Introduction to the PDC environment (redux)

Radovan Bast

PDC Center for High Performance Computing & Department of Theoretical Chemistry and Biology, KTH Royal Institute of Technology, Stockholm
Outline

- Different types of nodes
- Unix shell (bash)
- File systems, permissions, and transfer
- Parallel computing and parallel programming
- Programming environment
- How to run programs
- Software development tools and good practices
- Performance analysis and debugging
- How to get help

Different types of nodes

- A node is like a computer without keyboard or monitor, and often without disk
- A cluster consists of many nodes
- A node typically contains several processors
- A processor typically contains several cores
- Zoo of nodes:
  - Login nodes
  - Compute nodes
  - Shared interactive nodes
  - Exclusive interactive nodes
  - Transfer nodes
  - Service nodes
- A login node is for submitting jobs, editing files, and compiling small programs
Introducing the unix shell
Introducing the unix shell

- We are greeted with a command-line interface (CLI)
- Teleported back to the 70ies

$ ssh user@milner.pdc.kth.se
Last login: Fri Aug 8 10:14:59 2014 from example.com
user@milner-login1:~> _

- CLI often more efficient than GUI
- High action-to-keystroke ratio (expense: terse and "cryptic")
- Creativity through pipelines
- System is configured with text files
- Calculations are configured and run using text files
- Good for working over network
- Good for reproducibility
- Good for unsupervised workflows
Bash: Files and directories

- Command `pwd` tells me where I am. After login I am in the "home" directory:

  ```bash
  user@machine:~$ pwd
  /afs/pdc.kth.se/home/u/user
  ```

- I can change the directory with `cd`:

  ```bash
  user@machine:~$ cd tmp/talks/
  user@machine:~/tmp/talks$ pwd
  /afs/pdc.kth.se/home/u/user/tmp/talks
  ```

- I can go one level up with `cd ..`

- List the contents with `ls -l`:

  ```bash
  user@machine:~/tmp/talks$ ls -l
  total 237
  drwx------ 3 user csc-users 2048 Aug 17 15:21 img
  -rw------- 1 user csc-users 18084 Aug 17 15:21 pdc-env.html
  -rw------- 1 user csc-users 222051 Aug 17 15:22 remark-latest.min.js
  ```
- Files and directories form a tree
- We can explore the tree with `ls`, and `cd`:

```
user@machine:~:/tmp/talks$ ls -l img/
 total 343
 drwx------ 2 user csc-users 2048 Aug 17 15:21 kerberos
 -rw------- 1 user csc-users 310579 Aug 17 15:21 xc30-blade.png
 -rw------- 1 user csc-users 37812 Aug 17 15:21 xc30-cabinet.jpg
```

- All these commands bring you back to home:

```
$ cd $HOME
$ cd ~
$ cd
$ cd /afs/pdc.kth.se/home/u/user
```
Bash: Creating directories and files

- We create a new directory called "results" and change to it:

  ```
  $ mkdir results
  $ cd results
  ```

Creating and editing files

- Easy but not powerful:

  ```
  $ nano draft.txt
  ```

- More powerful: Emacs or Vi(m):

  ```
  $ emacs draft.txt
  $ vi draft.txt
  $ vim draft.txt # this is Vi "improved"
  ```
Copying, moving, renaming, and deleting

```
# copy file
$ cp draft.txt backup.txt

# recursively copy directory
$ cp -r results backup

# move/rename file
$ mv draft.txt draft_2.txt

# move/rename directory
$ mv results backup

# move directory one level up
$ mv results ..

# remove file
$ rm draft.txt

# remove directory and all its contents
$ rm -r results
```

- From the Unix point of view, deleting is forever!
Bash: History and tab completion

- From the Unix point of view, deleting is forever!

```bash
$ history
1860 vi /home/user/devel/gpunch/src/twoints/EriBlock.cpp
1861 cd ..
1862 git grep GenPrimSSSD
1863 vi src/twoints/EriBlock.h
1864 find . | grep SSSD
1865 vi ./src/twoints/GenPrimSSSD.h
1866 git pull
1867 cd build/
1868 make -j12
```

- Commands are numbered, I can repeat a command by number:

  ```bash
  !1864
  ```

- Use the tab key for tab completion
Bash: Finding things

- Extract lines which contain an expression with `grep`:
  
  ```bash
  # extract all lines that contain "fixme"
  $ grep fixme draft.txt
  ```

- Unix commands have many options/flags - examine them with `man`:
  
  ```bash
  $ man grep
  ```

- Another useful command is `apropos` - try it

- Find files with `find`:
  
  ```bash
  $ find ~ | grep lostfile.txt
  ```

- We can pipe commands and filter results with `|`
  
  ```bash
  $ grep energy results.out | sort | uniq
  ```
Bash: Redirecting output

- Redirect output to a file:

```bash
$ grep energy results.out | sort | uniq > energies.txt
```

- Append output to a file:

```bash
$ grep dipole results.out | sort | uniq >> energies.txt
```

- Print contents of a file to screen:

```bash
$ cat results2.txt
```

- Append contents of a file to another file:

```bash
$ cat results2.txt >> results_all.txt
```
Bash: Writing shell scripts

```
#!/usr/bin/env bash

# here we loop over all files that end with *.out
for file in *.out; do
echo $file
grep energy $file
done
```

- We make the script executable and execute it:

```
# make it executable
$ chmod u+x my_script

# run it
$ ./my_script
```
Bash: Passing arguments to scripts

```
#!/usr/bin/env bash

echo $1
echo $2
echo $3 $2 $1
```

- Arguments are numbered with $1, $2, $3, etc.
- We can now call the script with:

```
$ ./my_script foo bar raboof
```

foo
bar
raboof bar foo
File systems, permissions, and transfer
File systems at PDC

- AFS (Andrew File System)
  - distributed
  - global
  - backup

- Lustre (Linux cluster file system)
  - distributed
  - high-performance
  - no backup
AFS

- Andrew File System
- Named after the Andrew Project (Carnegie Mellon University)
- Distributed file system
- Homogeneous, location-transparent file name space
- Security and scalability
- Accessible "everywhere" (remember that when you make your files readable/writeable!)
- Access via Kerberos tickets and AFS tokens
- Your PDC home directory is located in AFS, example:

  /afs/pdc.kth.se/home/u/user

- OldFiles mountpoint (created by default) contains a snapshot of the files as they were precisely before the last nightly backup was taken.

  /afs/pdc.kth.se/home/u/user/OldFiles

- By default you get a very limited quota (0.5 GB) but you can ask for more
AFS permissions

- You probably know about Unix file permissions (chown, chmod):

```
-rw-r--r-- 1 user csc-users 3153 Feb 20 11:04 intro_pdc.rst
-rw-r--r-- 1 user csc-users 175 Feb 16 20:50 Makefile
```

- AFS permissions work differently:

```
$ fs listacl
Access list for . is
Normal rights:
  user:remote-users rlidwka
  system:anyuser l
  user rlidwka
```

- Google "AFS ACL" to find out how to change permissions.
- Example: give user "alice" read permissions for the current directory:

```
$ fs setacl . alice read
```

- Always remember that AFS is global.
Lustre

- Parallel distributed file system
- Large-scale cluster computing
- High-performance
- /cfs/klemming
- /cfs/milner
- /cfs/zorn
- Unix permissions
- Not global
Overview: PDC storage

- /afs
  - Home
  - Small, quota
  - Backup
  - AFS permissions
  - Not good for temporary job files

- /cfs
  - Large, no quota (but please be considerate and do not fill up the disk)
  - No backup
  - Unix permissions
  - Good for temporary job files
  - Submit jobs from /cfs (except Ferlin, explanation: next slide)
# Overview: mountpoints

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lindgren</td>
<td>/cfs/klemming/nobackup</td>
<td># persistent files</td>
</tr>
<tr>
<td></td>
<td>/cfs/klemming/scratch</td>
<td># temporary files</td>
</tr>
<tr>
<td></td>
<td>/afs</td>
<td># only login node</td>
</tr>
<tr>
<td>Milner</td>
<td>/cfs/milner/scratch</td>
<td># only login node</td>
</tr>
<tr>
<td></td>
<td>/cfs/klemming/nobackup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/cfs/klemming/scratch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/afs</td>
<td></td>
</tr>
<tr>
<td>Ferlin</td>
<td>/scratch</td>
<td># temporary files, local disks</td>
</tr>
<tr>
<td></td>
<td>/afs</td>
<td># persistent files</td>
</tr>
<tr>
<td>Povel</td>
<td>/cfs/klemming/nobackup</td>
<td># persistent files</td>
</tr>
<tr>
<td></td>
<td>/cfs/klemming/scratch</td>
<td># temporary files</td>
</tr>
<tr>
<td></td>
<td>/afs</td>
<td># important files, backup</td>
</tr>
<tr>
<td>Zorn</td>
<td>/cfs/zorn/nobackup</td>
<td># persistent files</td>
</tr>
<tr>
<td></td>
<td>/afs</td>
<td># important files, backup</td>
</tr>
<tr>
<td>Ellen</td>
<td>/cfs/klemming/nobackup</td>
<td># persistent files</td>
</tr>
<tr>
<td></td>
<td>/cfs/klemming/scratch</td>
<td># temporary files</td>
</tr>
<tr>
<td></td>
<td>/afs</td>
<td># important files, backup</td>
</tr>
</tbody>
</table>
File transfer between PDC machines

- `/afs` is mounted and visible on all machines (at least login node)
- No need to "transfer" files which are on `/afs`
- You can share large files between machines via `/cfs` (except Ferlin)

PDC machines and outside world (Linux and Mac OS X)

```bash
# from my laptop to Lindgren
$ scp myfile username@lindgren.pdc.kth.se:~/Private

# from Lindgren to my laptop
$ scp username@lindgren.pdc.kth.se:~/Private/myfile .

# from Lindgren to Triolith (NSC)
$ ssh user@lindgren.pdc.kth.se
$ scp Private/myfile user@triolith.nsc.liu.se:
```

- If the username is the same on source and destination machine, you can leave it out
Host aliases

- If you are tired of typing long hostnames, create aliases in `~/.ssh/config`

```markdown
Host trio
  Hostname triolith.nsc.liu.se
  ForwardX11 yes
  User eminem
```

PDC machines and outside world (Windows)

- PuTTY (only PuTTY downloaded from this link works at PDC)
  https://www.pdc.kth.se/resources/software/login-1/windows/putty

```bash
# have not tested this myself
C:\Program Files\PuTTY\pscp.exe myfile username@lindgren.pdc.kth.se:~/Private
```
Alternative to scp

- Install AFS client and copy directly
- Then AFS is mounted just like another disk on your computer

Transferring large (or very many) files to /cfs

- In this case do not scp to the login node
- Transfer files to the transfer node

# from my laptop to Lindgren
$ scp gigantic.tar.gz username@cfs-aux-4.pdc.kth.se:/cfs/klemming/nobackup/u/user
Parallel computing and parallel programming
Parallel computing

- Typical example

```java
double r = 0.0;

// n is reasonably large
for (int i = 0; i < n; i++)
{
    // this routine does most of the work
    r += compute_something(...);
}
```

- We would try to split the work (loop) among cores/threads/workers/tasks
- Each thread should either do something different or do the same thing on different data
- We need to think about synchronization
- We need to think about load balance
Typical parallel programming standards

- **MPI** (message passing interface)
  - C, C++, Fortran, R, Python
  - Distributed memory (mostly)
  - Explicit
- **OpenMP** (CPU, GPU)
  - C, C++, Fortran
  - Shared memory
  - Implicit
- **CUDA** (GPU)
  - NVIDIA
  - C, C++, Fortran
- **OpenACC** (GPU and accelerators)
  - NVIDIA, Cray, PGI
- **OpenCL** (GPU, CPU, and accelerators)
  - AMD, Intel
- ...
Parallel performance: Ideal vs. real

- It is **very important** to check that increase in number of processors is worth it
- In your interest (user/developer) and in our interest (computing center)
Parallel performance bottlenecks

- Synchronization overhead
- False sharing
- Load imbalance
- Amdahl law
- Memory bandwidth
- File I/O
Programming environment
Working with modules

- At PDC we use the module environment.
- Modules are used to load specific software into your environment.

```bash
# list all available modules
$ module avail

# show information about a module
$ module show fftw/3.3.0.4

# load a module
$ module load fftw/3.3.0.4

# list currently loaded modules
$ module list

# swap modules
$ module swap PrgEnv-cray PrgEnv-intel

# unload module
$ module unload fftw/3.3.0.4
```
$ module list # on Milner

Currently Loaded Modulefiles:
1) modules/3.2.6.7
2) nodestat/2.2-1.0501.47138.1.78.ari
3) sdb/1.0-1.0501.48084.4.48.ari
4) alps/5.1.1-2.0501.8471.1.1.ari
5) MySQL/5.0.64-1.0000.7096.23.2
6) lustre-cray_ari_s/2.4.3.0.80_0.5.1_1.0501.7664.12.1-1.0501.14255.11.3
7) udreg/2.3.2-1.0501.7914.1.13.ari
8) ugni/5.0-1.0501.8253.10.22.ari
9) gni-headers/3.0-1.0501.8317.12.1.ari
10) dmapp/7.0.1-1.0501.8315.8.4.ari
11) xpmem/0.1-2.0501.48424.3.3.ari
12) hss-llm/7.1.0
13) Base-opts/1.0.2-1.0501.47945.4.2.ari
14) craype-network-aries
15) craype/2.04
16) cce/8.2.3
17) cray-libscl/12.1.3
18) pmi/5.0.1-1.0000.9799.94.6.ari
19) rca/1.0.0-2.0501.48090.7.46.ari
20) atp/1.7.1
21) PrgEnv-cray/5.1.29
22) craype-ivybridge
23) cray-mpi/6.2.1
24) slurm
25) openssh/5.3p1-with-pam-gsskex-20100124
26) openafs/1.6.6-3.0.80-0.5.1_1.0501.7664-cray_ari_s
27) heimdal/1.5.2
28) snic-env/1.0.0
- Modules take care of setting proper environment variables.
- The module system does nothing which you could not do by setting environment variables by hand, but you probably want the help of the modules system.

```
$ module show fftw/3.3.0.4
[/opt/cray/modulefiles/fftw/3.3.0.4:
  setenv     FFTW_VERSION 3.3.0.4
  setenv     CRAY_FFTW_VERSION 3.3.0.4
  setenv     FFTW_DIR /opt/fftw/3.3.0.4/sandybridge/lib
  setenv     FFTW_INC /opt/fftw/3.3.0.4/sandybridge/include
  prepend-path  PATH /opt/fftw/3.3.0.4/sandybridge/bin
  prepend-path  MANPATH /opt/fftw/3.3.0.4/share/man
  prepend-path  CRAY_LD_LIBRARY_PATH /opt/fftw/3.3.0.4/sandybridge/lib
  setenv     PE_FFTW_REQUIRED_PRODUCTS PE_MPI
  prepend-path  PE_PKGCONFIG_PRODUCTS PE_FFTW
  setenv     PE_FFTW_TARGET_sandybridge sandybridge
  setenv     PE_FFTW_TARGET_x86_64 x86_64
  setenv     PE_FFTW_TARGET_interlagos interlagos
...```
Compiling code on Cray

- Available compiler sets: Cray, GNU, Intel, and PGI (Lindgren)
- By default, the PGI (Lindgren) or Cray (Milner) compiler set is loaded
- Use `module swap` to switch between these

```
# select Intel
$ module swap PrgEnv-cray PrgEnv-intel
```

- Module `cray-libsci` provides BLAS, LAPACK, BLACS, and SCALAPACK
- Module `cray-mpich` provides MPI
- On Cray we compile using compiler wrappers: ftn, cc, and CC
Using compiler wrappers on Cray

- Fortran

  ```
  $ gfortran [flags] source.F90 # wrong
  $ ifort [flags] source.F90 # wrong
  # correct
  $ ftn [flags] source.F90
  ```

- C

  ```
  $ gcc [flags] source.c # wrong
  $ icc [flags] source.c # wrong
  # correct
  $ cc [flags] source.c
  ```

- C++

  ```
  $ g++ [flags] source.cpp # wrong
  $ icpc [flags] source.cpp # wrong
  # correct
  $ CC [flags] source.cpp
  ```

- Same wrappers for MPI and sequential code
Compiling OpenMP code on Cray

- Intel

```bash
$ ftn -openmp source.F90
$ cc -openmp source.c
$ CC -openmp source.cpp
```

- Cray

```bash
$ ftn -h omp source.F90
$ cc -h omp source.c
$ CC -h omp source.cpp
```

- GNU

```bash
$ ftn -fopenmp source.F90
$ cc -fopenmp source.c
$ CC -fopenmp source.cpp
```
How about math libraries?

- Cray will automatically link to BLAS, LAPACK, BLACS, and SCALAPACK
- No need to worry about that
Compiling code on Ferlin/Povel

- Intel

```bash
$ ifort -openmp source.F90
$ icc -openmp source.c
$ icpc -openmp source.cpp
```

- PGI

```bash
$ pgf90 -mp source.F90
$ pgcc -mp source.c
$ pgcpp -mp source.cpp
```

- GNU

```bash
$ gfortran -fopenmp source.F90
$ gcc -fopenmp source.c
$ g++ -fopenmp source.cpp
```
Compiling code on Ferlin/Povel

- MPI (OpenMPI or MPICH)

```bash
$ mpiif90 source.F90
$ mpicc source.c
$ mpicxx source.cpp
```

- Intel MPI

```bash
$ mpiifort source.F90
$ mpiicc source.c
$ mpiicpc source.cpp
```

General recommendation (any machine, any vendor)

- Study the effect of optimization flags
Advice for portability: protect MPI code with the pre-processor

program hello
  implicit none

#ifndef ENABLE_MPI
  include 'mpif.h'
#endif

  integer :: irank, num_proc, ierr, tag

#ifndef ENABLE_MPI
  integer :: status(MPI_STATUS_SIZE)
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, irank, ierr)
#else
  irank = 0
  num_proc = 1
#endif

print *, 'rank ', irank, 'out of', num_proc, 'proc'

#ifndef ENABLE_MPI
  call MPI_FINALIZE(ierr)
#endif

end program

$ ftn -DENABLE_MPI source.F90
Compiling CUDA code on Zorn

- Zorn uses the module system as well

```bash
# check available modules
$ module avail

# load the CUDA kit
$ module load cuda/5.5

# compile the source
$ nvcc mysource.cc
```

- No compiler wrappers for Fortran/C/C++. Sources are compiled in the "traditional" way.
How to run programs
How to run programs

- After login we are on the login node
- A login node is for submitting jobs, editing files, and compiling small programs
- **We never run calculations interactively on the login node**
- We want balanced load of the resources
- Fair share according to time allocations
- Rather we use a queuing/batch system

- Only persons/groups belonging to a Charge Account Category (CAC; in other words time allocation) can submit
Queuing systems at PDC

- Lindgren
  - PBS
    - Can only submit a batch script
- Zorn
  - PBS
    - Can only submit a batch script
- Ferlin/Povel
  - EASY
    - Submit a batch script or book interactive node
- Milner
  - Slurm
    - Submit a batch script or book interactive node
- Ellen
  - Only interactive
MPI launchers

- Lindgren and Milner
  - aprun

```
aprun -n 48 ./my_exe > my_output
```

- Other machines
  - mpirun or mpiexec

```
mpirun -np 48 ./my_exe > my_output
```
PBS (Lindgren and Zorn)

# submit the job
$ qsub script.pbs

# see information about all your jobs
$ qstat -u $USER

# remove or stop a job
$ qdel [jobid]

Slurm (Milner)

# submit the job
$ sbatch script.slurm

# see information about all your jobs
$ squeue -u $USER

# remove or stop a job
$ scancel [jobid]
Example PBS job script

#!/bin/bash --login

# name of the job
#PBS -N my_job

# wall-clock time given to this job (20 minutes)
#PBS -l walltime=00:20:00

# must be a multiple of 24
#PBS -l mppwidth=48

#PBS -o stdout.txt
#PBS -e stderr.txt

# run the executable named my_exe
cd $PBS_O_WORKDIR
aprun -n 48 ./my_exe > my_output
Example Slurm job script

```bash
#!/bin/bash --login

# name of the job
#SBATCH -J my_job

# wall-clock time given to this job (20 minutes)
#SBATCH -t 00:20:00

# number of nodes
#SBATCH -N 2

# number of MPI processes per node (the following is actually the default)
#SBATCH --ntasks-per-node=40

# number of MPI processes
#SBATCH -n 80

#SBATCH -o stdout.txt
#SBATCH -e stderr.txt

# run the executable named my_exe
# and write the output to my_output

cd $SLURM_SUBMIT_DIR
aprun -n 80 ./my_exe > my_output 2>&1
```

- In this case the job script is a Bash script but it can be Python or Perl instead
Running with or without hyperthreading

- Hyperthreading is on by default on Milner
- System sees 40 "logical" cores per node but only 20 physical cores are present
- You can turn hyperthreading off
- This is important if you study performance

```bash
# no hyperthreading
# a node appears as 20 cores
$ aprun -j 1 [other flags] [program]

# hyperthreading enabled (this is the default)
# a node appears as 40 cores
$ aprun -j 2 [other flags] [program]
```
OpenMP usage

- With hyperthreading

```bash
#!/bin/bash --login

#SBATCH -J my_job
#SBATCH -t 00:20:00
#SBATCH -N 1
#SBATCH -n 40
#SBATCH -o stdout.txt
#SBATCH -e stderr.txt

export OMP_NUM_THREADS=40

cd $SLURM_SUBMIT_DIR
aprun -j 2 -n 1 -N 1 -d $OMP_NUM_THREADS ./my_exe > my_output
```
OpenMP usage

- Without hyperthreading

```bash
#!/bin/bash --login

#SBATCH -J my_job
#SBATCH -t 00:20:00
#SBATCH -N 1
#SBATCH -n 20
#SBATCH -o stdout.txt
#SBATCH -e stderr.txt

export OMP_NUM_THREADS=20

cd $SLURM_SUBMIT_DIR
aprun -j 1 -n 1 -N 1 -d $OMP_NUM_THREADS ./my_exe > my_output
```
Booking an interactive node (Milner)

- You can request an interactive node with `salloc`:

  ```
  $ alloc
  alloc: Granted job allocation 21223
  ```

- Then you get a node for interactive use.
- You can log out with `exit`:

  ```
  $ exit
  alloc: Relinquishing job allocation 21223
  alloc: Job allocation 21223 has been revoked.
  ```
Important note

On Lindgren submit all jobs from /cfs

# submit from here
/cfs/klemming/nobackup/u/user

and not from your home directory on /afs

# not here
/afs/pdc.kth.se/home/u/user
Software development tools and good practices
Software development tools and good practices

- Version control (Git or Hg)
- Unit tests, regression tests, test-driven development
- Cross-platform configuration tool: CMake
- Code documentation with Sphinx
- Profile before you optimize anything
- Think about the cache
Building portable software with CMake

- Write one `CMakeLists.txt` file which controls configuration

```cmake
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
project(ExcitingProject)
enable_language(Fortran)

# this is where we will place the Fortran module files
set(CMAKE_Fortran_MODULE_DIRECTORY ${PROJECT_BINARY_DIR}/modules)

# the executable is built from 3 source files
add_executable(
    exciting_project.x
    main.F90
    foo.F90
    bar.F90
)
```

- CMake generates a Makefile for you
Code documentation with Sphinx

- Write lightweight RST markup
- Generate html or LaTeX
- Host on https://readthedocs.org

Continuous Integration with Travis

- Host code on GitHub
- Write tests using Google Test or pytest or nose or your favourite testing framework
- Deploy testing on https://travis-ci.org
Performance analysis and debugging
Performance analysis

- CrayPAT
- Allinea Performance Reports
- Vampir
- Paraver

Debugging

- DDT
- TotalView
- Cray ATP
- gdb
<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb. Samp</th>
<th>Imb. Samp%</th>
<th>Group Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>6.0</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>87.5%</td>
<td>5.2</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>72.9%</td>
<td>4.4</td>
<td>0.6</td>
<td>14.3%</td>
<td>MPI_SENDRECV</td>
</tr>
<tr>
<td>14.6%</td>
<td>0.9</td>
<td>0.1</td>
<td>14.3%</td>
<td>mpi_finalize</td>
</tr>
<tr>
<td>10.4%</td>
<td>0.6</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>4.2%</td>
<td>0.2</td>
<td>1.8</td>
<td>100.0%</td>
<td>png_write_chunk</td>
</tr>
<tr>
<td>2.1%</td>
<td>0.1</td>
<td>0.9</td>
<td>100.0%</td>
<td>inflate</td>
</tr>
<tr>
<td>2.1%</td>
<td>0.1</td>
<td>0.9</td>
<td>100.0%</td>
<td>deflateSetDictionary</td>
</tr>
<tr>
<td>2.1%</td>
<td>0.1</td>
<td>0.9</td>
<td>100.0%</td>
<td>deflateEnd</td>
</tr>
<tr>
<td>2.1%</td>
<td>0.1</td>
<td>0.9</td>
<td>100.0%</td>
<td>IO</td>
</tr>
<tr>
<td>2.1%</td>
<td>0.1</td>
<td>0.9</td>
<td>100.0%</td>
<td>write</td>
</tr>
</tbody>
</table>

- [http://docs.cray.com/books/S-2315-52/html-S-2315-52/z1055157958smg.html](http://docs.cray.com/books/S-2315-52/html-S-2315-52/z1055157958smg.html)
Allinea Performance Reports

Summary: wave-apr.x is CPU-bound in this configuration

The total wallclock time was spent as follows:

**CPU** 94.0%  
Time spent running application code. High values are usually good.  
This is **very high**; check the CPU performance section for optimization advice.

**MPI** 6.0%  
Time spent in MPI calls. High values are usually bad.  
This is **very low**; this code may benefit from increasing the process count.

**I/O** 0.0%  
Time spent in filesystem I/O. High values are usually bad.  
This is **negligible**; there's no need to investigate I/O performance.

This application run was CPU-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As very little time is spent in MPI calls, this code may also benefit from running at larger scales.

**CPU**
A breakdown of how the 94.0% total CPU time was spent:

- Scalar numeric ops 23.3%
- Vector numeric ops 0.0%
- Memory accesses 73.5%
- Other 3.1%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

No time is spent in **vectorized instructions**. Check the compiler’s vectorization advice to see why key loops could not be vectorized.

**MPI**
Of the 6.0% total time spent in MPI calls:

- Time in collective calls 3.5%
- Time in point-to-point calls 96.5%
- Effective process collective rate 38.0 kB/s
- Effective process point-to-point rate 1.42 MB/s

Most of the time is spent in point-to-point calls with a very low transfer rate. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate further.

**I/O**
A breakdown of how the 0.0% total I/O time was spent:

- Time in reads 0.0%
- Time in writes 0.0%
- Effective process read rate 0.00 bytes/s
- Effective process write rate 0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

**Memory**
Per-process memory usage may also affect scaling:

- Mean process memory usage 6.92 MB
- Peak process memory usage 8.79 MB
- Peak node memory usage 5.7%

The peak node memory usage is very low. You may be able to reduce the amount of allocation time used by running with fewer MPI processes and more data on each process.
Vampir

- Example image courtesy Magnus Helmersson
- https://www.vampir.eu
Paraver

- Example image courtesy Xavi Aguilar
How to get help
PDC support

- Many questions can be answered by reading the web documentation: https://www.pdc.kth.se/support
- Preferably contact PDC support by email: support@pdc.kth.se
- Or by phone: +46 (0)8 790 7800
- You can also make an appointment to come and visit.
- Your email support request will be tracked - you get a ticket number.
- For follow-ups/replies always include the ticket number - they look like this: [SNIC support #12345]
How to report problems

- Do not report new problems by replying to old/unrelated tickets.
- Split unrelated problems into separate email requests.
- Use a descriptive subject in your email (unhelpful subject line: "problem").
- Give your PDC user name.
- Be as specific as possible.
- For problems with scripts/jobs, give an example. Either send the example or make it accessible to PDC support.
- Make the problem example as small/short as possible.
- Provide all necessary information to reproduce the problem.
- If you want the PDC support to inspect some files, make sure that the files are readable.
- Do not assume that PDC support personnel have admin rights to see all your files or change permissions.
References