe
#PBS –l nodes=Y:ppn=16:xk
#PBS –l nodes=X:ppn=32:xe+Y:ppn=16:xk
```

- Plan to support

```
#PBS –l procs=
#PBS –l hostlist=
```

- Plan to deprecate

```
#PBS –l size=
```

- No longer supporting `mpp*`

Batch job stdout/stderr

- Redirect application stdout/stderr to a file (preferably on scratch) rather than letting the batch software capture it.

```
> cd /scratch/…
> aprun -n … > outerr.$PBS_JOBID 2>&1 (bash)
```

- Easier to track job progress.
- Avoid excessive stdout and stderr from applications as task count increases. Consider reporting by rank 0 or a few ranks.

Application placement with aprun

- To list or not to list

```
aprun -n NPES -N 16 -d 2 …
aprun -n NPES -N 16 -cc 0,2,4,6,8,…  …
aprun -n NPES -S 4 -d 2
```

  All result in similar placement

- Even or odd cores
  - Tests with a couple applications suggest both give similar performance.

Application placement with aprun

Additional, useful options

- -S pes_per_numa_node

  Bind tasks per NUMA node

- -cc [core list|keyword]
  - Keyword: numa_node | none
  - Bind to cores of NUMA node, or no binding
- -ss

  allocate only the local, assigned NUMA node

Application placement with aprun

- Hybrid

```
OMP_NUM_THREADS=2
aprun -n NPES -N 16 -d 2 …
aprun -n NPES -N 16 -cc 0,1:2,3:… …
aprun -n NPES -S 4 -d 2 …
```

  All result in similar placement

- NUMA nodes and memory affinity

  An Interlagos processor has 2 NUMA nodes with 4 cores (8 integer cores) per node. "**First-touch**" is important.

Application placement with aprun

- MPMD

  aprun -n 1 -N 1 -cc 0 ./foo.exe : -n NPES -N 16 –d 2 ./foo.exe
- Useful when rank 0 has larger memory usage than remaining ranks.
- Executables share MPI_COMM_WORLD.

Core specialization

Core specialization binds a set of Linux kernel-space processes and daemons to one or more cores within a Cray compute node to enable the software application to fully utilize the remaining cores.

```
aprun -n NPES -r cores_per_node ...
```

- Two scenarios
  1. Fixed node count: fewer PEs
  2. Fixed PE count: more nodes
- Observations
  1. PSNAP measurements indicate cost of using 1/16th fewer cores & PEs can be offset by reduction in synchronous outliers.
  2. Hybrid application tests (WRF) show a modest improvement over just adding more nodes.

Checkpointing and resiliency

- Checkpoint
  - Frequency increases with node count.
- Request a couple extra nodes
  - alps marks nodes down if they fail.
  - J ob doesn't exit
  - Subsequent aprun will not use down nodes
- Use the "-R" option to aprun if your application works with a lost rank.
- Write job script to handle node failure
  - aprun returns 0 for successful exit

```
# launch application
 aprun …
# check if run was successful, if not keep running
 if [[ $? -ne 0 ]]; then
 aprun …
 fi
```

Atypical job models

- Running multiple, concurrent apruns
  - 100s-1000s of apruns in a job
  - It is documented, more or less

    http://docs.cray.com/books/S-2496-4101/html-S-2496-4101/cnlexamples.html#section.9.WvkIRDBp
  - Does bundle lots of small workloads.
  - Demanding on MOM node resources.
- Multiple, wide but short jobs back-to-back
  - Job scheduler overhead impacts utilization.
  - Better to merge jobs: series of apruns.

Monitoring jobs

System commands

- qstat, showq – torque and moab
- apstat, xtnodestat – alps

Scripts

- qpeek – look at job stdout/stderr
- apstat_system.pl – displays system status by node type
- qstat.pl – displays qstat output with node type and count
- showqgpu.pl - displays only XK jobs similar to showq

Why isn't my job running?

- Make sure job was submitted they way you expected: qstat –f JOBID
  - We employ a qsub filter that attempts to catch improperly configured requests.
- Check to see what is running with showq as eligible jobs are listed in a prioritized order. Using the '-i' option will show priority.
- Use 'checkjob JOBID' to see more about the job.
- Currently have a 300 second allocation cycle.

Obtaining nid list

- Will be available via portal
- For running job

```
> apstat -n -R `apstat -r | grep 262704 | grep ^A | cut -d" " -f2`
```

or

```
> checkjob 262704
```

- In batch job script
  - `MPICH_CPUMASK_DISPLAY`
- In application (`#include "pmi.h"`)
  - `PMI_Get_nidlist_ptr()`
  - `PMI_Get_meshcoord()` - location in torus

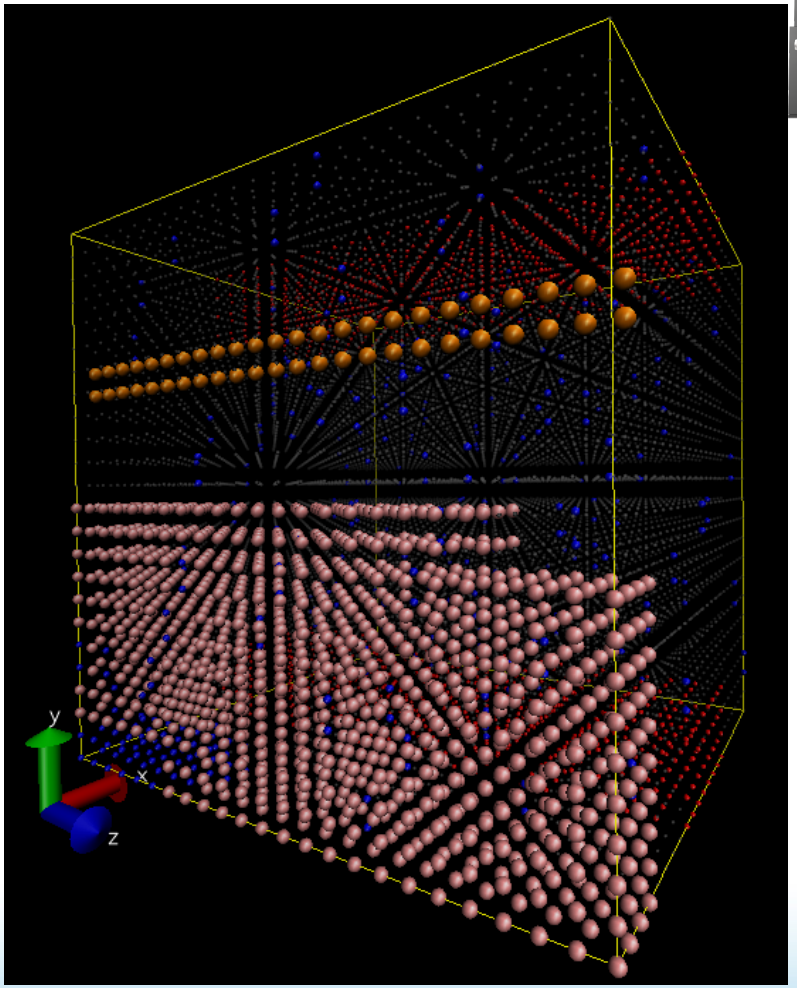
See `/opt/cray/pmi/default/include/pmi_cray_ext.h`

Node list ordering

- Currently the order of nodes is based on the xyz-by2 NID reordering method, units of 2x2x2, which is a merger of the small node packing of the simple NID number method and the inter-application interaction reduction of the "xyz" method.
- List of nodes handed over to torque/moab.

- Workload Management and Application Placement for the Cray Linux Environment - http://docs.cray.com/books/S-2496-4101/html-S-2496-4101/cnl_apps.html

View of Gemini torus



- Torus dimension 23x24x24
- Reentrant or periodic
  - No corners in a torus
- Grey – compute nodes
- Blue – service nodes
- Red – 3072 XK nodes
- Orange – 96 XE nodes in a cabinet (2x24 by 2)
- Pink – nodes in a 2048 XE node job
- To be in portal.

- Questions?