HPC User Environment 2

Brought to you by

Wei Feinstein,
Feng Chen, James Lupo, Le Yan,
Shaohao Chen & Xiaoxu Guan

HPC User Services
LSU HPC/LONI

Louisiana State University
Outline

◆ Recap of User Environment 1 training

◆ Topics to be covered today
  – Job management on clusters
    • Job priority
    • Backfill
  – Compiling and Analyzing programs
    • Serial program
    • Parallel program
    • Programs using accelerators
Available Computing Resources

- University: HPC@LSU
- State: LONI
- Nation: XSEDE
Available Clusters

◆ University: HPC@LSU
  superMike-2, superMIC, philip,
  shelob & pandora

◆ State: LONI
  QB-2 & Eric

◆ Nation: XSEDE
  superMIC
HPC Cluster Architectures

Two major HPC architectures

- Intel x86_64 clusters: SuperMike, SuperMIC, QB-2, Eric, Philip
  - Vendor: Dell
  - Operating System: Linux (RHEL 4/5/6)
  - Processor: Intel
- IBM PowerPC clusters: pandora
  - Vendor: IBM
  - Operating System: AIX
  - Processor: IBM power7
You have learned how to …

- Apply HPC/LONI accounts and allocations
- Reset passwd
- Login clusters from Linux/Mac/Window machines
  Window: Putty/MobaXterm
- File transfer
- Software management on clusters
  - Module: SuperMIC, QB-2
  - Softenv: SuperMike2, Eric, Philip, Shelob
What is a HPC cluster?
# Basic HPC cluster resource terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>The top-level organizational unit of an HPC cluster, comprising a set of nodes, a queue, and jobs.</td>
</tr>
<tr>
<td>Node</td>
<td>A single, named host in the cluster.</td>
</tr>
<tr>
<td>Core</td>
<td>An individual CPU on a node. For example, a quad-core processor is considered 4 cores.</td>
</tr>
<tr>
<td>Job</td>
<td>A user's request to use a certain amount of resources for a certain amount of time on cluster for the work.</td>
</tr>
</tbody>
</table>
Cluster Environment

- Multiple users may use multiple compute nodes.
- Each user may have multiple jobs running simultaneously.
- Multiple users may share the same node.

Software for managing and scheduling jobs is required.
Resource Manager/ Job Scheduler

- Software to manage resources (CPU time, memory etc.) and management workload
  - Linux clusters: Portable Batch System (PBS)
  - AIX clusters: Loadleveler
- HPC/LONI:
  - Torque: resource manager (open-source version of PBS)
  - Moab: job scheduler
  - Gold: allocation manager
- A batch queuing system determines
  - order of queuing jobs to be executed
  - which node(s) to be assigned for jobs
Job management basics

- Find appropriate queue: `qstat -q`
- Submit jobs
- Monitor jobs
- Understand the queuing system and requirements
Queue Characteristics – LONI clusters

<table>
<thead>
<tr>
<th>Machine</th>
<th>Queue</th>
<th>Max Runtime</th>
<th>ppn</th>
<th>Max running jobs</th>
<th>Max nodes per job</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eric</td>
<td>workq</td>
<td>3 days</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>Unpreemptable</td>
</tr>
<tr>
<td></td>
<td>checkpoint</td>
<td></td>
<td>8</td>
<td></td>
<td>48</td>
<td>Preemptable</td>
</tr>
<tr>
<td></td>
<td>single</td>
<td></td>
<td>1</td>
<td>32</td>
<td>1</td>
<td>ppn &lt; =8</td>
</tr>
<tr>
<td>QB2</td>
<td>workq</td>
<td>3 days</td>
<td>20</td>
<td>44</td>
<td>128</td>
<td>Unpreemptable</td>
</tr>
<tr>
<td></td>
<td>checkpoint</td>
<td></td>
<td>20</td>
<td></td>
<td>256</td>
<td>Preemptable</td>
</tr>
<tr>
<td></td>
<td>single</td>
<td>7 days</td>
<td>&lt;=20</td>
<td></td>
<td>1</td>
<td>Single-node jobs</td>
</tr>
<tr>
<td></td>
<td>bigmem</td>
<td>3 days</td>
<td>12,24,36,48</td>
<td>4</td>
<td>1</td>
<td>Big-memory jobs</td>
</tr>
</tbody>
</table>

Typically, workq and checkpoint queues are for parallel jobs, while single queue is for serial jobs. The ppn of workq and checkpoint has to be equal to the number of cores per node. Single queue can be also used for parallel jobs with the number of threads that is less than the number of cores per node.
## Queue Characteristics – LSU Linux clusters

<table>
<thead>
<tr>
<th>Machine</th>
<th>Queue</th>
<th>Max Runtime</th>
<th>ppn</th>
<th>Max running jobs</th>
<th>Max nodes per job</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperMike II</td>
<td>workq</td>
<td>3 days</td>
<td>16</td>
<td>34</td>
<td>128</td>
<td>Unpreemptable</td>
</tr>
<tr>
<td></td>
<td>checkpoint</td>
<td>3 days</td>
<td>16</td>
<td></td>
<td>128</td>
<td>Preemptable</td>
</tr>
<tr>
<td></td>
<td>bigmem</td>
<td>2 days</td>
<td>16</td>
<td></td>
<td>1</td>
<td>Big memory</td>
</tr>
<tr>
<td></td>
<td>gpu</td>
<td>3 days</td>
<td>16</td>
<td></td>
<td>16</td>
<td>Job using GPU</td>
</tr>
<tr>
<td></td>
<td>single</td>
<td>3 days</td>
<td>&lt;=4</td>
<td></td>
<td>1</td>
<td>Single node jobs</td>
</tr>
<tr>
<td>Philip</td>
<td>workq</td>
<td>3 days</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>Unpreemptable</td>
</tr>
<tr>
<td></td>
<td>checkpoint</td>
<td>3 days</td>
<td>8</td>
<td></td>
<td>4</td>
<td>Preemptable</td>
</tr>
<tr>
<td></td>
<td>bigmem</td>
<td>8</td>
<td>8</td>
<td></td>
<td>2</td>
<td>Big memory jobs</td>
</tr>
<tr>
<td></td>
<td>single</td>
<td>14 days</td>
<td>4</td>
<td>50</td>
<td>1</td>
<td>Single processor</td>
</tr>
<tr>
<td>SuperMIC</td>
<td>workq</td>
<td>3 days</td>
<td>20</td>
<td>34</td>
<td>128</td>
<td>Unpreemptable</td>
</tr>
<tr>
<td></td>
<td>checkpoint</td>
<td>3 days</td>
<td>20</td>
<td></td>
<td>360</td>
<td>Preemptable</td>
</tr>
</tbody>
</table>
Job management basics

- Find appropriate queue: `qstat -q`
- Submit jobs
- Monitor jobs
- Understand the queueing system and requirements
Two Job Types

◆ Interactive job
  – Purpose: testing, debugging, compiling
  – Set up an interactive environment on compute nodes for users
    • Advantage: can run programs interactively
    • Disadvantage: must be present when the job starts
    • Try not to run interactive jobs with large core count

◆ Batch job
  – Purpose: production run
  – Executed without user intervention using a job script
    • Advantage: the system takes care of everything
    • Disadvantage: user control is surrendered

Do not run jobs on the head node!!!
Submitting Jobs (interactive job)

qsub -I -V \\ 
-1 walltime=<hh:mm:ss> \\ 
-1 nodes=<num_nodes>:ppn=<num_cores> \\ 
-A <Allocation> \\ 
-q <queue name> \\ 
-X <enable X11 forwarding>

DO NOT directly ssh to compute nodes,
unless the nodes assigned to you by the job scheduler.
Submitting Jobs (Batch)

#!/bin/bash
#PBS -l nodes=1:ppn=1    # Number of nodes and processor
#PBS -l walltime=24:00:00  # Maximum wall time
#PBS -N myjob         # Job name
#PBS -o <file name>   # File name for standard output
#PBS -e <file name>   # File name for standard error
#PBS -q single       # The only queue accepts serial jobs
#PBS -A <loni_allocation>    # Allocation name
#PBS -m bea          # Send mail when job begin/end/abort
#PBS -M <email address>  # Send mail to this address

<shell commands>
/path/to/executable <options>
<shell commands>

Batch Job submission: qsub job_script
Job management basics

- Find appropriate queue: `qstat -q`
- Submit jobs
- Monitor jobs
- Understand the queuing system and requirements
Queue Querying – Linux Clusters

“showq”: information about active, eligible, blocked, recently completed jobs

$ showq
active jobs------------------------

<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>REMAINING</th>
<th>STARTTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>236875</td>
<td>ebeigi3</td>
<td>Running</td>
<td>16</td>
<td>1:44:29</td>
<td>Mon Sep 15 20:00:22</td>
</tr>
<tr>
<td>236934</td>
<td>mwu3</td>
<td>Running</td>
<td>16</td>
<td>00:03:27</td>
<td>Mon Sep 15 19:04:20</td>
</tr>
</tbody>
</table>
...

eligible jobs------------------------

<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>WCLIMIT</th>
<th>QUEUETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>236795</td>
<td>dmarcel</td>
<td>Idle</td>
<td>1456</td>
<td>00:15:00</td>
<td>Mon Sep 15 16:38:45</td>
</tr>
<tr>
<td>236753</td>
<td>rsmith</td>
<td>Idle</td>
<td>2000</td>
<td>4:00:00</td>
<td>Mon Sep 15 14:44:52</td>
</tr>
<tr>
<td>236862</td>
<td>dlamas1</td>
<td>Idle</td>
<td>576</td>
<td>2:00:00</td>
<td>Mon Sep 15 17:28:57</td>
</tr>
</tbody>
</table>
...

121 eligible jobs

blocked jobs------------------------

<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>WCLIMIT</th>
<th>QUEUETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>232741</td>
<td>myagho1</td>
<td>Idle</td>
<td>2000</td>
<td>1:00:00</td>
<td>Mon Sep 8 07:22:12</td>
</tr>
<tr>
<td>235545</td>
<td>tanping</td>
<td>Idle</td>
<td>1</td>
<td>2:21:10</td>
<td>Fri Sep 12 16:50:49</td>
</tr>
<tr>
<td>235546</td>
<td>tanping</td>
<td>Idle</td>
<td>1</td>
<td>2:21:10</td>
<td>Fri Sep 12 16:50:50</td>
</tr>
</tbody>
</table>
...
Job Monitoring - Linux Clusters

- Check details on a job
  
  qstat -n -u $USER : details of node assignment
  qstat -f jobid : details on your job

- Delete a job
  
  qdel jobid : delete job

- Check approximate start time
  
  showstart jobid

- Check details of a job
  
  checkjob jobid

- Check health of a job
  
  qshow -j jobid
# PBS Environmental Variables

```bash
$ echo $PBS_XXX
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Variable</th>
<th>Variable</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PBS_ENVIRONMENT</td>
<td>$PBS_MOMPORT</td>
<td>$PBS_NUM_PPN</td>
<td>$PBS_O_MAIL</td>
</tr>
<tr>
<td>$PBS_QUEUE</td>
<td>$PBS_WALLTIME</td>
<td>$PBS_GPUFILE</td>
<td>$PBS_TASKNUM</td>
</tr>
<tr>
<td>$PBS_SERVER</td>
<td>$PBS_NODEFILE</td>
<td>$PBS_O_HOME</td>
<td>$PBS_O_PATH</td>
</tr>
<tr>
<td>$PBS_JOBCOOKIE</td>
<td>$PBS_NODENUM</td>
<td>$PBS_O_HOST</td>
<td>$PBS_O_QUEUE</td>
</tr>
<tr>
<td>$PBS_JOBID</td>
<td>$PBS_JOBNAME</td>
<td>$PBS_O_LOGNAME</td>
<td>$PBS_O_WORKDIR</td>
</tr>
<tr>
<td>$PBS_VERSION</td>
<td>$PBS_NP</td>
<td>$PBS_O_LANG</td>
<td>$PBS_O_SHELL</td>
</tr>
<tr>
<td>$PBS_VNODENUM</td>
<td>$PBS_NUM_NODES</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
“top” command

- Pay attention to load and the memory consumption of your jobs!
- “ssh” to assigned compute nodes
- "top" to obtain a real-time view of the running jobs.

```
top - 19:39:56 up 89 days,  4:13,  1 user,  load average: 0.63, 0.18, 0.06
Tasks: 489 total,   2 running, 487 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.3%us,  0.0%sy,  0.0%ni, 93.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   65909356k total,  3389616k used, 62519740k free, 151460k buffers
Swap:  207618040k total,  5608k used, 207612432k free,  947716k cached

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>39595</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>266m</td>
<td>257m</td>
<td>592</td>
<td>R</td>
<td>99.9</td>
<td>0.4</td>
<td>0:06.94</td>
<td>a.out</td>
</tr>
<tr>
<td>39589</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>17376</td>
<td>1612</td>
<td>980</td>
<td>R</td>
<td>0.3</td>
<td>0.0</td>
<td>0:00.05</td>
<td>top</td>
</tr>
<tr>
<td>38479</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>108m</td>
<td>2156</td>
<td>1348</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.03</td>
<td>bash</td>
</tr>
<tr>
<td>39253</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>103m</td>
<td>1340</td>
<td>1076</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>236297.mike3.SC</td>
</tr>
<tr>
<td>39254</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>103m</td>
<td>1324</td>
<td>1060</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>bm_laplace.sh</td>
</tr>
<tr>
<td>39264</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>99836</td>
<td>1908</td>
<td>992</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>sshd</td>
</tr>
<tr>
<td>39265</td>
<td>fchen14</td>
<td>20</td>
<td>0</td>
<td>108m</td>
<td>3056</td>
<td>1496</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.03</td>
<td>bash</td>
</tr>
</tbody>
</table>
```
Pay attention to single queue usage

- Single queue – jobs only execute on a part of a single node i.e. `nodes=1:ppn=1/2/4/8`
- Memory concern --- jobs in the single queue should not use:
  - more than 2GB ( = 32G / 16) memory per core on Eric, Philip, Pandora and SuperMike II
  - more than 3.2GB ( = 64G / 20) memory per core on QB2 and SuperMIC
  - Scale up # cores (ppn) if more memory is required
- Typical warnings:
  - E124 - Exceeded memory allocation. This Job XXXX appears to be using more memory (GB) than allocated (9 > 3).
  - E123 - Exceeded ppn/core allocation. This Job XXXX appears to be using more cores than allocated (6 > 1). Please allocate the number of cores that the job will use, (ppn=6). This Job has 1 core(s) allocated (ppn=1).
More points...

◆ Service unit (SU) = (# cores) * (actual wall-time)
  - also apply to clusters with accelerators (GPU/Phi)

◆ Eric will be retired in the near future, migrating codes to QB-2 should be taken into consideration for LONI users

◆ Bigmem queue on QB-2 is for jobs need > 64 GB memory, not for jobs using more number of cores

◆ Accelerator usage is highly encouraged.
  Applications requesting such allocations get approved first
Outline

◆ Recap of User Environment 1

◆ Topics to be covered today
  – Job management on clusters
    • Job priority
    • Backfill
  – Compiling and Analyzing programs
    • Serial programs
    • Parallel programs
    • Programs using accelerators
Back to Cluster Architecture

- Resource/scheduling managers
  - Take resource request (job) from the login node
  - Check available resources and assign a priority number to the job
  - Queue the job
  - Start the job if available resources match the request
Resource manager philosophy

Maximize the usage of a cluster
- Prioritize workload (jobs) into a queue
- Backfill idle nodes to maximize utilization
Job Priority

- Not “First Come First Served”
- Jobs with higher priorities scheduled first
- Factors determine job priority:
  - credential priority
  - fairshare priority
  - resource priority
  - service priority
- Priority determination for queued/waiting jobs
  • mdiag -p
Job Priorities

$ mdiag -p

diagnosing job priority information (partition: ALL)

<table>
<thead>
<tr>
<th>Job</th>
<th>PRIORITY*</th>
<th>Cred( User:Class)</th>
<th>FS( User: WCA)</th>
<th>Serv(QTime:XFctr)</th>
<th>Res( Proc)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Weights</td>
<td>100( 10: 10)</td>
<td>100( 10: 50)</td>
<td>2( 2: 20)</td>
<td>30( 10)</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>-------------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>236172</td>
<td>246376</td>
<td>40.6(100.0: 0.0)</td>
<td>8.6( 19.6: 0.3)</td>
<td>4.0(1480.: 99.7)</td>
<td>46.8(2048.)</td>
</tr>
<tr>
<td>235440</td>
<td>242365</td>
<td>41.3(100.0: 0.0)</td>
<td>4.6( 8.2: 0.6)</td>
<td>6.6(3959.: 6.5)</td>
<td>47.5(512.0)</td>
</tr>
<tr>
<td>235441</td>
<td>242365</td>
<td>41.3(100.0: 0.0)</td>
<td>4.6( 8.2: 0.6)</td>
<td>6.6(3959.: 6.5)</td>
<td>47.5(512.0)</td>
</tr>
<tr>
<td>235442</td>
<td>242361</td>
<td>41.3(100.0: 0.0)</td>
<td>4.6( 8.2: 0.6)</td>
<td>6.6(3958.: 6.5)</td>
<td>47.5(512.0)</td>
</tr>
<tr>
<td>236396</td>
<td>241821</td>
<td>41.4(100.0: 0.0)</td>
<td>8.8( 19.6: 0.3)</td>
<td>2.2(664.0: 67.4)</td>
<td>47.6(1456.)</td>
</tr>
</tbody>
</table>
Priority components

- **Credential priority:** a constant
  
  \[
  \text{credweight} \times (\text{userweight} \times \text{job.user.priority}) = 100 \times (10 \times 100) = 100000
  \]

- **Fairshare priority:** usage in the last 7 days
  
  \[
  \text{fsweight} \times \min (\text{fscap}, (\text{fsuserdata} \times \text{DeltaUserFSUsage})) = 100 \times (10 \times 20),
  \]
  
  where \(\max(\text{DeltaUserFSUsage}) = 20\)

- **Service priority:** queuing time / wall-time
  
  \[
  \text{serviceweight} \times (\text{queuetimeweight} \times \text{QUEUETIME} + \text{xfactorweight} \times \text{XFACTOR}) = 2 \times (2 \times \text{QUEUETIME} + 20 \times \text{XFACTOR}), \text{ where XFACTOR} = 1 + \frac{\text{QUEUETIME}}{\text{WALLTIMELIMIT}}
  \]

- **Resource priority:** # processors requested
  
  \[
  \text{resweight} \times \min (\text{rescap}, (\text{procweight} \times \text{TotalProcessorsRequested})) = 30 \times \min (3840, (10 \times \text{TotalProcessorsRequested}))
  \]

[http://www.hpc.lsu.edu/docs/pbs.php]
How to get higher priority?

- Request more compute nodes
- Request a smaller walltime
- Do not submit too many jobs within one week
- Submit your job early to accumulate the queue time
How to maximize the usage of a cluster?

- Fill in high-priority (large) jobs
- Backfill low-priority (small) jobs
Backfilling

◆ FIRSTFIT
  ▪ Filter the list of feasible backfill jobs and select those fit in the current backfill window
  ▪ Start the first fitting job

◆ Scheduling optimization allows a job schedule to
  ▪ Running jobs out of order to better utilize available resources
  ▪ So long as NOT delay the highest priority job in the queue.
Backfilling

“showbf”

- Display available resources for immediate usage.
- Use this information to customize job submission script in order to obtain a quick job turnaround

$showbf -c workq

<table>
<thead>
<tr>
<th>Partition</th>
<th>Tasks</th>
<th>Nodes</th>
<th>Duration</th>
<th>StartOffset</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>40</td>
<td>5</td>
<td>18:50:35</td>
<td>00:00:00</td>
<td>11:16:49_09/04</td>
</tr>
<tr>
<td>ALL</td>
<td>8</td>
<td>1</td>
<td>INFINITY</td>
<td>00:00:00</td>
<td>11:16:49_09/04</td>
</tr>
</tbody>
</table>
How much time and how many nodes?

- Long enough for your job to complete, as short as possible to increase the chance of backfilling

- Enough nodes to complete your job, as less nodes as possible
Frequently Asked Questions

1) I submitted job A before job B. Why did job B start earlier than job A?

2) There are free nodes available, why is my job still waiting?

3) Why my job does not get accelerated when using a cluster?
   1. does your job utilize the parallel resource on the cluster?
   2. does you job have lots of I/O tasks?
   3. see next section...
Outline

◆ Recap of User Environment 1

◆ Topics to be covered today
  – More on job management
    • Job priority
    • Backfill
  – Compiling and Analyzing programs
    • Serial programs
    • Parallel programs
    • Programs using accelerators
## Compilers

### Serial compilers

<table>
<thead>
<tr>
<th>Language</th>
<th>Linux cluster</th>
<th>AIX cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>ifort, pgf77, pgf90</td>
<td>gfortran, xlf, xlf90</td>
</tr>
<tr>
<td>C</td>
<td>icc, pgcc</td>
<td>gcc, xlc</td>
</tr>
<tr>
<td>C++</td>
<td>icpc, pgCC</td>
<td>g++, xIC</td>
</tr>
</tbody>
</table>

### Parallel compilers

<table>
<thead>
<tr>
<th>Language</th>
<th>Linux cluster</th>
<th>AIX cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>mpif77, mpif90</td>
<td>mpxlf, mpxlf90</td>
</tr>
<tr>
<td>C</td>
<td>mpicc</td>
<td>mpcc</td>
</tr>
<tr>
<td>C++</td>
<td>mpiCC</td>
<td>mpCC</td>
</tr>
</tbody>
</table>
Compiling and Analyzing C serial program

```c
#include <stdio.h>
#include <math.h>
#include <time.h>

int main(char *argc, char **argv) {
    double s=0.0;
    clock_t start, end;
    int i;
    start = clock();
    for (i=0;i<1000000000;i++)
        s = i / 2.0 * 4.9; // do some floating point operations
    end = clock();
    double time_computing_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
    printf("cputime_in_sec: %e\n", time_elapsed_in_seconds);
    start = clock();
    system ("sleep 5"); // just sleep, does this accumulate CPU time?
    end = clock();
    time_sleeping_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
    printf("\ncputime_in_sec: %e\n", time_elapsed_in_seconds);
    return 0;
}
```
Watch the actual cpu time

$ gcc hello.c
$ time ./a.out

time_computing_in_sec: 4.540000e+00
time_sleeping_in_sec: 0.000000e+00

Real 0m9.547s
user 0m4.543s
sys 0m0.002s

$ gfortran hello.f90
CPU time vs Elapsed time

◆ CPU time (or process time):
  – CPU time for processing instructions of a computer program or operating system.

◆ Elapsed real time (real time /wall clock time)
  – Time taken from the start of a computer program until the end including I/O time and all other types of waits incurred by the program.

◆ If a program uses parallel processing, total CPU time for that program would be more than its elapsed real time.
  – \( \frac{(\text{Total CPU time})}{(\text{Number of CPUs})} \) would be same as elapsed real time if work load is evenly distributed on each CPU and no wait is involved for I/O or other resources.
Parallel schemes

**Shared memory**
- Single multicore node
- Multi-threading (OpenMP)

**Distributed memory**
- Multiple compute nodes
- Message Passing (MPI)

---

**Shared Memory**
- Cache-Coherent Interconnect

- Cache
- P

...  

**Non-Cache-Coherent Interconnect**

- P
- Memory

...  

- P
- Memory
**Compiling OpenMP code**

- **-openmp flag is required to compile OpenMP codes**
- **export** `OMP_NUM_THREADS= # threads`
- **Examples:**
  ```
gcc -fopenmp hello_openmp.c
ifort -openmp hello_openmp.f90
  ```

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Compiler flag</th>
<th>Default thread # (OMP_NUM_THREADS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU (gcc, g++, gfortran)</td>
<td>-fopenmp</td>
<td># threads = available cores</td>
</tr>
<tr>
<td>Intel (icc, icpc, ifort)</td>
<td>-openmp</td>
<td># threads = available cores</td>
</tr>
<tr>
<td>Portland Group (pgcc, pgCC, pgf77, pgf90)</td>
<td>-mp</td>
<td>one thread</td>
</tr>
</tbody>
</table>
Sample OpenMP - C code

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    int nthreads, tid;
    /* Fork a team of threads with their own copies of variables */
#pragma omp parallel private(nthreads, tid)
    {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);
        /* Only master thread does this */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
        /* All threads join master thread and disband */
    }
}
Sample OpenMP - Fortran code

program hello

integer nthreads,tid,omp_get_num_threads,omp_get_thread_num
! fork a team of threads giving them their own copies of variables
!$omp parallel private(nthreads, tid)
! obtain thread number
  tid = omp_get_thread_num()
  print *, 'hello world from thread = ', tid
! only master thread does this
  if (tid .eq. 0) then
    nthreads = omp_get_num_threads()
    print *, 'number of threads = ', nthreads
  end if
! all threads join master thread and disband
!$omp end parallel
end
# Available MPI libraries on LONI & HPC

<table>
<thead>
<tr>
<th>Cluster Name</th>
<th>Mvapich</th>
<th>Mvapich2</th>
<th>OpenMPI</th>
<th>IMPI</th>
<th>MPICH</th>
<th>MPICH2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eric</td>
<td>0.98</td>
<td>1.4</td>
<td>1.3.4</td>
<td>X</td>
<td>X</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.8.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QB2</td>
<td>X</td>
<td>2.0</td>
<td>1.8.1</td>
<td>4.1.3.048</td>
<td>3.0.3</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Super-Mikell</td>
<td>X</td>
<td>1.9</td>
<td>1.6.2</td>
<td>4.1.3.048</td>
<td>3.0.2</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0.1</td>
<td>1.6.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.1</td>
<td>1.6.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philip</td>
<td>X</td>
<td>X</td>
<td>1.4.3</td>
<td>X</td>
<td>1.2.7</td>
<td>1.3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.6.1</td>
<td></td>
<td></td>
<td>1.4.1</td>
</tr>
<tr>
<td>SuperMIC</td>
<td>X</td>
<td>2.0</td>
<td>1.8.1</td>
<td>4.1.3.048</td>
<td>3.0.3</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## MPI Compilers (1)

<table>
<thead>
<tr>
<th>Language</th>
<th>Linux clusters</th>
<th>AIX clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>mpif77, mpif90</td>
<td>mpxlf, mpxlf90</td>
</tr>
<tr>
<td>C</td>
<td>mpicc</td>
<td>mpcc</td>
</tr>
<tr>
<td>C++</td>
<td>mpiCC</td>
<td>mpCC</td>
</tr>
</tbody>
</table>

- **Check current MPI implementation**
  
  ```
  $ which mpicc
  /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/bin/mpicc
  ```

- **Compile MPI programs**
  
  ```
  $ mpif90 hello.90  
  $ mpicc hello.c
  ```
MPI Compilers (2)

◆ MPI compilers are wrappers of
  – Intel/ PGI/ GNU compiler
  – Combined with header files/ libraries needed to build MPI codes
    – mpicc/mpif90-show for details

◆ Please compile and run your code with the same version of MPI!
Compiling MPI programs

Always verify what compiler/library is being used:

$ mpicc -show
  icc -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include
  -L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -lmpi
  -ldl -lm -Wl,--export-dynamic -lrt -lns1 -libverbs -libumad
  -lpthread -lutil

$ mpif90 -show
  ifort -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include
  -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib
  -L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -lmpi_f90 -lmpi_f77 -lmpi -ldl -lm -Wl,--export-dynamic -lrt -lns1 -libverbs - libumad -lpthread -lutil
Compiling a MPI C program

mpicc hello_mpi.c

#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int name_len, world_size, world_rank;
    char processor_name[ MPI_MAX_PROCESSOR_NAME ];
    //Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Get the number and rank of processes
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the name of the processor
    MPI_Get_processor_name(processor_name, &name_len);
    // Print off a hello world message
    printf("I am from processor %s, rank %d out of %d processors \n", processor_name, world_rank, world_size);
    // Finalize the MPI environment.
    MPI_Finalize();
}
Compiling a MPI Fortran program

```fortran
mpif90 hellp_mpi.f90

program hello_mpi
  include 'mpif.h'
  !use mpi
  character 10 name
  ! Initialize the MPI library:
  call MPI_Init(ierr)
  ! Get size and rank
  call MPI_Comm_Size(MPI_COMM_WORLD, numtasks, ierr) call
  MPI_Comm_Rank(MPI_COMM_WORLD, rank, ierr)
  ! print date
  if (nrank == 0) then
    write( , )'System date' call
    system('date')
  endif
  call MPI_Barrier(MPI_COMM_WORLD, ierr)
  ! print rank
  call MPI_Get_Processor_Name(name, len, ierr)
  write( , )'I am ', nrank, 'of', numtasks, 'on ', name
  ! Tell the MPI library to release all resources it is using: call
  MPI_Finalize(ierr)
end program hello_mpi
```
Compiling MPI programs (2)

Check which libraries are used

$ ldd a.out

# ldd - print shared library dependencies

    linux-vdso.so.1 => (0x000007fff907ff000)
    libmpi_f90.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib/libmpi_f90.so.1 (0x000002b9ae577e000)
    libmpi_f77.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib/libmpi_f77.so.1 (0x000002b9ae5982000)
    libmpi.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib/libmpi.so.1 (0x000002b9ae5bb9000)
...
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00000003b2180000)
...
    libifport.so.5 =>
/usr/local/compilers/Intel/composer_xe_2013.0.079/compiler/lib/intel64/libifport.so.5 (0x000002b9ae61ee000)
    libifcore.so.5 =>
/usr/local/compilers/Intel/composer_xe_2013.0.079/compiler/lib/intel64/libifcore.so.5 (0x000002b9ae641d000)
PBS Job Script – run an MPI Job

#!/bin/bash

#PBS -l nodes=2:ppn=16
#PBS -l walltime=02:30:00
#PBS -N myjob
#PBS -o test.out
#PBS -e test.err
#PBS -q checkpt
#PBS -A hpc_train_2015

export NPROCS=`wc -l $PBS_NODEFILE | gawk '/{print $1}'`

cd $PBS_O_WORKDIR

mpirun -machinefile $PBS_NODEFILE -np $NPROCS ./hello_mpi

“mpirun” could be different for different MPI implementations. Use “mpirun --help” to check.
Testing a MPI program interactively

- Make sure you are running your jobs on the correct nodes
- Important if you want to run less processes than ppn
- Understand the usage of $PBS_NODEFILE

```bash
$ qsub -I -X -l nodes=2:ppn=16 -l walltime=01:00:00 -q gpu
$ echo $PBS_NODEFILE
/var/spool/torque/aux///236660.mike3

[user@mike429 ~]$ cat $PBS_NODEFILE
mike429
  ...
# 16 repeats of mike429
mike429
mike430
  ...
# 16 repeats of mike430
mike430

[user@mike429 hybrid]$ cat $PBS_NODEFILE | uniq > hosts
[user@mike429 hybrid]$ cat hosts
mike429
mike430
```

9/16/2015  HPC User Environment 2
Running and Analyzing MPI program

[user@mike315 mpi]$ mpicc hello_mpi.c
[user@mike315 mpi]$ mpirun -np 32 -hostfile $PBSNODEFILE ./a.out
I am from processor mike315, rank 1 out of 32 processors I am from processor mike315, rank 6 out of 32 processors I am from processor mike315, rank 9 out of 32 processors I am from processor mike315, rank 12 out of 32 processors I am from processor mike315, rank 0 out of 32 processors I am from processor mike315, rank 2 out of 32 processors I am from processor mike315, rank 3 out of 32 processors I am from processor mike315, rank 7 out of 32 processors I am from processor mike315, rank 10 out of 32 processors I am from processor mike315, rank 5 out of 32 processors I am from processor mike315, rank 13 out of 32 processors I am from processor mike315, rank 4 out of 32 processors I am from processor mike315, rank 8 out of 32 processors I am from processor mike334, rank 17 out of 32 processors I am from processor mike315, rank 11 out of 32 processors I am from processor mike315, rank 14 out of 32 processors I am from processor mike315, rank 15 out of 32 processors I am from processor mike334, rank 18 out of 32 processors
Compiling hybrid (MPI+OpenMP) program

- $ mpicc -openmp hello_hybrid.c

```c
#pragma omp parallel default(shared) private(itd, np)
{
    gtd = omp_get_num_threads();  // get total num of threads in a process
    itd = omp_get_thread_num();   // get thread id
    gid = nrank*gtd + itd;        // global id
    printf("Gid %d from thd %d out of %d from process %d out of %d on %s\n", 
        gid, itd, gtd, nrank, numprocs, processor_name);
    if (nrank==0 && itd==0)
    {
        // system("pstree -ap -u $USER");
        system("for f in `cat $PBS_NODEFILE|uniq`; do ssh $f pstree -ap -u 
            $USER; done");
        system("sleep 10");
    }
}
```
Analyzing a hybrid program

[user@mike315 hybrid]$ export OMP_NUM_THREADS=4

[user@mike315 hybrid]$ mpirun -np 2 -x OMP_NUM_THREADS ./a.out

Gid 0 from thread 0 out of 4 from process 0 out of 2 on mike315
Gid 2 from thread 2 out of 4 from process 0 out of 2 on mike315
Gid 1 from thread 1 out of 4 from process 0 out of 2 on mike315
Gid 3 from thread 3 out of 4 from process 0 out of 2 on mike315
Gid 4 from thread 0 out of 4 from process 1 out of 2 on mike315
Gid 6 from thread 2 out of 4 from process 1 out of 2 on mike315
Gid 7 from thread 3 out of 4 from process 1 out of 2 on mike315
Gid 5 from thread 1 out of 4 from process 1 out of 2 on mike315
Using Accelerators

- Nvidia GPU: thousands of cores
  - Cuda, OpenACC
- Intel Xeon Phi: processor(s)+ coprocessor(s)
  - Native, offload to Phi
Pandora – LSU AIX Clusters

◆ Four 8-core IBM Power7 3.3 GHz processors per node, 4 threads per core (128 cores)
◆ 8 nodes total
◆ Advantage: good for multithread jobs, such as applications using OpenMP
◆ Disadvantage: porting efforts from Linux system
# Queue Characteristics – LSU AIX Clusters

“llclass” -- list of queues

<table>
<thead>
<tr>
<th>Machine</th>
<th>Queue</th>
<th>Max Runtime</th>
<th>Max slots per job</th>
<th>Max nodes per job</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pandora</td>
<td>Interactive</td>
<td>30 minutes</td>
<td>8</td>
<td>1</td>
<td>Unpreemptable</td>
</tr>
<tr>
<td></td>
<td>Workq</td>
<td>3 days</td>
<td>224</td>
<td>7</td>
<td>Preemptable</td>
</tr>
<tr>
<td></td>
<td>Single</td>
<td>3 days</td>
<td>64</td>
<td>2</td>
<td>Small jobs</td>
</tr>
</tbody>
</table>
Jobs management - AIX clusters

- Program compilation
  
  xlcc test.c -o test
  mpcc test_mpi.c -o test_mpi

- Job submission

  llsubmit jobscript : submit job llcancel

- Job deletion

  jobid : delete job
Job Monitoring - AIX Clusters

- Command: `showllstatus.py`
  - Show job status and nodes running on
- Command: `llq <options> <job_id>`
  - All jobs are displayed if `<job_id>` is omitted
  - Display detailed information: `llq -l <job_id>`
  - Check the estimated start time: `llq -s <job_id>`
  - Show jobs from a specific user: `llq -u <username>`

```
-bash-3.2$ llq
Id   Owner   Submitted  ST  PRI  Class     Running On
-----------------------------------------------
pandora1.19106.0  mainak   9/1 23:41 R  50  workq  pandora008
pandora1.19108.0  ghoshbd  9/2 14:58 R  50  workq  pandora005
pandora1.19109.0  ghoshbd  9/2 15:08 R  50  workq  pandora007
pandora1.19110.0  ghoshbd  9/2 15:33 R  50  workq  pandora002
pandora1.19111.0  ghoshbd  9/2 15:44 R  50  workq  pandora004
pandora1.19112.0  ghoshbd  9/2 15:58 I  50  workq
pandora1.19113.0  ghoshbd  9/2 16:10 I  50  workq
pandora1.19114.0  mainak   9/4 08:16 I  50  workq
```

8 job step(s) in queue, 3 waiting, 0 pending, 5 running, 0 held, 0 preempted
LoadLeveler Job Script - Serial

#!/bin/sh
@ job_type= serial
@ output = /work/default/username/$(jobid).out
@ error = /work/default/username/$(jobid).err
@ notify_user= youremail@domain
@ notification = error
@ class = single
@ wall_clock_limit= 24:00:00
@ requirements = (Arch == “POWER5”)
@ environment = COPY_ALL
@ queue

<shell commands>
poe <path_to_executable> <options>
<shell commands>
LoadLeveler Job Script - Parallel

#!/bin/sh
#@ job_type= parallel   Job type
#@ output = /work/default/username/$(jobid).out  Standard output
#@ error = /work/default/username/$(jobid).err  Standard error
#@ notify_user= youremail@domain  Notification
#@ notification = error  Notify on error
#@ class = checkpoint  Queue
#@ wall_clock_limit= 24:00:00  Wall clock time
#@ node_usage= shared node usage
#@ node = 2  # of nodes
#@ total_tasks= 16  # of processors
#@ requirements = (Arch == "POWER7")  # Job requirement
#@ environment = COPY_ALL  Environment
#@ queue

<shell commands>
poe<path_to_executable> <options>
<shell commands>
Exercise

- Submit a small job to run “sleep 180” and “print PBS variables”
  - Create a script to submit a 5 min job and print from within the job script PBS variables $PBS_NODEFILE, $PBS_O_WORKDIR. Also run “sleep 180” to give you a few minutes to verify status.
  - Once the job is running, find out the Mother Superior node and other slave nodes assigned to your job using qstat.
  - Log into MS node and verify that your job is running and find your temporary output file
  - Modify your script to print hello from each of your assigned nodes
- Run a shell script using mpirun to print process id of shell
Future Trainings

- Next week training: Basic Shell Scripting
  Wednesdays 9:00am, Sep 23, 2015, Frey 307

- Check out other trainings at HPC webpage: www.hpc.lsu.edu