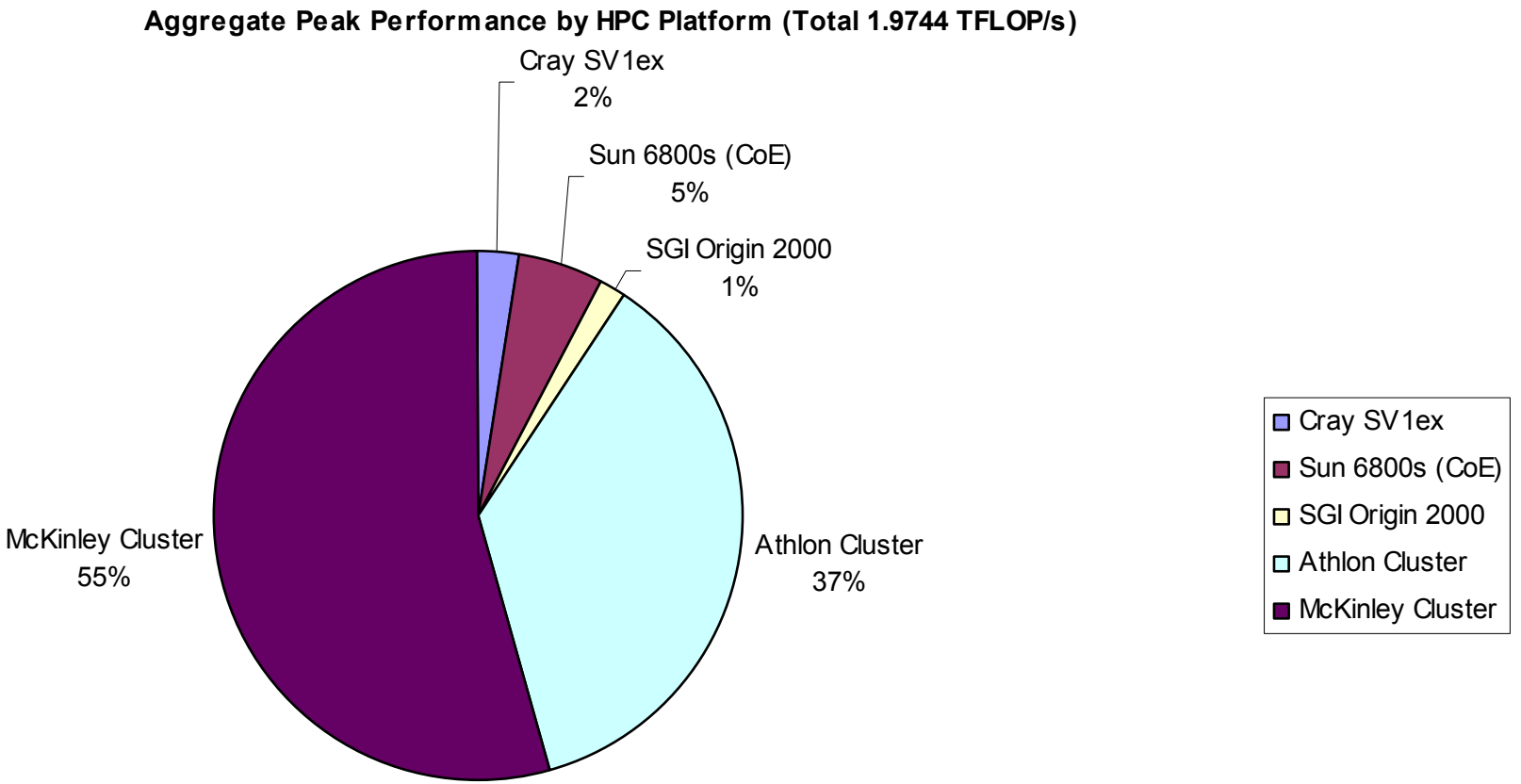
Using the Intel Itanium-2 (McKinley) Cluster at OSC

Science & Technology Support Group
High Performance Computing
Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212

Table of Contents

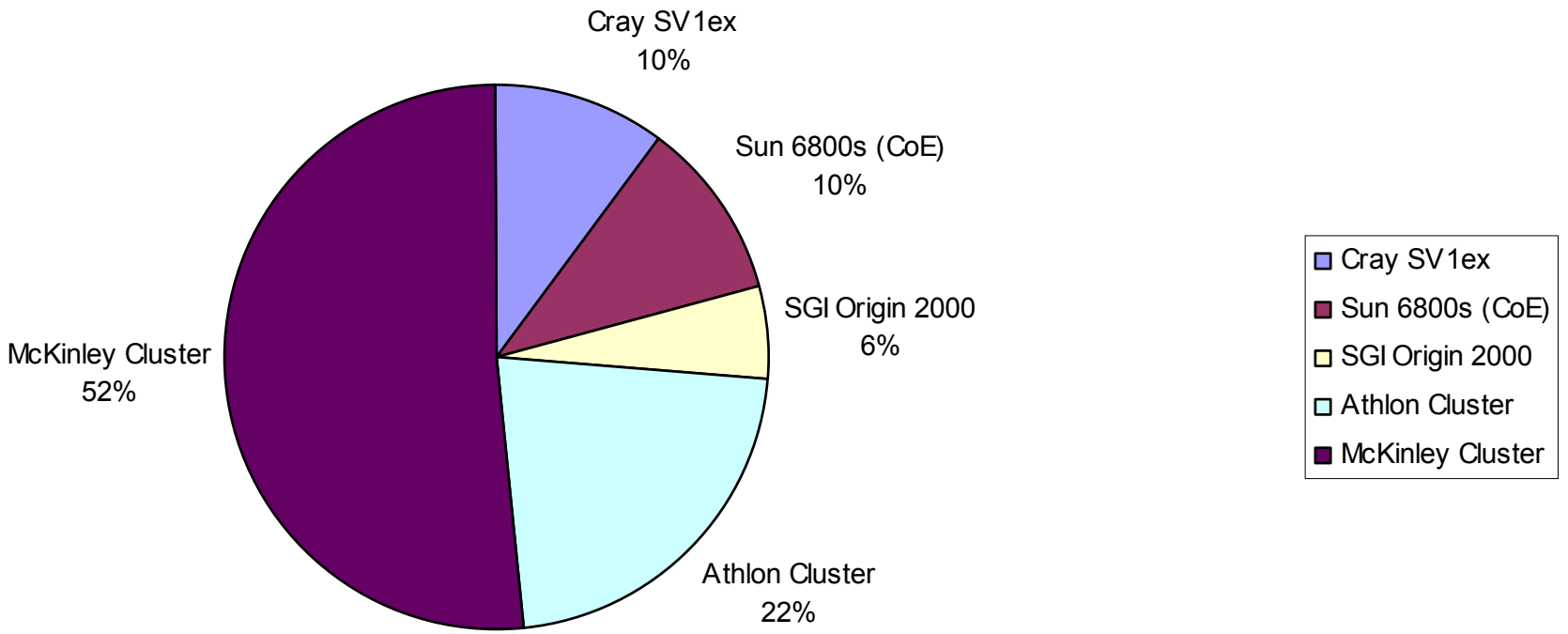
Introduction

- **The OSC Intel Itanium-2 Cluster represents a significant computational resource for the OSC user community.**



Introduction

- **Even when considering the reduced charging rates for clusters, the Itanium-2 cluster represents most of OSC's computing capacity.**



Course Goals

- **Minimize user development time**
  - Review key features of the users environment
  - Current status of development tools (debuggers, profilers, etc.)
  - Outline the scheduling policies and batch system procedures

- **Maximize job performance and throughput**
  - Introduce the new Itanium architecture
  - Discuss optimization techniques
    - Code modification
    - Compiler options
  - Highlight parallel programming techniques
    - OpenMP
    - Message Passing Interface
    - Multi-level programming

Acknowledgments

- **Jim Giuliani, OSC**
  - Originally developed this course on the IA64 cluster
- **Doug Johnson, OSC**
  - Lead system architect
- **Pete Wyckoff, OSC**
  - Lead system architect

Hardware Overview

- Hardware introduction
- Front end node configuration
- Compute node configuration
- Itanium processor architecture
- Processor and memory performance
- System area network
- External network connectivity

Hardware Introduction

At a high level, the OSC Itanium-2 cluster consists of the following:

- **A dual processor front-end node for interactive use, compiling, testing, etc.**
- **128 dual processor compute nodes for concurrent execution of serial and parallel batch jobs**
- **20 dual processor compute nodes with more memory for serial batch jobs**
- **A high-speed system area network (SAN) for inter-node communication.**
- **External network access**
- **Parallel file system for large, high bandwidth scratch space**
  - Approximately 8 TB of disk space!!!
  - Very fast I/O bandwidth

Front End Node Configuration

Starting to focus in on the individual components within the cluster....

Interactive logins are handled by a front-end node:

- Two Intel Itanium-2 processors running at 900 MHz
- 12 GB RAM
- (1) 100 Base-T Ethernet interface
- (2) 1000 Base-T Gigabit Ethernet interfaces
- 73 GB of local SCSI disk (58 GB local `/tmp`)

Parallel Compute Node Configuration

Total of 128 compute nodes

Each has:

- Two Intel Itanium-2 processors running at 900 MHz
- 4 GB RAM
- (1) Myrinet 2000 SAN interface
- (1) 100 Base-T Ethernet interface
- (1) 1000 Base-T Gigabit Ethernet interface
- 73 GB of local SCSI disk (~57 GB local `/tmp`).

Serial Compute Node Configuration

Total of 20 compute nodes

Each has:

- **Two Intel Itanium-2 processors running at 900 MHz**
- **12 GB RAM**
- **(1) 100 Base-T Ethernet interface**
- **(1) 1000 Base-T Gigabit Ethernet interface**
- **73 GB of UW SCSI disk (~57 GB local `/tmp`).**

Itanium-2 Processor Architecture

- **Itanium-2 is Intel's second generation 64-bit architecture, based on the Explicitly Parallel Instruction Computing (EPIC) design philosophy**

- **To achieve improved performance, Itanium architecture code accomplishes the following:**
  - Increases instruction level parallelism (ILP)
  - Improves branch handling
  - Hides memory latencies
  - Supports modular code

- **The EPIC architecture is dependent on the compilers to write code to achieve the aforementioned performance improvements**

Itanium Processor Architecture

- **The Intel Itanium-2 architecture supports the following data types:**
  - Integer: 1, 2, 4 and 8 byte(s)
  - Floating-point single, double and double-extended formats
  - Pointers: 8 bytes
- **The basic data type in Itanium architecture is 8 bytes.**
- **Apart from a few exceptions, all integer operations are on 64-bit data**
- **Registers are always written as 64 bits.**
- **3 levels of cache**

All of the nodes in the OSC cluster use the Intel Itanium-2 processor with a 900 MHz clock

Processor Performance

11 Total Execution Units

- 4 integer units
- 2 load/store units
- 2 floating-point units
- 3 branch predict units

Fully pipelined

- 10-stage pipeline

Peak Floating Point Performance

- 7.2 GFLOPs per processor, single precision
- 3.6 GFLOPs per processor, double precision

Actual Performance Seen From Linpack Benchmark

- 761.98 GFLOPs on Parallel Linpack at 256 processors (~3.0 GFLOPs/processor)

Memory Hierarchy

Cache

- L1 instruction cache: 16 kB, 4-way set assoc., 32 byte line
- L1 data cache: 16 kB on-die, 4-way set assoc., 32 byte line
  Floating point loads bypass the L1 data cache
- L2 unified cache: 256 kB on-die, 6-way set assoc., 64 byte line
- L3 unified L3 cache: 1.5 MB off-die, but dedicated bus to processor

Main Memory System

- Front Side Bus runs at 266 MHz
- multi-channel PC2100 DDR-SDRAM memory

Memory Performance

- 256-bit wide data path, 200MHz memory bus clock
- 2.5 ns latency
- 6.4 GB/s peak, 3.0 GB/s on stream_d memory copy, 3.7 GB/s on stream_d triad.

System Area Network

Each node has one Myrinet SAN interfaces:

- **Switched 2.0 Gbps bidirectional network.**
- **Private to the cluster -- no outside traffic.**
- **MPI runs over Myrinet using Myricom's GM message passing library -- no overhead from TCP/IP stack.**

External Network Connectivity

- All nodes mount the file-system containing the users' home directories from the OSC Mass Storage System:

  `mss.osc.edu`

- Interactive logins using the `ssh` protocol from anywhere on the Internet are handled by the front end node

  `ia64.osc.edu`

- The `ssh` protocol is preferred because it does not send clear-text passwords and uses encryption

- Documentation can be found on the OSC Technical Information Web Server, `http://oscinfo.osc.edu/`

The Linux Operating System

- Linux features
- Processes and threads in Linux

Features of Linux

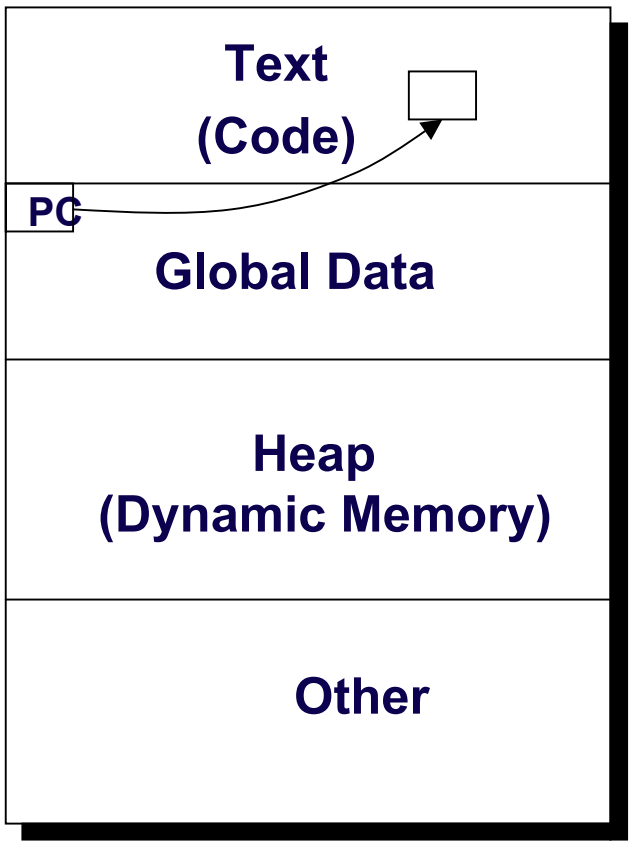
- Freely distributable with full source code.
- Runs on a variety of platforms (Intel IA32 and IA64, DEC Alpha, MIPS, Sun SPARC, several embedded processors).
- Multi-threaded, fully preemptive multitasking.
- Implements most of the POSIX and Open Group Single UNIX system APIs.
- Protocol and source compatibility with most other UNIX-like operating systems.

Processes, Threads and Load Sharing in Linux

- The basic block of scheduling in UNIX has historically been the *process.*
- Recent UNIXes have also added the concept of multiple *threads* of execution within a single process.
- Linux supports both processes and threads.
- Linux's internal scheduler will also try to load-balance running processes and threads, so that they will be given full use of a processor so long as there are as many or fewer active processes/threads as there are processors.

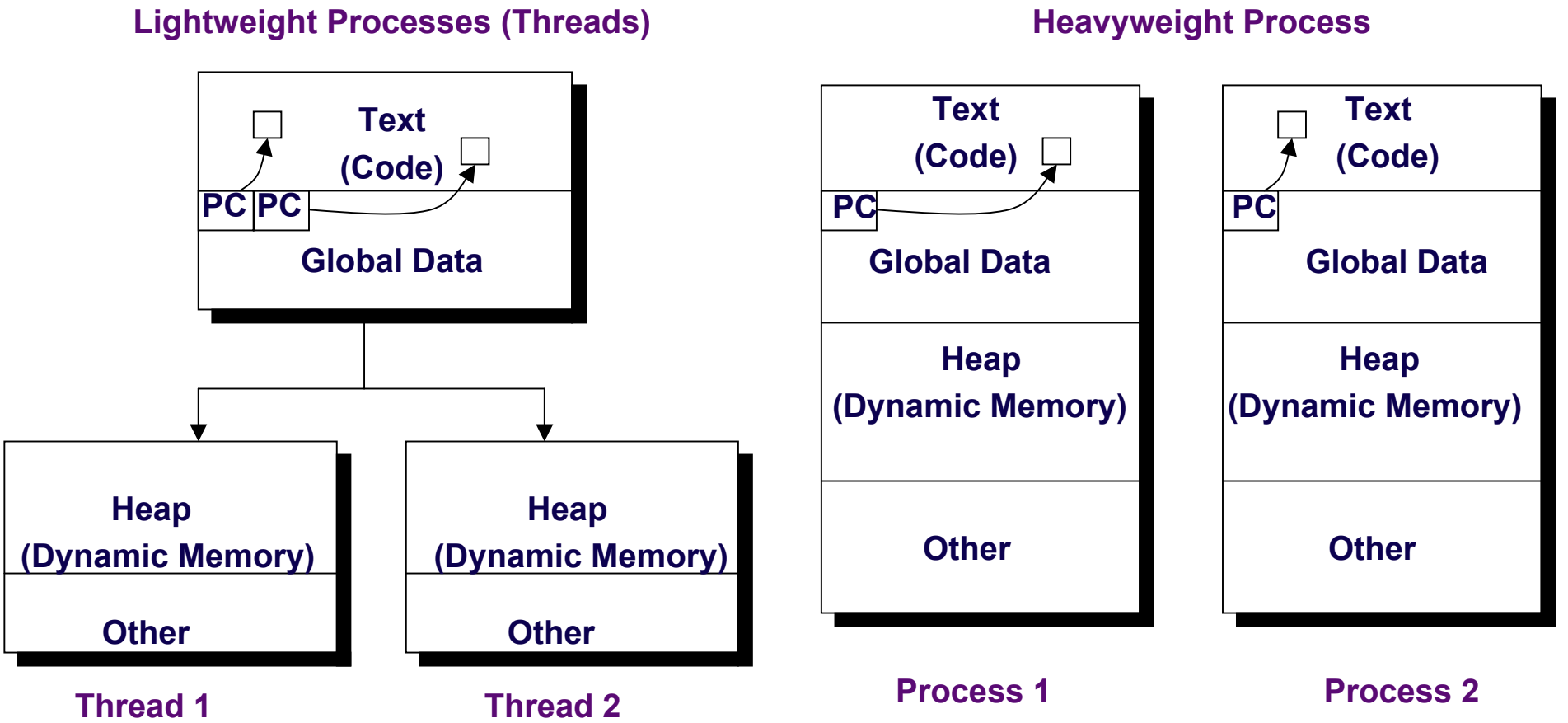
Process - Definition

A Process *Image*



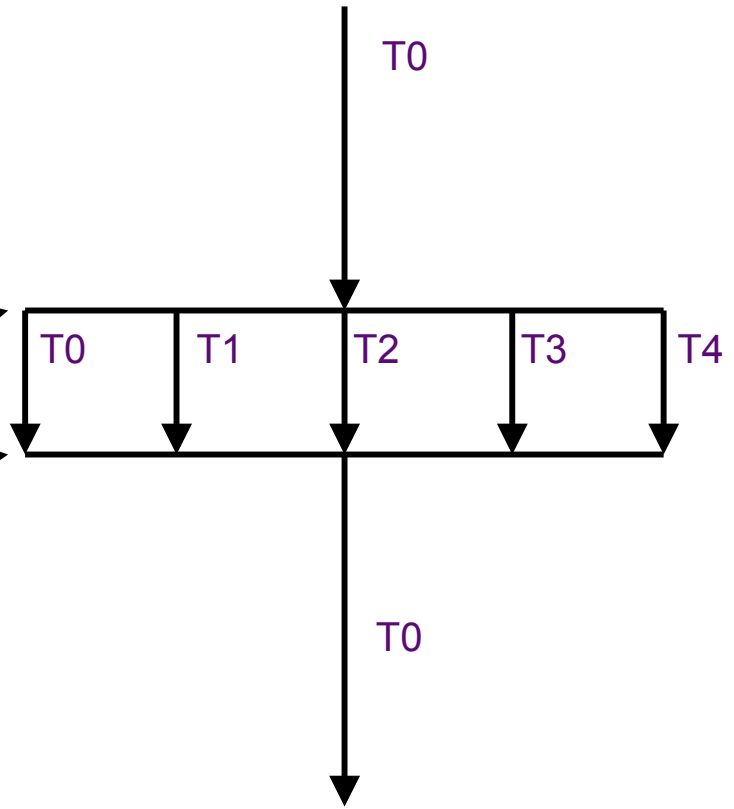
- **A process is a running program.**
- **Elements of a process:**
  - Memory (*text*, *data*)
  - Register contents
  - *Program Counter* (PC)
  - Process status
- **Each process has a unique *process id*.**
- **Keep concepts of *process* and *processor* separate.**

Types of Processes

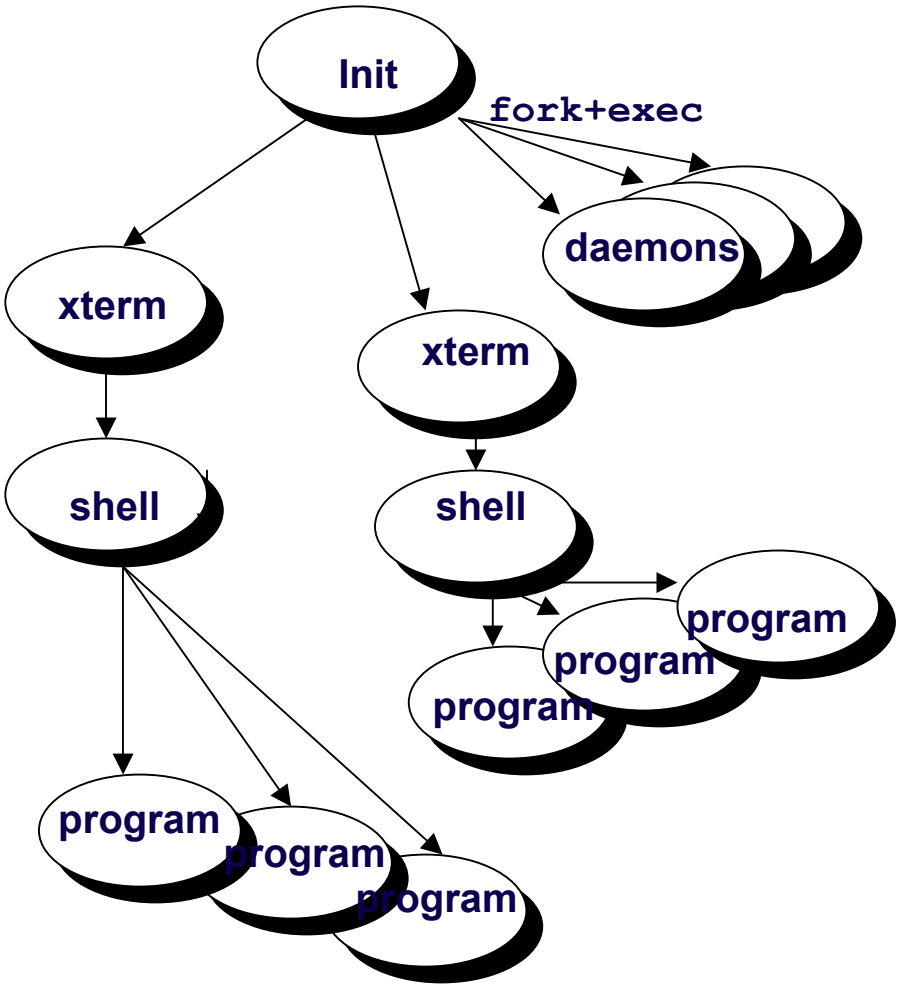


Lightweight Process - An Example

```
#
# Pseudocode for lightweight thread example
#

static data(10000)

InitializeData

SpawnThreads
JoinThreads

Output

Done
```



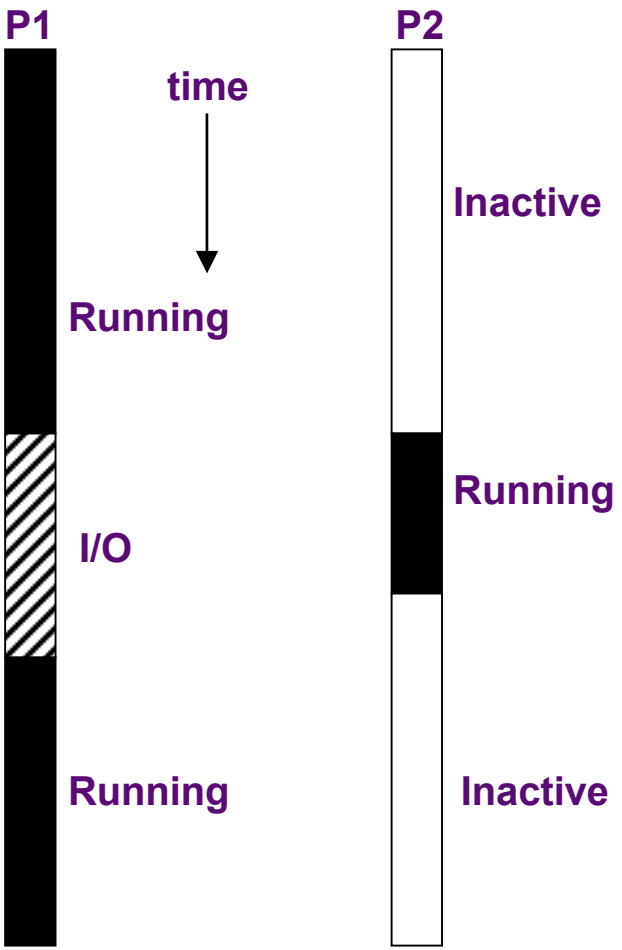
Process Creation



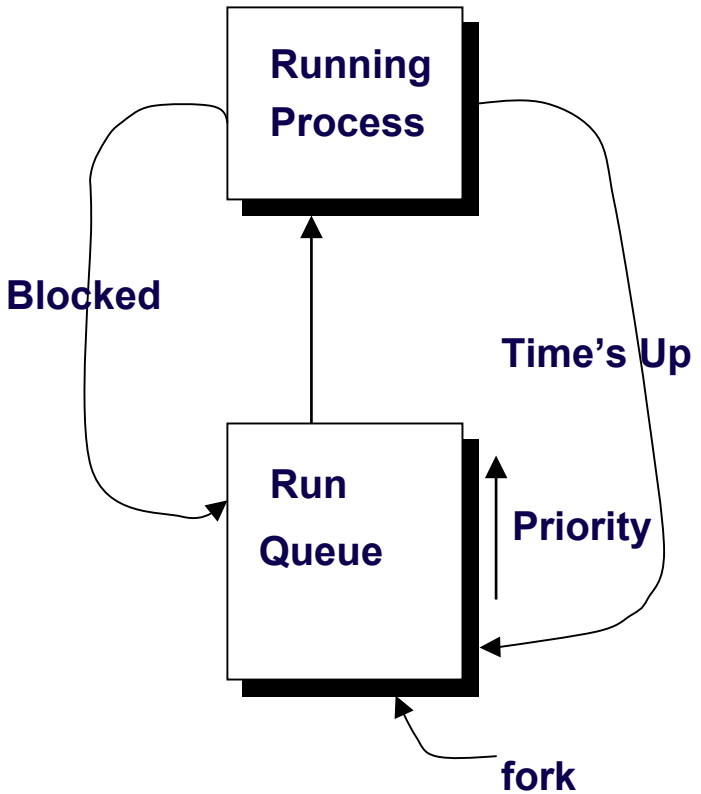
- Processes exist in a hierarchy
- The `init` process is at the top of the process hierarchy. (has process id (*pid*) of 1.
- All processes other than init are created by a *parent* process and are considered *child* processes.
- Heavyweight processes are created by a call to `fork(2)`. (Typically involves call to `exec(2)`)
- Lightweight processes are created by a call to `clone(2)` and are form a *process group*.
- Lightweight processes have the same process name.

Process States

- Processes can be in one of several states:
  - S Sleeping (blocked), waiting for a resource.
  - R Running (actually doing work).
  - Z Terminated and parent not waiting
  - T Stopped.
  - I In intermediate state of creation.
  - X Waiting for memory.
- Sometimes processes *spin wait* or *busy wait*. They eat CPU without doing anything useful.
- Processes can be *switched out* to allow a higher-priority process to run, or while waiting for something to happen like I/O.



Load Sharing - Time Slicing



- The front end node of the cluster is a shared resource.
- Any number of people can be using the system at any given time.
- Processes are scheduled for efficient and equitable use of CPU resources.
- The scheduler (part of the OS kernel) handles the running of processes using *run queues* and process *priorities*.
- No scheduler is perfect.
- All processes can run only for a specified *time slice* before giving up control to another process. (30 millisec default)

Priority and `niceness`

- Every process has its own priority
- Priority is simply a number between 0 and 254. The higher the number, the lower the priority.
- As a process runs, its priority gets worse (i.e., the number gets larger). Priority is periodically updated by the kernel.
- Processes also have a "niceness" associated with them. It is represented by a number between -20 and 19. (0 is the default).
- By increasing the niceness value for a process, the priority of the process is effectively made worse. Syntax follows:

  `/bin/nice -`*`increment`* `<`*`pid`*`>`

- Can only increase niceness unless you are superuser
- May want to use if you don't need quick turnaround.

User Environment Management

- Accessing the cluster
- Modules
- Text editing
- System status
- File management
- 3rd party applications

Accessing the Cluster

- There is only one way to remotely access the front end node of the cluster (`ia64.osc.edu`):

```
ssh userid@ia64.osc.edu
```

- `ssh` sends your commands over an encrypted stream, so your passwords and so forth can't be sniffed over the network

- Batch nodes are not connected to the external network

Remote X Display from the Cluster

- **You can run applications which use the X Window System on the front end node and have them displayed on your remote workstation or PC (PC requires special software).**
- **If you use `ssh` from a UNIX workstation, you should be able to display X applications remotely with no further work**
- **If `ssh` does not automatically forward your display, try invoking it with the -X option:**

```
ssh -X userid@ia64.osc.edu
```

- **If that doesn't work or if you are connecting from a PC running Windows, you'll need to set an environment variable called DISPLAY in your session on the front end node to point to your workstation:**

```
export DISPLAY="mypc.some.edu:0.0"
```
(for ksh users)

```
setenv DISPLAY mypc.some.edu:0.0
```
(for csh users)

- **You'll also need to tell your UNIX workstation that the front end node is allowed to display to it:**

```
xhost +ia64.osc.edu
```

More on X Display from the Cluster

- **While you can run virtually any X client program on the front end node displayed to your remote workstation, the OSC systems staff would prefer that you use this only for programs which can't be run any other way.**

- **In particular, running remotely displayed `xterm` or `rxvt` sessions chews up lots of I/O bandwidth and doesn't really doesn't gain you anything over `ssh`.**

- **Remote X display in interactive batch jobs (something we'll discuss later with respect to debugging MPI programs) is only supported for `ssh` sessions.**

Modules

- The “modules” interface is a way to allow multiple versions of software to coexist.
- They allow you to add or remove software from your environment without having to manually modify environment variables.
- This is a “Cray-ism” which OSC has adopted for all of our HPC systems.

Using modules

- **You can get a list of modules you currently have loaded by running `module list`:**

```
[mck-login1]$ module list
Currently Loaded Modulefiles:
     1) mpich_gm
     2) totalview
     3) modules_0_2
     4) pbs_2_3_12
```

- **To get a list of all available modules, run `module avail`:**

```
[mck-login1]$ module avail
---------- /usr/local/lanl-modules-0.2/modules --------
   hdf -> hdf_4_1_2
   pbs -> pbs_2_3_12
...list continues...
```

OSC
Innovations in computing, networking, and education

Using Modules (con't)

- **To add a software module to your environment, run `module load <modulename>`:**

```
[mck-login1]$ /home/jimg> module load scms
[mck-login1]$ /home/jimg> which scms
/usr/local/scms/bin/scms
[mck-login1]$ /home/jimg> module list
Currently Loaded Modulefiles:
          …scms…
```

- **To remove a software package from your environment, run `module unload <modulename>`:**

```
[mck-login1]$ /home/jimg> module unload scms
[mck-login1]$ /home/jimg> which scms
scms: Command not found.
[mck-login1]$ /home/jimg> module list
Currently Loaded Modulefiles:
          …no scms…
```

Modules and the UNIX Shell

- **Modules work by modifying environment variables like $PATH and $MANPATH within your shell**

- **Because of this, you should NOT explicitly set $PATH in your `.profile` or `.cshrc`; instead, you should append directories to the existing $PATH:**

  `setenv PATH $HOME/bin:$PATH` (for csh users)

  `export PATH=$HOME/bin:$PATH` (for ksh users)

- **Also, if you use a mixture of `csh` and `ksh` (for instance, you use `csh` interactively but write batch scripts in `ksh`), you should add the following to your `.profile` and `.cshrc`:**

```
# .profile modules init
. $MODULESHOME/init/ksh
# .cshrc modules init
source $MODULESHOME/init/csh
```

Text Editing

- As with virtually all Unix systems, the front end node has the `vi` editor installed:

```
[mck-login1]$ which vi
/bin/vi
```

- The popular `emacs` editor is also available, as well as `jed` (an `emacs`-like editor which uses much less memory):

```
[mck-login1]$ which emacs
/usr/bin/emacs

[mck-login1]$ which jed
/usr/bin/jed
```

System Status

- **Linux supplies a number of tools for examining what the front end node is running:**

```
uptime
w
ps
top
```

- **In addition, there are commands for examining the PBS batch queue state on the rest of the cluster and the OSC accounting information:**

```
qstat
```
(more on this later)

```
OSCusage
```

uptime and w

- The `uptime` command prints out how long the system has been up, along with the number of users currently logged in and the load average (the number of processes/threads actively running) for the last minute, five minutes, and fifteen minutes:

```
[mck-login1]$ uptime
 10:00am  up 8 days, 18:37, 15 users,  load average: 0.05, 0.08, 0.08
```

- The `w` command gives the same information as `uptime`, but also lists all the users currently logged on the system:

```
[mck-login1]$ w
 10:02am  up 8 days, 18:38, 15 users,  load average: 0.01, 0.06, 0.07
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
djohnson pts/0    neptune.osc.edu   9Oct01  5days  0.37s  0.37s  -tcsh
pw       pts/2    quasar.osc.edu   Mon 4pm 16:11m  0.90s  0.71s  bash
osu2782  pts/6    pollux.mps.ohio-  9Oct01  8:04  12.01s  3.25s  -bin/csh
troy     pts/8    dhcp065-024-120- Wed 9pm 10:29m  0.24s  0.24s  -tcsh
jimg     pts/9    gemini.osc.edu   Tue 8am  0.00s  0.26s  0.03s  w
djohnson pts/10   neptune.osc.edu  10Oct01  5days  0.47s  0.25s  bash
…[list truncated]...
```

Monitoring Processes with `ps` and `top`

`ps aux`

```
 USER        PID %CPU %MEM   SIZE    RSS TTY STAT START   TIME COMMAND
root          1  0.0  0.0   2848   288 ?        S    Oct09   0:05 init
pw        28063  0.0  0.0   5728  1264 pts/19   S    Oct16   0:00 -bash
osu2782   10287  0.0  0.2  20688 11136 pts/6    S    Oct09   0:08 emacs Makefile
bin        1253  0.0  0.0   3696   704 ?        S    Oct09   1:39 portmap
```

- Use `ps` to see the current state of a process.
- `ps aux` will show the long listing for every process on the system.
- Use `top` to see CPU usage for processes.
- `kill` and `killall` send signals to processes:

```
kill -KILL <pid>
killall -KILL <progname>
```

Sample `top` output

```
 10:13am  up 8 days, 18:49, 16 users,  load average: 0.34, 0.10, 0.08
110 processes: 107 sleeping, 2 running, 0 zombie, 1 stopped
CPU0 states: 31.37% user,  0.39% system,  0.0% nice, 67.27% idle
CPU1 states: 68.26% user,  0.31% system,  0.0% nice, 30.46% idle
Mem:  4054624K av, 4030144K used,   24480K free,       0K shrd, 1509584K buff
Swap: 10176800K av,   62128K used, 10114672K free                 1665216K
   cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
31888 jimg      16   0 77776  75M 13072 R    99.8  0.4   0:25 f90com
31889 jimg      14   0  2896 2896  2208 R     1.2  0.0   0:00 top
10287 osu2782   12   0 11920  10M  5968 S     0.2  0.0   0:09 emacs
    1 root      12   0   416  288   288 S     0.0  0.0   0:05 init
    2 root      12   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root      20  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    4 root      20  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU1
    5 root      12   0     0    0     0 SW    0.0  0.0   0:16 kswapd
    6 root      12   0     0    0     0 SW    0.0  0.0   0:00 kreclaimd
    7 root      12   0     0    0     0 SW    0.0  0.0   0:03 bdflush
    8 root      12   0     0    0     0 SW    0.0  0.0   0:30 kupdated
```

OSC
Innovations in computing, networking, and education

OSCusage

- `OSCusage` is an interface to OSC's local accounting database.
- It lets you see the RU usage for your project on a specified date or range of dates.

```
[mck-login1]$ ./OSCusage -v 10/08/2001 10/09/2001
                    Usage Statistics for project PZS0150
                         for 10/08/2001 to 10/09/2001
                           RU Balance: -1784.59949

                Start                  RUs      Charge            Status
Username  Date  Date & Time            Used        Type     Queue            Job
    jimg 10/09                      0.03755 CPU-BEOWUL    serial           test
    jimg 10/09                      0.32939 CPU-BEOWUL    serial           test
[.....]
    jimg 10/09  10/08 10:14:31      0.00000 CPU-Origin
    jimg 10/09  10/08 10:13:32      0.00272 CPU-Origin
    jimg 10/09                      0.00000    CPU-T94                RESIDUAL
------------       jimg TOTAL     0.38346
 ============    PZS0150 TOTAL     0.38346
```

OSC
Innovations in computing, networking, and education

OSCusage (con't)

- **By default, you'll see output for everyone on your project on the previous day. To see only your own statistics, use the `-q` option:**

```
[mck-login1]$ ./OSCusage -q

                    Usage Statistics for project PZS0150
                         for 10/18/2001 to 10/18/2001

                              RU Balance: -1784.59949

Username  Dates                                RUs  Status

    jimg  10/18/2001 to 10/18/2001         0.00378  ACTIVE
 ============   PZS0150 TOTAL               0.00378
```

`OSCusage` (con't.)

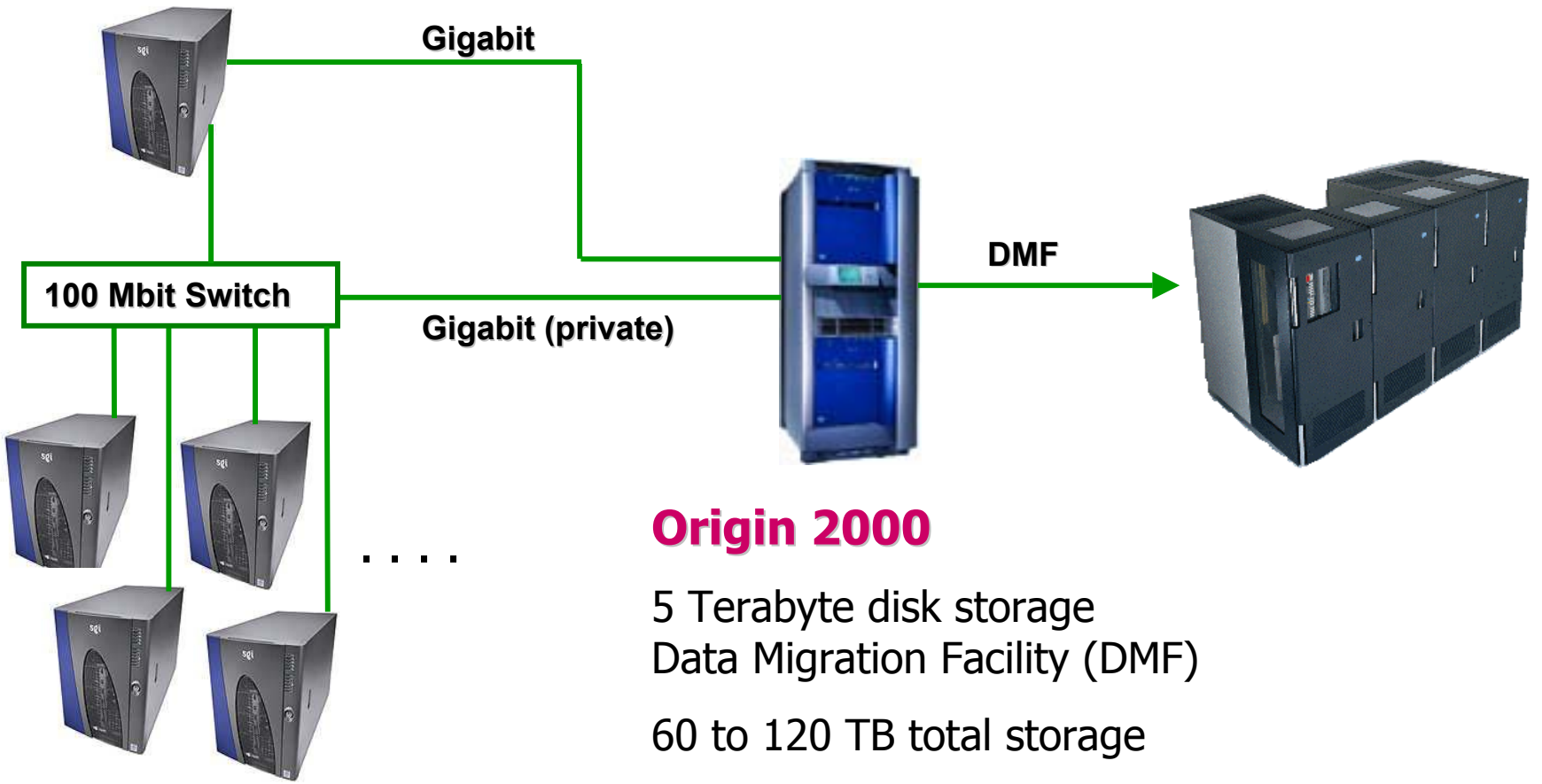
- To see a date or range of dates, specify the start and end dates in MM/DD/YYYY form; the ending date is needed only if you want more than one day.

- The `-v` (verbose) flag will give you more details on how much was charged for CPU usage on each of OSC's machines as well as for disk usage on the mass storage server.

File Management

- **File management on OSC's cluster is largely automatic -- the mass storage server automatically takes care of moving files between disk and tape.**
- **However, since you do get charged a small amount for the total amount of storage you're using, you may want to compress large unused files using either the `compress` or `gzip` commands. `gzip` tends to do a better compression job, but it also isn't yet universally available.**
- **ftp**
  - ftp is available for transferring files from your workstation to the mass storage server (`mss.osc.edu`)
- **scp**
  - scp is available for transferring files to the mass storage server as well as the the front end machine

Mass Storage Support



Origin 2000

5 Terabyte disk storage
Data Migration Facility (DMF)

60 to 120 TB total storage

3rd Party Applications

Which have been validated for use on the Itanium-2:

- **Bioinfomatics**
  - NCBI Toolbox, including BLAST
- **Computational Chemistry**
  - Gaussian 98
  - MPQC
  - NWChem
- **Structural Analysis**
  - LS-Dyna3D
- **Miscellaneous**
  - Gnuplot

Program Development Tools and Libraries

- GNU Compilers
- Intel Compilers
- MPI Compiler Wrappers
- Libraries
- Debuggers
- Performance analysis tools

GNU Compilers

Virtually every Linux system includes the GNU compiler suite from the Free Software Foundation. This a freely available open source compiler system including support for:

- C (gcc)
- C++ (g++)
- Fortran 77 (g77)

While these are quite good compilers in terms of standards conformance, they do not generate as fast code as other compilers and lack support for parallelization.

GNU Compilers: Common Options

`-c`	(compile only; do not link)
`-DMACRO[=value]`	(defines preprocessor macro `MACRO` with optional `value`; default value is 1)
`-g`	(generate symbols for debugging; disables optimization)
`-I/dir/name`	(add `/dir/name` to the list of directories to be searched for `#include`d files)
`-lname`	(add library `libname.{a|so}` to the list of libraries to be linked -- order is important!)
`-L/dir/name`	(add `/dir/name` to the list of directories to be searched for library files)
`-o outfile`	(name resulting output file `outfile`; default is `a.out`)
`-UMACRO`	(removes definition of `MACRO` from preprocessor)

GNU Compilers: Common Options (con't.)

`-O0` (no optimization; default)

`-O1` (light optimization)

`-O2` (moderate optimization)

`-O3` (heavy optimization; may cause slight numerical differences)

`-fexpensive-optimizations` (enables minor but expensive optimizations)

`-finline-functions` (enables function inlining)

`-fschedule-insns` (enables instruction scheduling and reordering)

`-funroll-loops` (enables loop unrolling optimizations)

GNU Compilers: C/C++ Options

- `-ansi` (enforces ANSI C/C++ compliance; default; opposite of `-traditional`)
- `-pedantic` (increases strictness of language compliance)
- `-traditional` (enforces K&Rv1 C or pre-ANSI C++ compliance; opposite of `-ansi`)
- `-Wall` (enables all common warnings)

Recommened flags: `-O2 -funroll-loops -Wall -ansi -pedantic`

GNU Compilers: F77 Options

`-ffree-form`	(allows Fortran 90 style free form source)
`-ff90`	(allows some Fortran 90 constructs)
`-finit-local-zero`	(initializes all local variables to zero)
`-malign-double`	(causes word alignment of `DOUBLE PRECISION` variables)
`-pedantic`	(issues warnings on non-standard code)
`-Wall`	(enables all common warnings)
`-Wsurprising`	(issues warnings on code which may be interpreted differently on different systems)
`-fno-underscoring`	(Disables appending an underscore to external subroutine names)

Recommended flags: `-O2 -funroll-loops -malign-double -Wall -pedantic`

OSC
Innovations in computing, networking, and education

Intel Compilers

Because of the performance of code generated by the GNU compilers, the OSC cluster also has the Intel Linux compilers installed. The Intel compilers include support for:

- C and C++ (`ecc`)
- Fortran 90 (`efc`)

The Intel compiler suite also includes a debugger, which is currently not ready for production use. In the interim, the gnu debugger is recommended. Complete manuals can be found on the Web at http://oscinfo.osc.edu/manuals/.

Intel Compilers: Common Options

`-c`	(compile only; do not link)
`-DMACRO[=value]`	(defines preprocessor macro `MACRO` with optional `value`; default value is 1)
`-g`	(generate symbols for debugging; disables optimization)
`-I/dir/name`	(add `/dir/name` to the list of directories to be searched for `#include`d files)
`-lname`	(add library `libname.{a|so}` to the list of libraries to be linked)
`-L/dir/name`	(add `/dir/name` to the list of directories to be searched for library files)
`-o outfile`	(name resulting output file `outfile`; default is `a.out`)
`-UMACRO`	(removes definition of `MACRO` from preprocessor)

Intel C and C++ Compiler Options

Command Line Syntax

```
ecc [options] file1 [file2 ...] [linker options]
```

- **-O, -O1 and -O2** (default)
  - No difference
  - constant propagation, copy propagation, dead-code elimination
  - global register allocation, instruction scheduling
  - register variable detection, common subexpression elimination
  - variable renaming, strength reduction-induction variable
  - tail recursion elimination and software pipelining
- **-O3**
  - Enables -O2 option with more aggressive optimizations
  - prefetching
  - scalar replacement
  - loop transformations

Intel C and C++ Compiler Options

`-Ze`	Conform to the ANSI/ISO standard. Default is to accept extensions to the ANSI standard.
`-mp`	Restricts some optimizations to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI and IEEE standards
`-ip | -ipo`	Interprocedural Optimizations
`-prof_gen`	Compiler produces instrumented code which will write out performance profile information during execution. Profile data written to a unique dynamic information file.
`-prof_use`	Produces a profile-optimized executable and merges available dynamic information files.
`-unroll[n]`	Unroll loops [n] times. Only done on loops that the compiler thinks should be unrolled. If you omit [n], the compiler will determine [n].

Intel F90 Compiler

- **Command Line Syntax**

```
efc [options] file1.f [file2.f ...] [linker options]
```

- **Environment Variables**
  - LIB: specifies the diretory path for the math libraries
  - INCLUDE: specifies the directory path for the include files
  - TMP: specifies the directory in which to store temporary files

- **Specifying Executable Files**
  - use the `-ofile` option to specify an alternate name

- **There is currently a problem with the Intel compilers and linking against shared libraries**

Intel F90 Compiler: Preprocessor

- **The compiler preprocesses files as an optional first phase of the compilation, and can be invoked separately**

  `fpp`

- **You can enable preprocessor for any Frotran file by specifying the `-fpp` option to the compiler**
  - fpp0: disable preprocessing
  - fpp1: enable CVF conditional compilations and `#directives` (default)
  - fpp2: enable only `#directives`
  - fpp3: enable only CVF conditional compilation directives
- **`-openmp`, which we will learn more about later, automatically invokes the preprocessor**
- **`-Dname[=value({#|text})]`**
  - `-D` defines the assertion and macro names
  - `-U` suppresses a definition

Intel F90 Compiler: General Options

- **Listing Options**
  - `-G0` writes a listing of the source file to `stdout`
  - `-G1` writes a listing of the source file to `stdout` without INCLUDE files expanded
- **Debugging**
  - The compiler lets you generate code to support symbolic debugging with optimizations
  - Debugging information returned may be inaccurate as a side-effect of optimization

```
-g -O[0|1|2] -fp-
```

-g : no optimization and epb register used as the frame pointer for debugging

-g -O2 -fp- : Level 2 optimizations and epb register used as the frame pointer for debugging

Intel F90 Compiler: General Options

`-ftz`	Flushes denormal results to zero (recommended)
`-r8`	Treat all variables, constants, functions and intrinsics as 64 bit value
`-r16`	Treat all variables, constants, functions and intrinsics as 128 bit values
`-lowercase`	maps external names to lowercase alphabetic characters
`-uppercase`	maps external names to uppercase
`-nus`	Disables appending an underscore to external subroutine names

Intel F90 Compiler: Optimizations

- **-O or -O1**
  - constant propagation, copy propagation, dead-code elimination
  - global register allocation
  - global instruction scheduling and control speculation
  - optimized code selection, partial redundancy elimination
  - strength reduction/induction variable simplification
  - variable renaming, predication, software pipelining
- **-O2** (default)
  - Same as -O1 but with loop unrolling and inlines intrinsics
- **-O3**
  - -O2 with prefetching, scalar replacement and loop transformation
- **-mp**
  - Restricts optimization that cause some minor loss or gain of precision in floating-point arithmetic to maintain a declared level of precision and to ensure that floating-point arithmetic more nearly conforms to the ANSI and IEEE standards.

Intel F90 Compiler: Interprocedural Optimizations

- `-ip` : inline function expansion for calls to procedures defined within the current source file
- `-ipo` : inline function expansion for calls to procedures defined in separate files

Strongly recommended
2x speedup on most Fortran codes

- As one command:

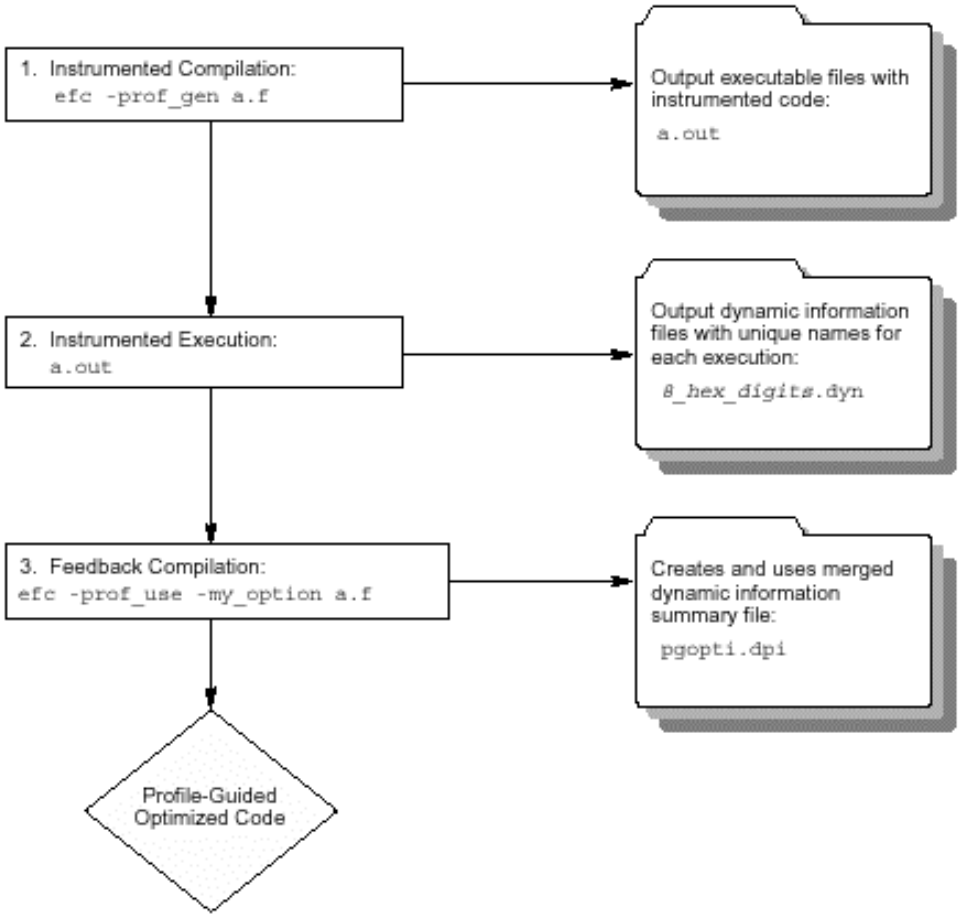
```
efc -ipo -o executable file1.f90 file2.f90 file3.f90
```

- As separate commands:

```
efc -ipo -c file1.f90 file2.f90 file3.f90
efc -ipo -o executable file1.o file2.o file3.o
```

Profile-guided Optimizations

- Profile guided optimizations tell the compiler which areas of an application are most frequently executed
- Knowing these areas, the compiler is able to be more selective and specific in optimizing the application
- `-prof_gen`
- `-prof_use`
- Profile-guided optimizations disable `-ip` and `-ipo`



Profile Guided Optimizations: Example

```
[mck-login1]$ efc -O3 -w dp4.f -o dp4
[mck-login1]$ time ./dp4
RESP=   1.741    0.049 ERRSQ=0.43E-04 ITERAT= 1733  NITER=  1
RESP=   1.528    0.127 ERRSQ=0.84E-03 ITERAT= 1038  NITER=  2
RESP=   1.522    0.117 ERRSQ=0.86E-05 ITERAT=  817  NITER=  3
RESP=   1.522    0.116 ERRSQ=0.63E-05 ITERAT=  758  NITER=  4
real    436.7s
user    436.27s
sys     0.1s
[mck-login1]$ efc -O3 -prof_gen -w dp4.f -o dp4
[mck-login1]$ ./dp4
[mck-login1]$ efc -O3 -prof_use -w dp4.f -o dp4
/usr/local/intel/compiler60/ia64/bin/profmerge: merging dynamic file:   3bcf359e.dyn
[mck-login1]$ time ./dp4
RESP=   1.741    0.049 ERRSQ=0.43E-04 ITERAT= 1732  NITER=  1
RESP=   1.528    0.127 ERRSQ=0.84E-03 ITERAT= 1038  NITER=  2
RESP=   1.522    0.117 ERRSQ=0.86E-05 ITERAT=  818  NITER=  3
RESP=   1.522    0.116 ERRSQ=0.63E-05 ITERAT=  759  NITER=  4
real   416.35s
ser    416.01s
sys    0.01s
```

5% speedup on this numerical code, up to 2x on others

OSC
Innovations in computing, networking, and education

MPI Compiler Wrappers

The MPICH/GM implementation of MPI uses a set of compiler scripts to keep users from having to remember how to set include and library paths for the their MPI compiles. These scripts call the system compilers to do the actual compilation. The scripts support the following languages:

- C and C++ (`mpicc` -- wrapper for `ecc`)
- Fortran 90 (`mpif90` -- wrapper for `efc`)

These compiler scripts accept the same arguments as the compiler they wrap, i.e. `mpicc` accepts the same arguments as `ecc`, `mpif90` accepts the same arguments as `efc`, etc.

MPI Compiler Wrappers (con't.)

The MPI compiler wrappers also accept a few command line arguments of their own:

- `-mpilog` (generates MPE log files compatible with the `jumpshot` MPI profiler)
- `-mpitrace` (prints trace messages on entry and exit to all MPI routines)

Libraries

The OSC cluster has several numerical libraries installed which provide:

- Increased performance with highly optimized or hand tuned routines
- Increased functionality with a wide array of mathematical routines

Libraries available are:

- **Intel Math Kernel Library (MKL)**
  http://oscinfo.osc.edu/software/mkl/Index.htm
- **HP Vector Math Library**
  http://www.hp.com/go/mlib
- **GNU Scientific Library (GSL)**
  http://sources.redhat.com/gsl
- **ATLAS Library (*not yet available*)**
  http://math-atlas.sourceforge.net
- **NAG Mathematics Libraries (*not yet available*)**
  http://www.nag.com

Intel Math Kernel Library

Current Status:

- Works with Intel compilers

```
module load mkl
efc -O2 -ftz lapack1.f90 $MKL
```

Features

- BLAS Levels 1-3
  - Vectors and matrix operations
- LAPACK
  - Routines for solving dense linear algebra problems
- FFTs
  - mixed-radix FFTs, convolutions and correlations

HP Vector Math Library

Current Status:

- Works with Intel compilers

```
module load mlib
efc -O2 -ftz lapack1.f90 $MLIB
```

Features

- BLAS Levels 1-3
  - Vectors and matrix operations
- LAPACK
  - Routines for solving dense linear algebra problems
  - Better performance than MKL in many cases
- FFTs
  - mixed-radix FFTs, convolutions and correlations

GNU Scientific Library (GSL)

Current Status:

- Works with GNU and Intel compilers

```
gcc -O2 blas1.c -o blas1 -lgsl -lgslcblas
g77 -O2 -fno-underscoring blas1.f -o blas1 -lgsl -lgslcblas
efc -O2 -nus blas1.f -o blas1 -lgsl -lgslcblas
```

- For g77 codes you will need to add the `-fno-underscoring` flag
- Not as optimized as MKL or MLib, but provides a wider range of numerical functions

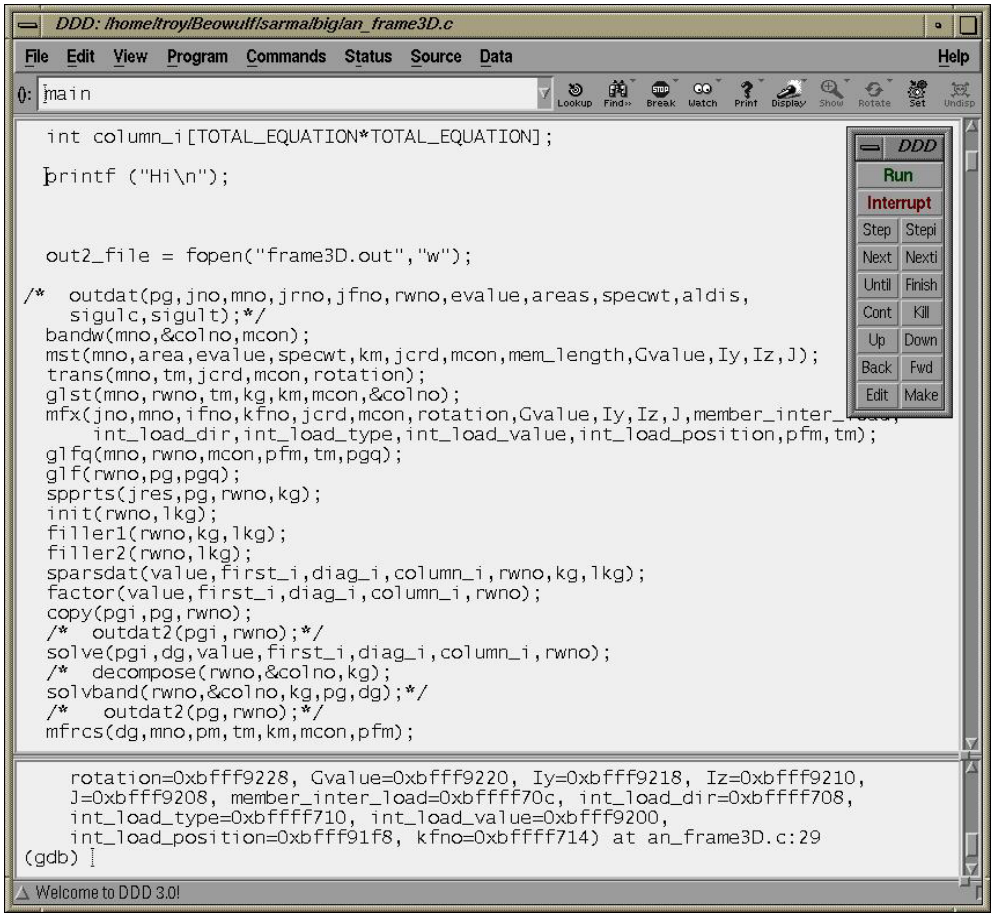
Features

- Vector and matrix operations
  - BLAS Levels 1-3 Interface
- Linear algebra
  - Some LAPACK functionality
- Eigensystems, FFTs
- Numerical integration, Ordinary Differential Equations, Interpolation and more.....

Debuggers

- Almost all Linux systems include the `gdb` command line symbolic debugger and its graphical front end `ddd`.
- In addition, Intel compilers include a command line debugger, `ldb`, but this is not ready for general use.
- OSC has also purchased a license for the `totalview` parallel debugger. This is still being ported to the IA64 platform.

Debuggers: ddd



Performance Analysis Tools

Performance analysis tools on Linux systems are currently a little more primitive than on their supercomputer cousins. However, Linux has support for the following:

- Timing
- Profiling
- Hardware performance counters

Performance Analysis Tools: Timing

- **The easiest way to time a program running on a single node is with the `/usr/bin/time` command:**

```
[mck-login1]$ /usr/bin/time ./dp4
RESP=   1.741    0.049 ERRSQ=0.43E-04 ITERAT= 1732  NITER=  1
RESP=   1.528    0.127 ERRSQ=0.84E-03 ITERAT= 1038  NITER=  2
RESP=   1.522    0.117 ERRSQ=0.86E-05 ITERAT=  818  NITER=  3
RESP=   1.522    0.116 ERRSQ=0.63E-05 ITERAT=  759  NITER=  4
123.12user 0.06system 2:03.39elapsed 99%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+420minor)pagefaults 0swaps
```

- **Note that you should hardcode the path, as some shells have a built-in `time` command which is less informative.**
  - `/usr/bin/time` will give results for
    - user time (CPU time spent running your program)
    - system time (CPU time spent by your program in system calls)
    - elapsed time (wallclock)
    - % CPU
    - memory, pagefault, and swap statistics
    - I/O statistics

Performance Analysis Tools: Timing (con't.)

You can also manually add calls to timing routines in your code:

- **C/C++**
  - Wallclock: `time`(2), `difftime`(3), `getrusage`(2)
  - CPU: `times`(2)
- **Fortran 90**
  - Wallclock: `SYSTEM_CLOCK`(3)
  - CPU: `DTIME`(3), `ETIME`(3),
- **MPI (C/C++/Fortran)**
  - Wallclock: `MPI_Wtime`(3)

Performance Analysis Tools: Profiling

Profiling is a method by which you can determine in which routines your code spends the most time. This usually requires support from the compiler as well as an analysis tool. The OSC cluster has one such tools:

- `gprof`

In addition, the OSC cluster also supports the `jumpshot` utility for profiling MPI codes.

Profiling: `gprof`

`gprof` is the GNU profiler. To use it, you need to do the following:

- Compile and link your code with the GNU compilers (`gcc`, `egcs`, `g++`, `g77`) using the `-pg` option flag.
- Run your code as usual. A file called `gmon.out` will be created containing the profile data for that run.
- Run `gprof progname gmon.out` to analyze the profile data.

Profiling: `gprof` Example

```
[mck-login1]$ g77 -O2 -pg dp4.f -o dp4
[mck-login1]$ ls
dp4     dp4.f

[mck-login1]$ ./dp4.db(...gmon.out created...)


[mck-login1]$ gprof cdnz3d \ gmon.out | more
```

Profiling: `gprof` Example (con't.)

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 24.67    942.76   942.76  4100500     0.00     0.00  lxi_
 23.51   1841.45   898.69  4100500     0.00     0.00  leta_
 20.10   2609.66   768.21  4100500     0.00     0.00  damping_
 12.64   3092.90   483.24  4100500     0.00     0.00  lzeta_
 11.55   3534.28   441.38  4100500     0.00     0.00  sum_
  4.12   3691.73   157.45      250     0.63    14.83  page_
  2.91   3802.84   111.11      250     0.44     0.44  tmstep_
  0.41   3818.62    15.78      500     0.03     0.03  bc_
  0.03   3819.59     0.97                             pow_dd
```

(…output continues…)

Profiling: `jumpshot`

`jumpshot`

a Java based profiling tool that comes with the MPICH implementation of MPI. It allows you to profile all calls to MPI routines. To use `jumpshot`, you need to do the following:

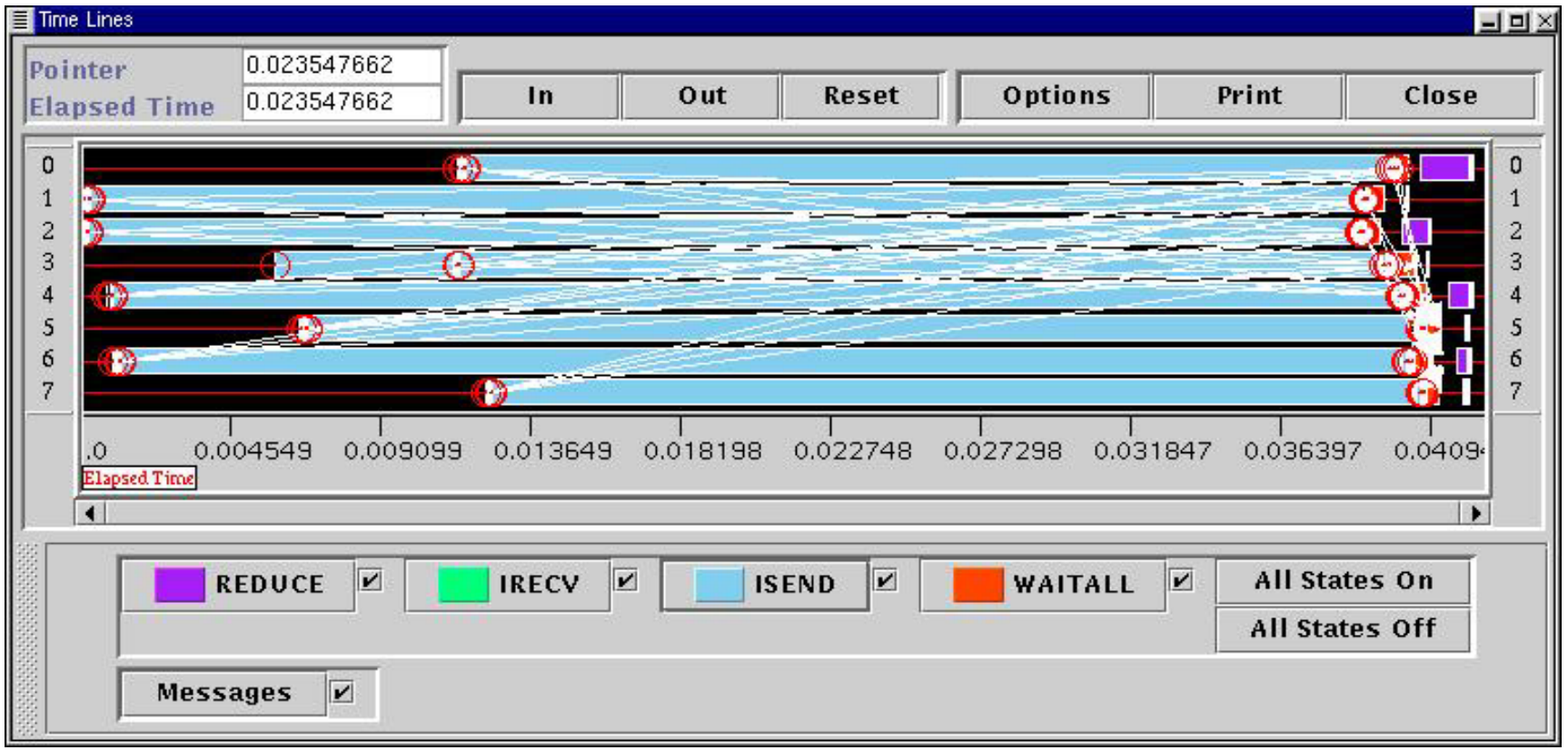
- **Compile your MPI code using one of the MPI compiler wrappers (`mpicc`, `mpiCC`, `mpif90`) using the `-mpilog` option, and link using `-lmpe`.**
- **Run your MPI code as usual. A `.clog` file will be created (i.e. if your executable is named `progname`, a log file called `progname.clog` will be created).**
- **Run `jumpshot` on the `.clog` file (eg. `jumpshot progname.clog`)**

Profiling: `jumpshot` Example

Batch script for compiling and executing an MPI program instrumented to generate jumpshot log data

```
#PBS -N mpi_test
#PBS -l walltime=5:00:00
#PBS -l nodes=2:ppn=2
#PBS -j oe
set -x
hostname
cd $HOME/work/ia64/mpi_test
mpif90 solver.f -w -mpilog -o solver -lmpe
mpiexec solver

[mck-login1]$ qsub jumpshot.pbs          (...solver.clog created...)
[mck-login1]$ /usr/local/src/mpich-1.2.1..7/mpe/viewers/jumpshot-
   3/bin/jumpshot solver.clog
```

Profiling: `jumpshot` Example (con't)



Performance Analysis Tools: Performance Counters

The Itanium-2, like most modern processors, has several event counters which can be used to measure low-level performance characteristics. HP has developed a tool called `pfmon` for accessing these counters.

- `pfmon` is a little more "bare bones" than performance counter tools on other platforms (eg. Cray's `hpm` or SGI's `perfex`) in that it only presents raw event counts -- it's up to you to turn those into MFLOPs, memory bandwidth, etc.
- `pfmon` also does not time your program, so you need to use it in concert with the `time` command.
- Usage:
  ```
  time pfmon --events=ev1[,ev2,ev3,…] myprog
  [myprog args]
  ```
- Selected events (do `pfmon --show-events` to see all):
  - BUS_MEMORY_EQ_128BYTE_SELF (# 128-byte cache line loads)
  - FP_OPS_RETIRED (# floating point ops completed)
  - IA64_INST_RETIRED (# instructions retired)

Performance Counters: `pfmon`

Batch job which runs `pfmon` on a user code:

```
#PBS -l cput=0:30:00
#PBS -l walltime=0:30:00
#PBS -l nodes=1:ppn=2
#PBS -l mem=80MB
#PBS -N cdnz3d
#PBS -j oe
#PBS -S /bin/ksh
cd $HOME/IA64/cdnz3d
cp cdnz3d cdin.dat acq.dat cdnz3dxyz.bin $TMPDIR
cd $TMPDIR
time pfmon \
 --events=BUS_MEMORY_EQ_128BYTE_SELF,FP_OPS_RETIRED,IA64_INST_RETIRED \
 ./cdnz3d
```

Output:

```
                  269816997 BUS_MEMORY_EQ_128BYTE_SELF
               288986454551 FP_OPS_RETIRED
               800202338622 IA64_INST_RETIRED
real     3m25.684s
user     3m25.586s
sys      0m0.057s
```

Computing Performance Metrics

- MIPS = 1.0e-6 * IA64_INST_RETIRED/(time in seconds)
- MFLOPS = 1.0e-6 * FP_OPS_RETIRED/(time in seconds)
- Memory bandwidth (MB/s) = 1.28e-4 * BUS_MEMORY_EQ_128BYTE_SELF/(time in seconds)
- Note that these counts may include instructions executed in mispredicted branches which are discarded, so they may overestimate actual performance by ~10%.

Batch Processing with PBS

- PBS basics
- Determining job requirements
- Creating a job script
- Submitting a job
- Monitoring a job
- Deleting a job
- Job output
- SMP jobs
- Parallel jobs
- Maui Scheduler
- Tips and Tricks

Why Batch?

- **Interactive resource limits**
  (10 min. CPU time, 32MB memory on the front end node)
  - csh: use the `limit` command to check this
  - ksh: use `ulimit -a` command to check this

- **Improves overall system efficiency by weighing user requirements against system load.**

- **Makes sure all users can get equal access to resources by enforcing a scheduling policy.**

- **Only way to access compute nodes!**

Introduction to PBS

- PBS is short for “Portable Batch System”; it is an open source batch queuing system.
- It is an outgrowth/extension of the NQS batch queuing system from the NAS project at NASA Ames Research Center.
- PBS is available for virtually anything that is UNIX-like, including Linux, the BSDs, UNICOS, IRIX, Solaris, AIX, HP/UX, and Digital UNIX.

How PBS Handles a Job

- User determines resource requirements for a job and writes a batch script.
- User submits job to PBS with the `qsub` command.
- PBS places the job into a queue based on its resource requests and runs the job when those resources become available.
- The job runs until it either completes or exceeds one of its resource request limits.
- PBS copies the job's output into the directory from which the job was submitted and optionally notified the user via email that the job has ended.

Determining Job Requirements

- **For single CPU jobs, PBS needs to know three resource requirements:**
  - Wall-clock time
  - Memory

- **For multiprocessor parallel jobs, PBS also needs to know how many nodes/CPUs are required.**
  - Don't need to specify memory limits for parallel jobs

- **Other things to consider:**
  - Job name?
  - Working in `/tmp` or `$TMPDIR`?
  - Where to put standard output and error output?
  - Should the system email when the job completes?

Determining Job Requirements (con't)

- **Memory requirements can be estimated using the `size` command:**

```
[mck-login1]$ size dp4
   text     data              bss       dec      hex  filename
1650312    71928  6044136  7766376   768168  dp4
```

- **The output of `size` is in bytes, so this program above requires about 7 MB. Note that the `size` command does not take dynamic memory into account.**
- **CPU time requirements can be determined by running short jobs interactively; however, this requires you to understand how CPU time scales with the size of your problem.**
- **The number of CPUs used is up to you, but you are limited to the number physically available (currently 256 for parallel jobs).**

PBS Job Scripts

- An PBS job script is just a regular shell script with some comments (the ones starting with `#PBS`) which are meaningful to PBS. These comments are used to specify properties of the job.
- PBS job scripts always start in your home directory, `$HOME`. If you need to work in another directory, your job script will need to `cd` to there.
- Every PBS job has a unique temporary directory, `$TMPDIR`. This in on each compute node's local disk array and thus is much faster than your home directory, which is mounted over the network from the mass storage server. For best I/O performance, you should try to copy all the files you need into `$TMPDIR`, do your work there, and then copy any files you want to keep back to your home directory.

PBS Job Scripts (con't)

- **Useful PBS options:**
  - `-e errfile` (redirect standard error to `errfile`)
  - `-I` (run as an interactive job)
  - `-j oe` (combine standard output and standard error)
  - `-l walltime=N` (request N seconds of wallclock time; N can also be in hh:mm:ss form)
  - `-l vmem=N[KMG][BW]` (request `N` {kilo|mega|giga}{bytes|words} of virtual memory per node)
  - `-l nodes=N:ppn=M` (request N nodes with M processors per node)
  - `-m e` (mail the user when the job completes)
  - `-m a` (mail the user if the job aborts)
  - `-o outfile` (redirect standard output to `outfile`)
  - `-N jobname` (name the job `jobname`)
  - `-S shell` (use `shell` instead of your login shell to interpret the batch script; must include a complete path)
  - `-V` (job inherits the full environment of the current shell, including `$DISPLAY`)

A First Batch Script

- **Here is a simple batch job:**

```
[mck-login1]$ cat dp4.job
#PBS -N dp4
#PBS -l walltime=5:00:00
#PBS -l nodes=1:ppn=1
#PBS -j oe
set -x
hostname
cd $TMPDIR
cp $HOME/work/ia64/dp4/dp4.f .
efc -O3 -w -ftz dp4.f -o dp4
time ./dp4
cp *.dat $HOME/work/ia64/dp4
```

This job asks for one CPU on one node, 40MB of memory, and 5 hours of CPU time. Its name is "dp4"

Submitting a Job

- **To submit a job to PBS, use the `qsub` command:**

```
[mck-login1]$ qsub wakko.job
38125.ia64.osc.edu
```

- **`qsub` can take all of the options show for job scripts on the command line as well; specifying these on the command line overrides settings in the job script.**

- **Notice that `qsub` prints a request number (38125 in the case shown above). This number is important for finding the output files generated by this run as well as for killing the job if necessary.**

Monitoring a Job

- **The status of batch jobs can be shown with the `qstat` command:**

```
[mck-login1]$ qstat -a

ia64.osc.edu:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
38125.ia64.osc. jimg     serial   dp4         27831   1  --  490mb 05:00 R 00:00
```

- **Progress of running batch jobs can be monitored with `qpeek`:**

```
[mck-login1]$ qpeek 38126
tset: standard error: Inappropriate ioctl for device
Enter UNIX terminal type, tset: standard error: Inappropriate ioctl for device
+ hostname
node72
+ cd /tmp/pbstmp.38126.ia64.osc.edu
+ cp /home/jimg/work/ia64/dp4/dp4.f .
+ efc -O3 -w -ftz dp4.f -o dp4
   program DPOSC
   external subroutine DPINP
   external subroutine DPINIT
```

`qstat` Output Fields

- **`Job Id` (request number)**
- **`Username` (userid)**
- **`Queue` (queue the job is in)**
- **`Jobname` (name of the job)**
- **`SessId` (job identifier)**
- **`NDS` (number of nodes requested)**
- **`TSK` (number of CPUs per node requested)**
- **`Req'd Memory` (memory requested [if waiting] or used [if running])**
- **`Req'd Time` (CPU time requested)**
- **`S` (status)**
  - `R` (running)
  - `Q` (queued and waiting)
- **`Elap Time` (time the job has been running)**
- **nodes the job is running on**

Killing a Job

- If, for whatever reason, you need to delete a queued job or kill a running job, use the `qdel` command.

- Usage: `qdel request_number`

Job Output

- **When an PBS job ends, it writes out two files in the directory from which it was submitted:**
  - `<jobname>.e<request_number>` (stanard error)
  - `<jobname>.o<request_number>` (standard output)

- **These two files can be combined using the `-j oe` option, or directed to set file names using the `-e errfile` and `-o outfile` options.**

- **These are in addition to any files generated by the programs run in your job.**

SMP Jobs

So far, the job scripts we've seen have been serial, uniprocessor jobs. The following is an example of a job that used more than one processor on a single node:

```
[mck-login1]$ cat dp4.job
#PBS -N dp4
#PBS -l walltime=5:00:00
#PBS -l nodes=1:ppn=2
#PBS -j oe
set -x
hostname
cd $TMPDIR
cp $HOME/work/ia64/dp4/dp4.f .
efc -O3 -w -ftz dp4.f -o dp4
time ./dp4
efc -O3 -w -ftz -parallel dp4.f -o dp4
export OMP_NUM_THREADS=2
time ./dp4
```

More on SMP and Serial Jobs

- The only real difference between a uniprocessor job and an SMP job (at least from PBS's point of view) is the

  ```
  -l nodes=1:ppn=2
  ```

  limit in the SMP job. This tells PBS to allow the job to run two processes (or threads) concurrently on one node.

- If you simply request a number of nodes (eg. `-l nodes=1`), PBS will assume that you want one processor per node.

Parallel Jobs

Both serial and SMP jobs run on only 1 node. However, most MPI programs should be run on more than 1 node. Here is an example of how to do that:

```
[mck-login1]$ cat parallel.job
#PBS -N mpi_test
#PBS -l walltime=5:00:00
#PBS -l nodes=2:ppn=2
#PBS -j oe
set -x
hostname
cd $HOME/work/ia64/mpi_test
mpif90 solver.f -w -mpilog -o solver -lmpe
mpiexec solver
```

More on Parallel Jobs

- The `mpiexec` command is used to run MPI jobs over the Myrinet in PBS. It figures out from PBS on which nodes a job is supposed to run and starts it running on only those nodes.
- You can use (and we encourage you to use!) more than one processor per node in parallel jobs. To use two processors per node on `N` nodes, add a `-l nodes=N:ppn=2` limit to your job.
- If you need to run a shell command on all of the nodes assigned to your job (eg. copying a data file into or out of `$TMPDIR`), use `pbsdcp`:

```
cd $HOME/work/ia64/mpi_test
# scatter executable and input
pbsdcp solver input.dat $TMPDIR
cd $TMPDIR
mpiexec ./solver
# gather output files
pbsdcp -g "*.dat" $HOME/work/ia64/mpi_test
```

Maui Scheduler

- **OSC uses the Maui scheduler rather than the scheduler that comes with PBS, because Maui has a number of features that the PBS schedulers do not:**
  - Advance reservations
  - Fair-share scheduling
  - Quality of service levels
- **Maui also comes with its own set of tools for checking on job state:**
  - `showq` (lists currently running and queued jobs)
  - `showstart` (estimates start time of a job)
  - `showbf` (describes resources currently available for backfill scheduling)

Maui Scheduler: showq

```
[mck-login1]$ showq
ACTIVE JOBS--------------------
           JOBNAME USERNAME      STATE  PROC   REMAINING            STARTTIME

             …
             87706  osu2779    Running    16    10:57:11  Thu May  2 09:13:08
             87710  osu2778    Running    16    11:03:27  Thu May  2 09:19:24
             87712  osu2778    Running    16    11:05:12  Thu May  2 09:21:09
             88620     troy    Running     4  1:00:33:20  Fri May  3 13:49:17
             87063  osu2779    Running     8  1:03:10:00  Wed May  1 09:25:57
             …
    25 Active Jobs     133 of  142 Processors Active (93.66%)
                        67 of   71 Nodes Active      (94.37%)

IDLE JOBS----------------------
           JOBNAME USERNAME      STATE  PROC     WCLIMIT            QUEUETIME
             88519   utl170       Idle     1  3:01:01:00  Fri May  3 11:06:01
1 Idle Job

BLOCKED JOBS----------------
           JOBNAME USERNAME      STATE  PROC     WCLIMIT            QUEUETIME
             88449  osu2779       Idle    16  1:16:00:00  Fri May  3 08:52:47
Total Jobs: 27   Active Jobs: 25   Idle Jobs: 1   Blocked Jobs: 1
```

OSC Innovations in computing, networking, and education

Terminology of `showq` Listings

- Active jobs: Jobs which are currently running. Active jobs are listed in order of expected completion, soonest first.

- Idle jobs: Jobs which are not running but can run once sufficient resources become available. Idle jobs are listed in priority order, for highest to lowest. The highest priority idle job has a reservation to run as soon as sufficient resources become available; all other idle jobs are candidates for backfill.

- Blocked jobs: Jobs which are not running because they are held or exceed a job limit policy. Blocked jobs will not run until they move into the idle jobs list.

Maui Scheduler: `showstart`

```
[mck-login1]$ showstart 88519
job 88519 requires 1 proc for 3:01:01:00
Earliest start is in       7:02:27:31 on Fri May 10 17:00:00
Earliest Completion is in 10:03:28:31 on Mon May 13 18:01:00
Best Partition: DEFAULT
```

- **Note that `showstart` only gives the scheduler's best estimate based on the current queue state, which can (and will) change as jobs are submitted, run, and exit.**
- **Advance reservations (such as scheduled system downtime) can also influence job start time.**

Maui Scheduler: showbf

```
[mck-login1]$ showbf
backfill window (user: 'troy' group: 'G-0541' partition: ALL)
Fri May  3 14:41:14

 23 procs available for    2:17:18:46
```

- **`showbf` can give a rough idea of how many processors are immediately available, and how long they're available for.**
- **As with `showstart`, the output from `showbf` is only an estimate.**

Tips and Tricks

- The following `csh` aliases are handy for checking what the PBS load on the Itanium-2 cluster is like:

```
alias myjobs 'qstat -a | grep `whoami`'
alias rjobs 'qstat -r | grep "R[0-9 ]" | grep -v IDENT'
alias nrjobs 'rjobs | wc -l'
alias qjobs 'qstat -i | grep "Q[a-z ]" | grep -v TSK'
alias nqjobs 'qjobs | wc -l'
alias njobs 'echo `nrjobs` running \+ `nqjobs` queued'
```

- There is also a graphical utility called `xpbs` which you can use to construct, submit, and track PBS jobs.

SMP Programming with OpenMP

- What's OpenMP?
- A simple OpenMP program
- Compiling OpenMP programs
- Running OpenMP programs
- OpenMP and the Intel compilers

What's OpenMP?

OpenMP is a *de facto* standard for portable, directive-based threaded parallel programming for SMP systems. It consists of:

- A set of compiler directives.
- A library of support functions.

OpenMP is supported by a number of vendors' compilers, including SGI, Compaq Digital, the Portland Group, and Kuck and Associates (recently purchased by Intel).

A Simple OpenMP Program

```
[mck-login1]$ more omphw.f90
PROGRAM omphw
IMPLICIT NONE
INTEGER,EXTERNAL :: omp_get_thread_num

!$OMP PARALLEL
WRITE (*,*) 'Hello from thread ',omp_get_thread_num()
!$OMP END PARALLEL

END PROGRAM omphw
```

Compiling OpenMP Programs

- To compile a program with OpenMP support, you need to use one of the Intel compilers and add the `-openmp` option:

```
[mck-login1]$ efc -openmp omphw.f90 -o omphw
```

- This is in addition to any optimization flags and so forth, of course.

- If you have an OpenMP program that uses one of the library calls (like `omp_get_thread_num()` in the previous example) and compile without the `-mp` flag, the compiler will complain that it can't link in the OpenMP library.

Running OpenMP Programs

To run an OpenMP program, you first need to tell the program how many threads to use. This can be done in two ways:

- **Hardcoded into the program source using the `omp_set_num_threads()` function.**
- **Set at run-time using the `OMP_NUM_THREADS` environment variable.**

Once the number of threads is set, you can run your program like any other:

- csh: setenv OMP_NUM_THREADS 2
- ksh: export OMP_NUM_THREADS=2

```
[mck-login1]$ ./omphw
 Hello from thread 0
 Hello from thread 1
```

OSC
Innovations in computing, networking, and education

Running OpenMP Programs in Batch

To run an OpenMP program in batch, make sure to request multiple processors and set `OMP_NUM_THREADS` in your batch job:

```
[mck-login1]$ cat dp4.job
#PBS -N dp4
#PBS -l walltime=5:00:00
#PBS -l nodes=1:ppn=2
#PBS -j oe
set -x
cd $TMPDIR
cp $HOME/work/ia64/dp4/dp4_omp.f .
efc -O3 -w -ftz -mp dp4_omp.f -o dp4_omp
export OMP_NUM_THREADS=2
time ./dp4_omp
```

Parallel Programming with MPI

- What's MPI?
- A simple MPI program
- Compiling MPI programs
- Running MPI programs

What's MPI?

MPI is the *de facto* standard for portable message passing parallel programming on distributed memory systems. It consists of:

- A message passing library
- A run-time environment (`mpiexec` and compiler wrappers)

MPI is supported by all of the major parallel machine manufacturers (Compaq Digital, IBM, SGI, Sun), and there are several third-party implementations for various hardware and software platforms. The OSC cluster uses MPICH/ch_gm, which is a version of the MPICH reference implementation of MPI that runs over Myrinet; this supports all of the MPI-1.1 standard as well as the MPI-IO part of the MPI-2 standard.

A Simple MPI Program

```
      PROGRAM sample2
C Run with four processes
      INCLUDE 'mpif.h'
      INTEGER err, rank, size
      integer status(MPI_STATUS_SIZE)
      integer x,y,z
      common/point/x,y,z
      integer ptype
      CALL MPI_INIT(err)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
      call MPI_TYPE_CONTIGUOUS(3,MPI_INTEGER,ptype,err)
      call MPI_TYPE_COMMIT(ptype,err)
      print *,rank,size
      if(rank.eq.3) then
         x=15
         y=23
         z=6
         call MPI_SEND(x,1,ptype,1,30,MPI_COMM_WORLD,err)
        else if(rank.eq.1)then
         call MPI_RECV(x,1,ptype,3,30,MPI_COMM_WORLD,status,err)
         print *,'P:',rank,' coords are ',x,y,z
        end if
      CALL MPI_FINALIZE(err)
      END
```

Compiling MPI Programs

- To compile an MPI program, you need to compile with the MPI compiler wrappers (`mpicc`, `mpiCC`, `mpif77`, and `mpif90`):

```
mpif90 sample2.f -w -o sample2
```

- The MPI compiler wrappers accept the same arguments as the compilers they wrap as far as optimization and so forth.

Running MPI Programs

- **To manage contention for the Myrinet SAN, OSC asks that MPI programs be run in batch only.**
- **OSC provides a program called `mpiexec` which automatically determines from PBS on which nodes an MPI program is allowed to run and starts the program running:**

```
[mck-login1]$ cat parallel.job
#PBS -N mpi_test
#PBS -l walltime=5:00:00
#PBS -l nodes=2:ppn=2
#PBS -j oe
set -x
hostname
cd $HOME/work/ia64/mpi_test
mpif90 sample2.f -w -o sample2
mpiexec ./sample2
```

Running MPI Programs (con't.)

```
[mck-login1]$ qsub parallel.job
38128.ia64.osc.edu
[mck-login1]$ qstat -an
ia64.osc.edu:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
38128.ia64.osc. jimg     parallel mpi_test    28081   2  --    2gb 05:00 R   --
   node72/1+node72/0+node71/1+node71/0
```

Running MPI Programs (cont.)

```
[mck-login1]$ cat mpi_test.o38128
tset: standard error: Inappropriate ioctl for device
Enter UNIX terminal type, tset: standard error: Inappropriate ioctl for device
+ hostname
mck072
+ cd /home/jimg/work/ia64/mpi_test
+ mpif90 sample2.f -w -o sample2
   program SAMPLE2
231 Lines Compiled
+ mpiexec ./sample2
           3           4
           2           4
           0           4
           1           4
 P:           1  coords are           15          23           6
```

Multilevel Parallel Programming

- A collision between OpenMP and MPI
- A simple multilevel parallel program
- Compiling multilevel parallel programs
- Running multilevel parallel programs
- When does multilevel parallel make sense?
- Multilevel parallelism tips and tricks

A Collision Between OpenMP and MPI

- Because of the architecture of the Itanium cluster (i.e. a cluster of SMP systems), it is possible to write programs which take advantage of both the message passing and shared memory environments simultaneously.
- In this type of approach, MPI message passing is used for coarse-grained parallelism between nodes, and OpenMP compiler directives are used for fine-grained parallelism within a node.
- This approach allows you to take maximum advantage of the compute power of the nodes, because there is less contention between multiple processes for the Myrinet interface cards.
- Codes written in this fashion are also ready for use on extremely large systems such as the DOE ASCI Blue Mountain (6000+ CPUs) and ASCI White (8000+ CPUs) supercomputers and for use on computational grid systems.

A Simple Multilevel Parallel Program

```
[mck-login1]$ more mlhw.f90
PROGRAM mlhw
INCLUDE 'mpif.h'
INTEGER :: ierr,rank,size,tnum
INTEGER,EXTERNAL :: omp_get_thread_num
CALL MPI_Init(ierr)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rank,ierr)
CALL MPI_Comm_size(MPI_COMM_WORLD,size,ierr)
CALL omp_set_num_threads(4)
!$OMP PARALLEL PRIVATE(tnum)
tnum=omp_get_thread_num()
!$OMP CRITICAL
WRITE (*,*) 'Hello from thread ',tnum,' on node ',rank,' of ',size
!$OMP END CRITICAL
!$OMP BARRIER
!$OMP END PARALLEL
CALL MPI_Finalize()
END PROGRAM mlhw
```

Compiling Multilevel Parallel Programs

- You need to compile multilevel parallel programs with the MPI compiler wrappers (`mpicc`, `mpif90`, etc.).
- However, you also need to use the `-openmp` option to enable OpenMP support:

```
[mck-login1]$ mpif90 -O2 -openmp -o mlhw mlhw.f90
```

- This should also work using the PCF directives rather than OpenMP; however, OpenMP is the preferred method.

Running Multilevel Parallel Programs

- As with MPI programs, multilevel parallel programs must be run in batch.

- Also, setting the `OMP_NUM_THREADS` environment variable does not work in multilevel parallel programs; you must use the `omp_set_num_threads()` library call in your program instead.

Running Multilevel Parallel Programs in Batch

```
[mck-login1]$ more mlhw.pbs
#PBS -l nodes=4:ppn=2
#PBS -j oe
#PBS -N mlhw
cd $HOME/multilevel
mpiexec -pernode ./mlhw

[mck-login1]$ qsub mlhw.pbs
31822.ia64.osc.edu
```

Running Multilevel Parallel Programs in Batch (con't.)

```
[mck-login1]$ more mlhw.o31822
Hello from thread 0 on node 0 of 4
Hello from thread 0 on node 1 of 4
Hello from thread 0 on node 2 of 4
Hello from thread 1 on node 0 of 4
Hello from thread 0 on node 3 of 4
Hello from thread 1 on node 2 of 4
Hello from thread 1 on node 1 of 4
Hello from thread 1 on node 3 of 4
```

When Does Multilevel Parallel Make Sense?

Obviously, multilevel parallel programming is not easy, because you need to know both OpenMP and MPI. However, the following types of applications may benefit from multilevel parallel approaches:

- Existing MPI applications which have vector-style nested loop computations.
- Existing OpenMP applications which are amenable to domain decomposition with MPI.
- Applications with multiple interacting grid zones which can be treated “mostly” independently (eg. multiblock CFD solvers).

Multilevel Parallelism Tips and Tricks

- Make sure all MPI calls are outside of any OpenMP parallel regions; otherwise each thread will try to call the MPI routine, possibly resulting in deadlock. (In theory, wrapping MPI calls in `master` or `single` directives should also work; however, in practice this seems to have problems.)

- You should only invoke as many MPI processes as there are Myrinet interface cards available to your job (currently 1 per node). You can use the `-n N` option to force `mpiexec` to start `N` MPI processes rather than the default of 1 MPI process per requested processor. You can also use the `-pernode` option to force `mpiexec` to run 1 MPI process per node.

PVFS Parallel File System

- **OSC Parallel I/O Cluster**
- **Accessing PVFS**
- **Examples**
  - Serial jobs
  - Parallel jobs
- **When to Use PVFS**
- **Caveats**

OSC Parallel I/O Cluster

- **A new service, currently accessible only from the Itanium cluster, is the parallel I/O cluster:**
  - 16 I/O nodes, each with
    - 2 Pentium III processors running at 933MHz
    - 1 GB RAM
    - 3ware IDE RAID controller
    - 8 80-GB disks in RAID 5 (~520 GB usable space)
    - Gigabit and 100Mbit Ethernet interfaces
  - PVFS software from Clemson University and Argonne National Lab
    - Equivalent of RAID 0 (striping) across the I/O nodes
    - ~8 TB of usable space, mounted on `/pvfs`
    - Large block sizes (64kB by default, settable on a per-file basis at the time of file creation)
    - Accessible in two ways
      - Linux file system driver for standard UNIX file semantics
      - MPI-IO for high performance parallel I/O

Accessing the PVFS Parallel File System

- To access the PVFS file system from a batch job, you'll need to tell the batch system you intend to use it by adding a `pvfs` attribute to your job's `nodes` request:

```
#PBS -l nodes=4:ppn=2:pvfs
```

- In a batch job which requests PVFS, there will be an environment variable `$PFSDIR` -- this is similar to `$TMPDIR` in that it is a directory that only exists for the duration of the job, but it resides on PVFS and is accessible by all the nodes in your job (as opposed to `$TMPDIR` which is private to each node).

Example: Serial Job Using PVFS

```
[mck-login1]$ cat bigfile.pbs
#PBS -N bigfile
#PBS -j oe
#PBS -l nodes=1:ppn=2:pvfs
#PBS -l walltime=10:00:00
cd myscience
cp input.dat $PFSDIR
cd $PFSDIR
$HOME/myscience/bigfileapp
cp output.dat $HOME/myscience
```

Example: Parallel Job Using PVFS

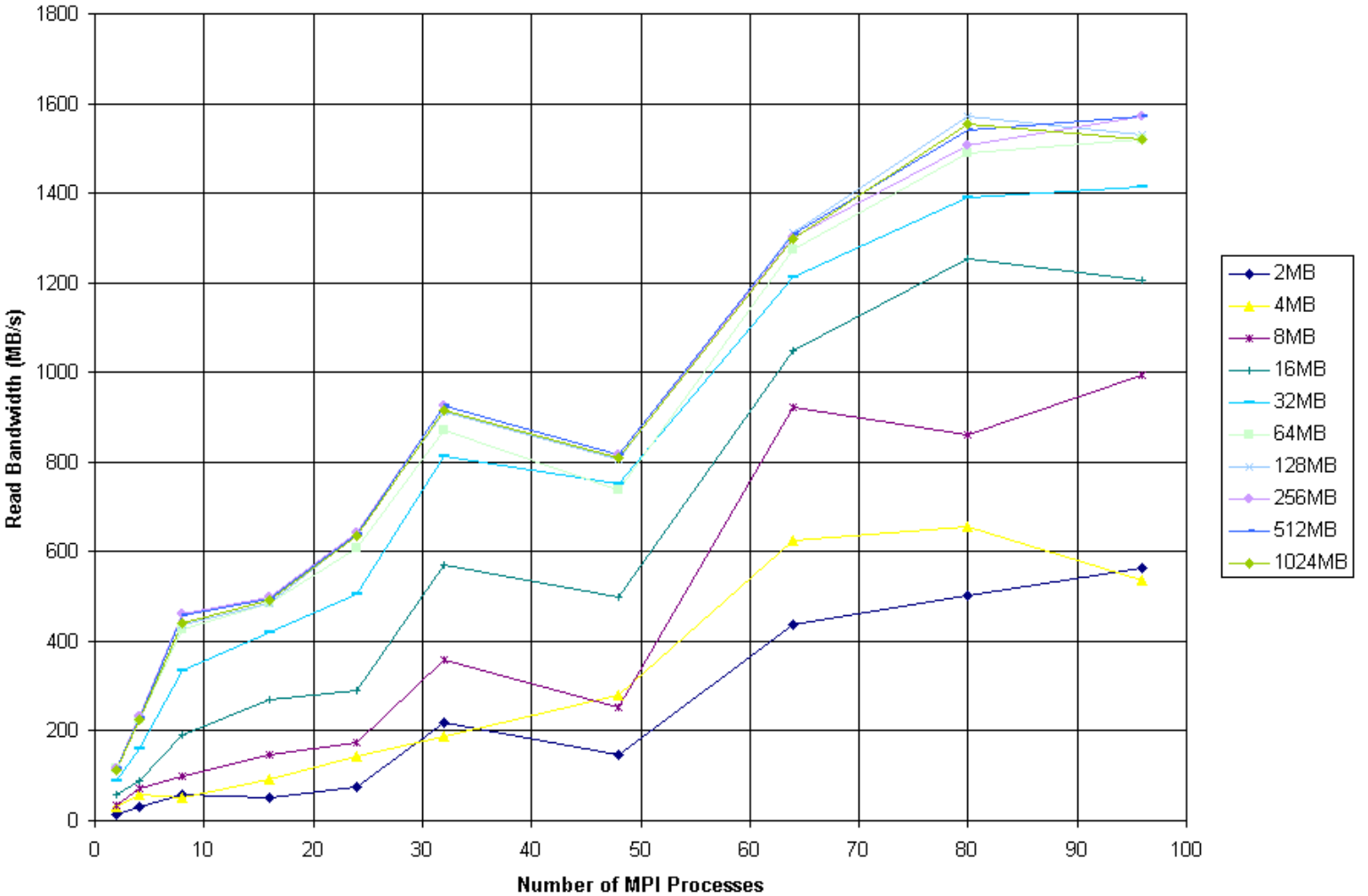
```
[mck-login1]$ cat mpi-io.pbs
#PBS -N mpi-io
#PBS -j oe
#PBS -l nodes=8:ppn=2:pvfs
#PBS -l walltime=24:00:00
cd $HOME/myscience
pbsdcp parallel-io-app $TMPDIR
cp input.dat $PFSDIR
cd $PFSDIR
mpiexec $TMPDIR/parallel-io-app
cp output.dat $HOME/myscience
```

When to Use PVFS

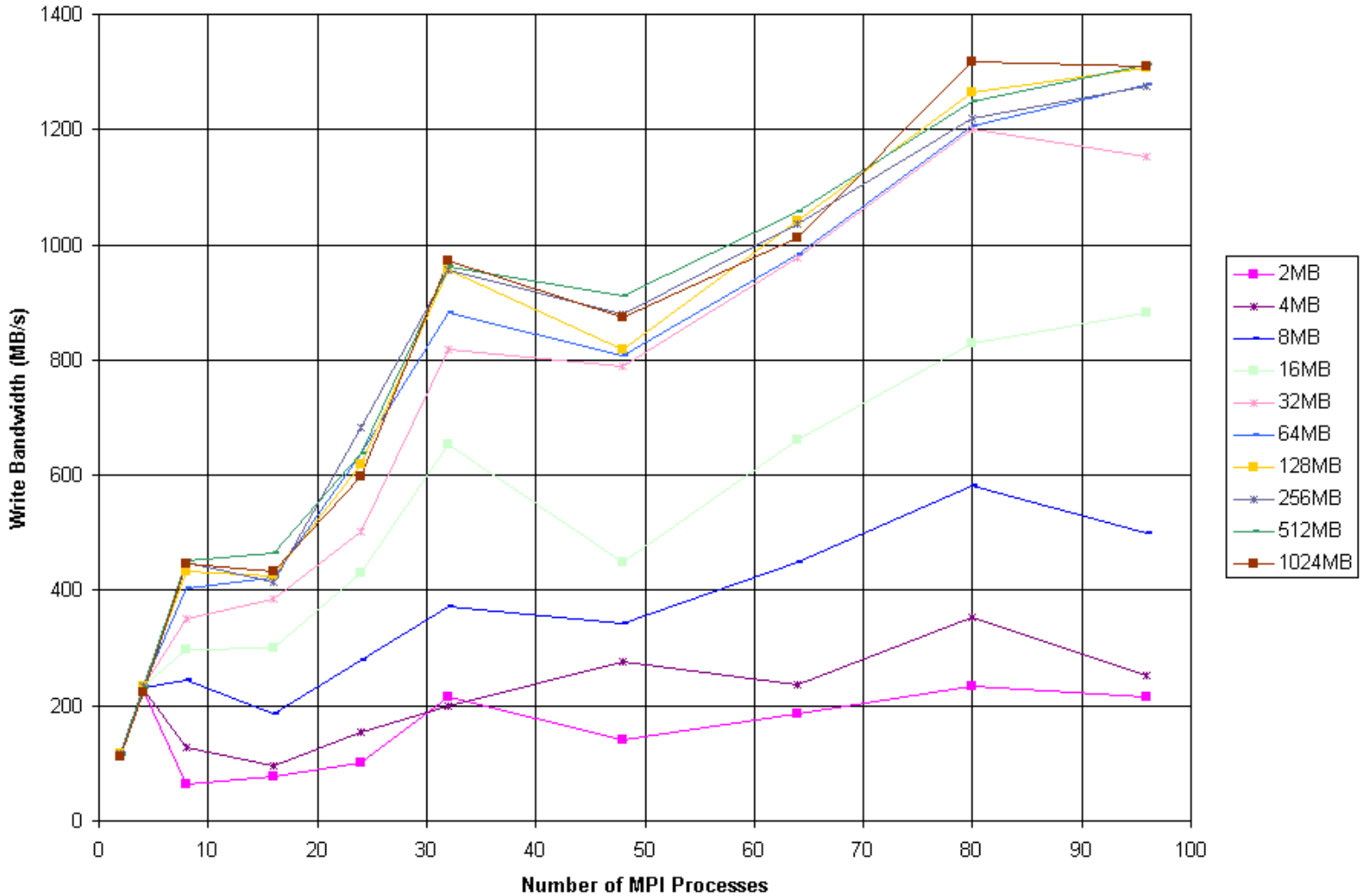
- Jobs which need large scratch files: `/tmp` on each node has roughly 70 GB of available space. By comparison, `/pvfs` has just short of 8 TB (i.e. over two orders of magnitude more) available, is globally accessible, and is about as fast as a locally attached single disk in most cases.

- Jobs which use MPI-IO: Parallel programs which use the MPI-2 I/O routines (`MPI_File_*()`) to PVFS will see significant performance improvements over doing I/O to `/tmp` or `/home`. I/O rates of over 1.5 GB/s have been observed for jobs with large node counts, and rates of 100-400 MB/s are commonplace.

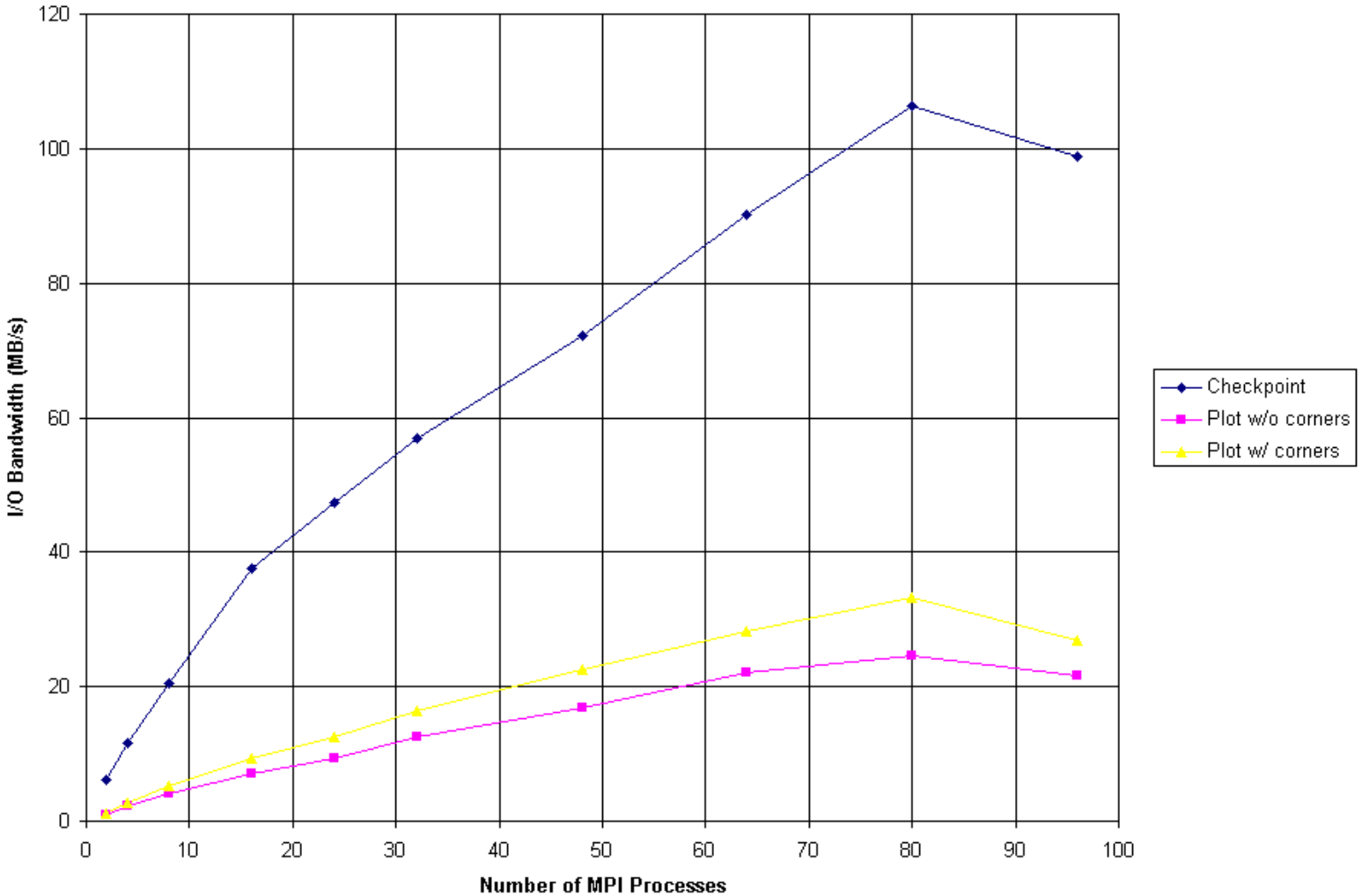
MPI Parallel Read Performance to PVFS



MPI Parallel Write Performance to PVFS



ASCI Flash Parallel I/O Benchmark



PVFS Caveats

- **`/pvfs` is NOT backed up!!! It is intended strictly for temporary use; do not keep files on it over the long term without also storing them in your home directory.**

- **Do not store executables on `/pvfs`; while this will often work, it occasionally will cause odd problems.**

Other Sources of Information

- Online manuals
- Software documentation on the web
- Related workshop courses

Online Manuals

- Like most UNIX-like systems, Linux includes a set of reference manuals as part of the operating system. This can be accessed by typing `man cmdname`, where `cmdname` is the name of the command or library routine for which you need information.

- You can also do a keyword search of all of the currently accessible manual pages by running `man -k keyword`.

Software Documentation on the Web

Many current software packages have documentation in web-accessible HTML format in addition to (or as a replacement for) standard UNIX man pages. Some examples of this include:

- The Myricom GM low-level communications library for Myrinet (http://www.myri.com/GM/doc/gm_toc.html)
- The PBS queuing system (http://www.openpbs.org/docs.html)
- The Intel compiler suite (http://oscinfo.osc.edu/manuals)
- The Intel Itanium Processor homepage (http://developer.intel.com/design/itanium/index.htm)

Other software documentation can be found at http://oscinfo.osc.edu/software

Related Workshop Courses

OSC offers several other courses which many be of interest to users of the OSC cluster:

- C Programming
- Features of the C++ Programming Language
- An Introduction to Fortran 90
- Parallel Programming with MPI
- Parallel Programming with OpenMP
- Using the ScaLAPACK Parallel Numerical Library
- Parallel I/O Techniques
- Performance Tuning for Microprocessor Architectures

More information on these courses and more can be found at http://oscinfo.osc.edu/training/.

PBS Structure

SERVER

SCHEDULER

MOM

PBS Server

- There is one server process
- It creates and receives batch jobs
- Modifies batch jobs
- Invokes the scheduler
- Instructs moms to execute jobs

PBS Scheduler

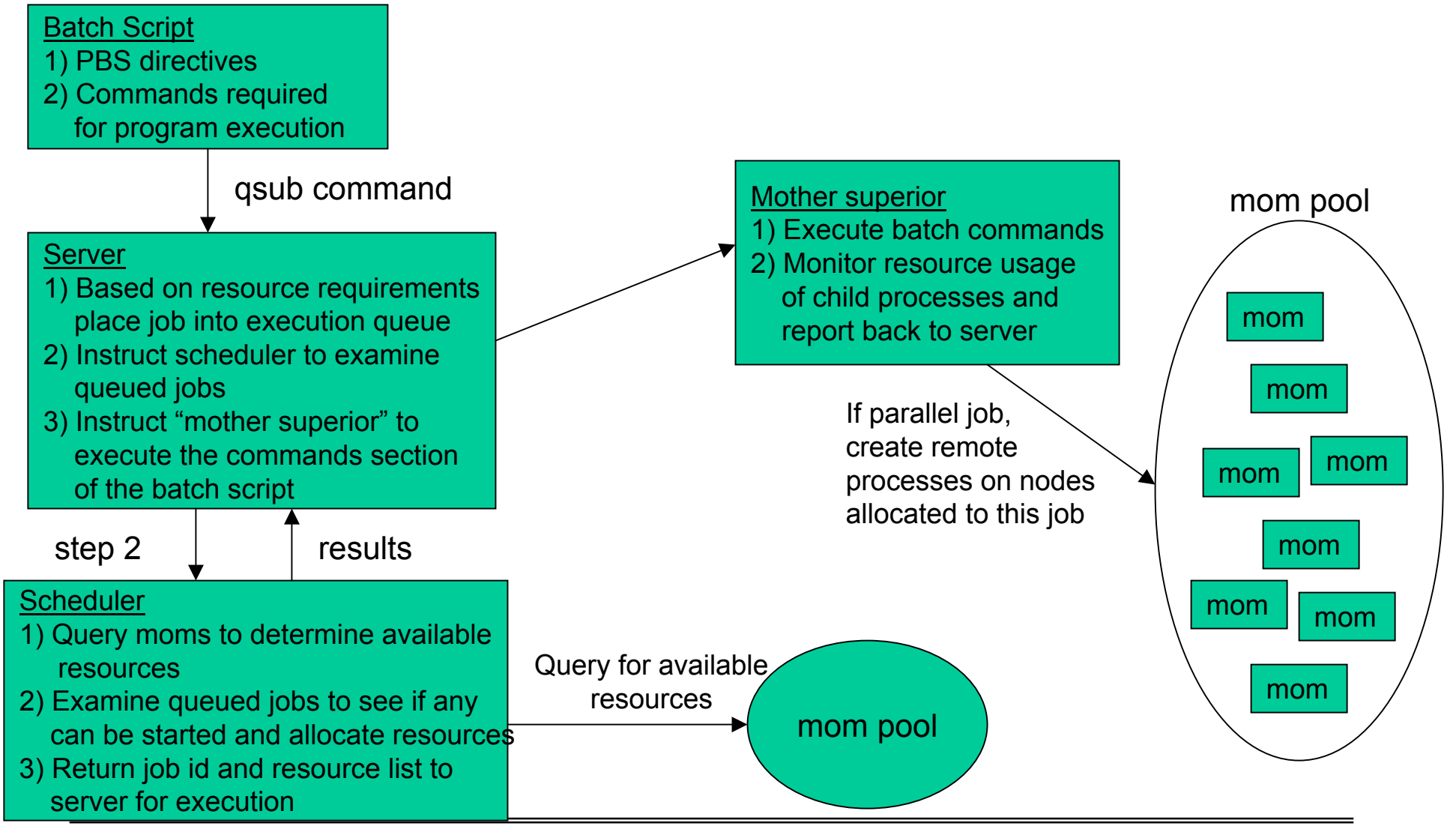
- There is one scheduler process
- Contains the policy controlling which job is run, where and when it is run
- Communicates with the "moms" to learn about state of system
- Communicates with server to learn about the availability of jobs

PBS Machine Oriented Miniserver

- One process required for each compute node
- Places jobs into execution
- Takes instruction from the server
- Requires that each instance have its own local file system

PBS provides an Application Program Interface (API) to communicate with the server and another to interface with the moms

How PBS Handles Jobs



Parallel Job Control - `mpiexec`

PBS Task Manager

- In addition to the PBS API, which provides access to the PBS server, there is a task manager interface for the moms
- Based on the PSCHED API (http://parallel.nas.nasa.gov/PSCHED)

Mpiexec uses the task manager library of pbs to spawn copies of the executable on all the nodes in a pbs allocation. It is functionally equivalent to

```
rsh node "cd $cwd; $SHELL -c 'cd $cwd; exec executable arguments'"
```

The PBS server API is used to extract resource request information and construct the resource configuration file (nodes, etc.)

We use GM, which requires information on NICs, that is constructed by mpiexec as well (PBS does not know about NICs)

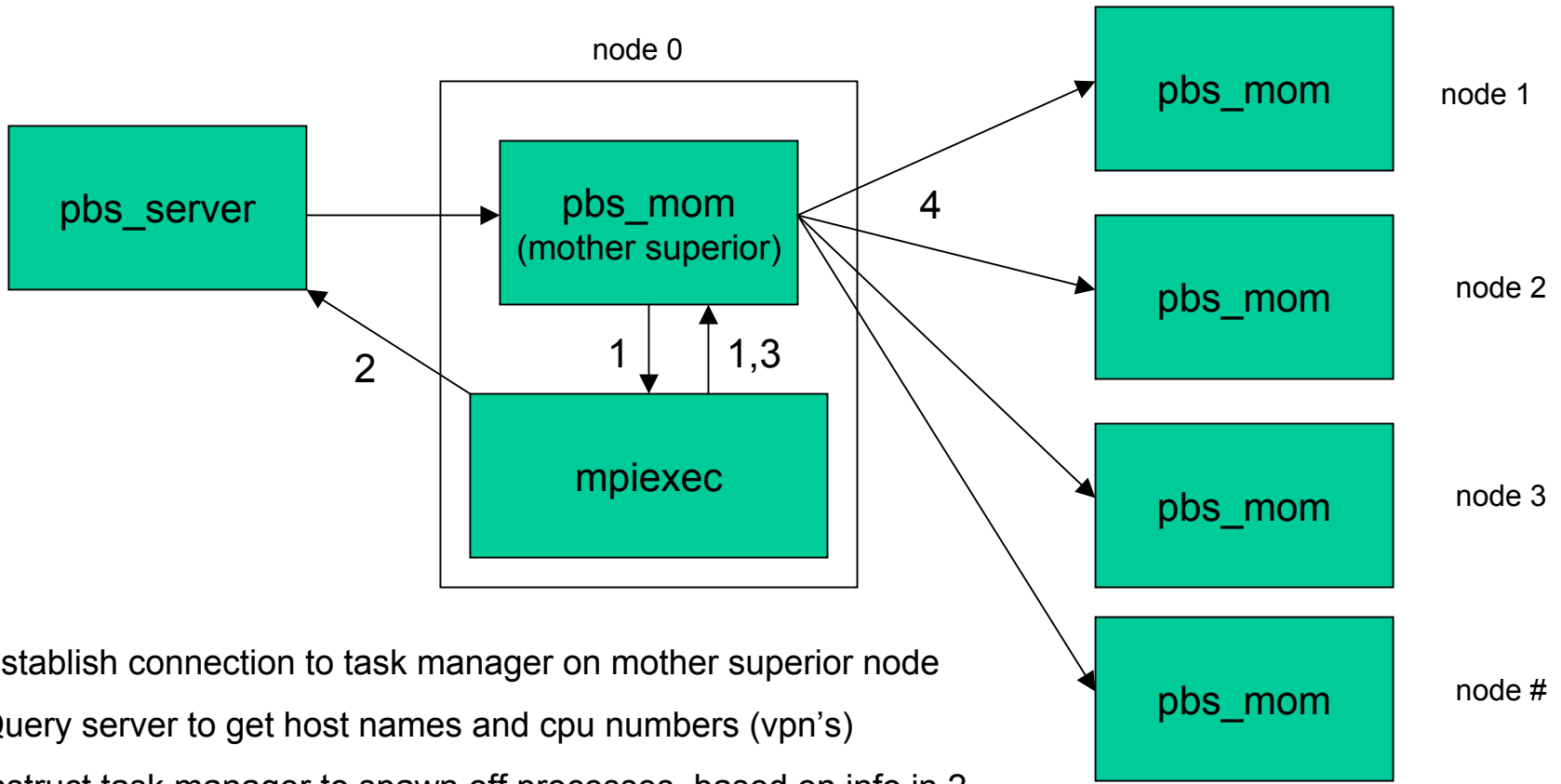
`mpiexec` Format

```
mpiexec [OPTION]... executable [args]...
```

`-n numproc`	**Use only the specified number of processes**
`-tv, -totalview`	**Debug using totalview**
`-perif`	**Allocate only one process per myrinet interface** This flag can be used to ensure maximum communication bandwidth available to each process
`-pernode`	**Allocate only one process per compute node. For SMP nodes, only one processor will be allocated a job. This flag is used to implement multiple level parallelism with MPI between nodes, and threads within a node**
`-config configfile`	**Process executable and arguments are specified in the given configuration file. This flag permits the use of heterogeneous jobs using multiple executables, architectures, and command line arguments.**
`-bg, -background`	**Do not redirect stdin to task zero. Similar to the "-n" flag in rsh(1).**

MPIEXEC

C program written at OSC for PBS and available under GPL



1) Establish connection to task manager on mother superior node
2) Query server to get host names and cpu numbers (vpn's)
3) Instruct task manager to spawn off processes, based on info in 2
4) Mother superior instructs moms in her pool to start indicated tasks