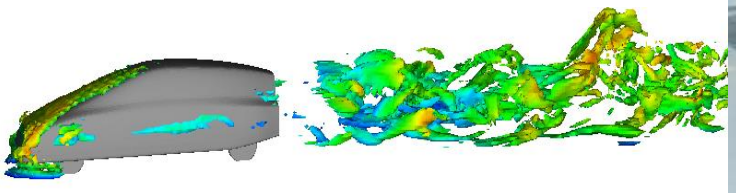
Programming Techniques for Supercomputers



HPC Services @ RRZE

University Erlangen-Nürnberg

Sommersemester 2019

Outline

- **Login to RRZE's Emmy cluster**
- **Basic environment**
- **Some guidelines**
- **First Assignment**

Login

- **Login to terminals with your CIP (CS) account**
  - Obtainable from the CIP admins from 12:00 to 13:00
  - CIP admin whereabouts in front of CIP pools
  - Choose your preferred desktop, e.g. KDE
- **Start any kind of shell, e.g. `konsole`**
  (All workstations in the tutorial room should run linux)
- **Login to RRZE cluster front-end machines**
  - **`ssh ptfsXXXh@emmy.rrze.uni-erlangen.de`**
  - **`-X`** *enables X forwarding*
  - Front-end nodes: **`emmy1`**, **`emmy2`**

```
$ ssh -X ptfsXXXh@emmy.rrze.uni-erlangen.de
Last login: Tue Apr 19 16:36:04 2016 from fau-eduroam-1-34.wlan.rrze.net
================================================================================
Welcome at "Emmy", RRZE's IvyBridge/NEC cluster!
[...BLURB...]
================================================================================
ptfsXXXh@emmy2:~$
```

Compiling

- **Make compiler available for use:**
  - `module load intel64`
  - `icc` → Intel C compiler
  - `icpc` → Intel C++ compiler
  - `ifort` → Intel Fortran compiler

- **Recommended Intel compiler options**
  - `-O3` high optimization level
  - `-xHost` optimize for CPU the compiler is running on
  - `-fno-alias` assume no overlap between any arrays or elements
  - Other options (`-c`, `-o`, `-S`, `-L`, `-I`, `-l`,…) are the same as for GCC

- **Additional software**
  - `module available` → overview over all available software
  - `module list` → currently loaded modules
  - `module unload <modulename>` → unload module

Acquiring a cluster node

- **Issue an interactive job**
  - `qsub -l nodes=1:ppn=40,walltime=01:00:00 -I`
  - Requests all 40 virtual cores of one node for one hour
  - `-I` requests an interactive login shell on the compute node
  - Always request a full node (ppn=40)!
- **The node is yours alone for the allocated time**
  - Compiling is possible on the cluster nodes, modules work

```
ptfsXXXh@emmy2:~$ qsub -l nodes=1:ppn=40,walltime=01:00:00 -I
[... BLURB ...]
qsub: waiting for job 585626.eadm to start
qsub: job 585626.eadm ready

Starting prologue... Tue Apr 19 16:29:11 CEST 2016
Master node: e0104
Kill all process from other users
Adjust oom killer config
Clearing buffers and caches on the nodes.
Power management available, enabling ondemand governor
End of prologue: Tue Apr 19 16:29:18 CEST 2016
ptfsXXXh@e0104:~$
```

Fixing the clock frequency

- **Modern CPUs can adjust their own clock speed depending on some conditions (“Turbo Mode” etc.)**
  - # of active cores
  - Temperature
  - ???
- **Accurate and reproducible benchmarking requires a constant clock speed**
- **Emmy allows you to set the clock speed at job submission**

```
ptfsXXXh@emmy2:~$ qsub -l nodes=1:ppn=40:f2.2,walltime=01:00:00 -I
[...]

Starting prologue... Tue Apr 19 16:29:11 CEST 2016
Master node: e0104
Kill all process from other users
Adjust oom killer config
Clearing buffers and caches on the nodes.
Power management available, setting specified frequency of 2200000 using
performance governor
End of prologue: Tue Apr 19 16:29:18 CEST 2016
ptfsXXXh@e0104:~$
```

Job scripts

- **Interactive jobs are inadequate for “production runs”**
  - Parameter studies
  - Long runs

- **Substitute `-I` by the name of a shell script:**

```
$ qsub -l nodes=1:ppn=40:f2.2,walltime=01:00:00 script.sh
```

```
#!/bin/bash
#
# the script runs in $HOME, so
# change to correct directory (i.e. the directory
# from which the job was submitted)
cd $PBS_O_WORKDIR
# start executable
./a.out
```

Measuring time

- **Remember:** Performance $P = \frac{W}{T_{wall}}$

  W = work

  T_{wall} = “wallclock time”, elapsed time
- **Accurate time measurement is important!**
  - Very short periods difficult to measure
  - Measure at least for 100 ms
- **Example code in `~unrz55/GettingStarted`**

```
#include <timing.h>
int main(int argc, char *argv[]) {
  double wcTimeStart,wcTimeEnd,cpuTimeStart,cpuTimeEnd,wcTime;

  timing(&wcTimeStart,&cpuTimeStart);
    /* PUT YOUR CODE HERE */
  timing(&wcTimeEnd,&cpuTimeEnd);

  wcTime = wcTimeEnd-wcTimeStart;
  printf("Walltime: %.3lf s\n",wcTime);
  return 0;
}
```

Where to work if not at the university?

Best practice:

- **Login to `cshpc.rrze.uni-erlangen.de`**
  - All necessary tools installed
  - Access to all hpc systems and most filesystems
  - Linux Desktop from Windows: Nomachine
    https://www.anleitungen.rrze.fau.de/hpc/dialogserver/
  - Available from outside RRZE/FAU
  - Compile on Emmy frontends

General guidelines

- Do not run benchmarks on the frontend nodes, as multiple programs and users interact there
- You may do test runs, e.g., compilation tests and verification on frontends, but also on the compute nodes (via an interactive job)
- For obtaining lots of results, write your own scripts and execute them via the batch system
- Check your results for plausibility (Cool! My code runs @ petaflop/s!)

Some links

- Getting Started Guide for your work with the cluster:
  - https://moodle.rrze.uni-erlangen.de/mod/page/view.php?id=7276
- Linux tutorial for n00bs:
  - https://ryanstutorials.net/linuxtutorial/
- Intel processor details (just one example):
  - https://en.wikipedia.org/wiki/Ivy_Bridge_(microarchitecture)
- Confused about all those CPU code names?
  - https://en.wikipedia.org/wiki/Intel_Core
- Emmy cluster details:
  - https://www.anleitungen.rrze.fau.de/hpc/emmy-cluster/

- Blogs by RRZE HPC staff:
  - https://www.blogs.fau.de/hager/
  - https://www.blogs.fau.de/zeiser/

First Assignment

- https://moodle.rrze.uni-erlangen.de/mod/assign/view.php?id=7277

- Hand-in via moodle is NOT mandatory! But we will offer feedback.
- Hand-in before Monday to be included in the discussion.