Exercises for Architectures of Supercomputers

1st Exercise, 23./24.10.2019

Johannes Hofmann

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Agenda

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- General information about exercises

- Introduction to the Erlangen Regional Computing Center's (RRZE) cluster environment
  - Accounts
  - HPC resources
  - Overview
  - Working with cluster resources

- Reminder: Linux command-line basics

- This week's (short) exercise

General information about exercises

- Exercises are **not** mandatory but **part of the exam**
- You can work on the assignments in teams of two
- Typical exercise schedule
  - Solution to last week's exercise is presented
  - New assignment for the next week is given
  - Remaining time: I'm here to provide help and go around to look at your solutions to last week's exercise

Agenda

- General information about exercises
- **Introduction to the Erlangen Regional Computing Center's (RRZE) cluster environment**
  - Accounts
  - HPC resources
  - Overview
  - Working with cluster resources
- Reminder: Linux command-line basics
- This week's (short) exercise

Accounts

- All students enrolled in the StudOn course by 1pm on Oct 21 have received an HPC entitlement package in IdM (`https://idm.fau.de`)
  - If you missed the deadline, chose a group partner that has an account
- Validate you received an account and write down your username
  - Log in at http://idm.fau.de using your IdM credentials
  - Go to Profile → Data overview
  - Under Special services → Service package (HPC)

  - Usernames: `hpcv175h – hpcv212h`
  - By default, your password for the account is the same as your IdM password
    - You can change it by clicking on the three dots next to the server package. (it might take a few minutes for the new password to become active)

HPC Resources

For the exercise we will use RRZE's "Emmy" and "Meggie" clusters

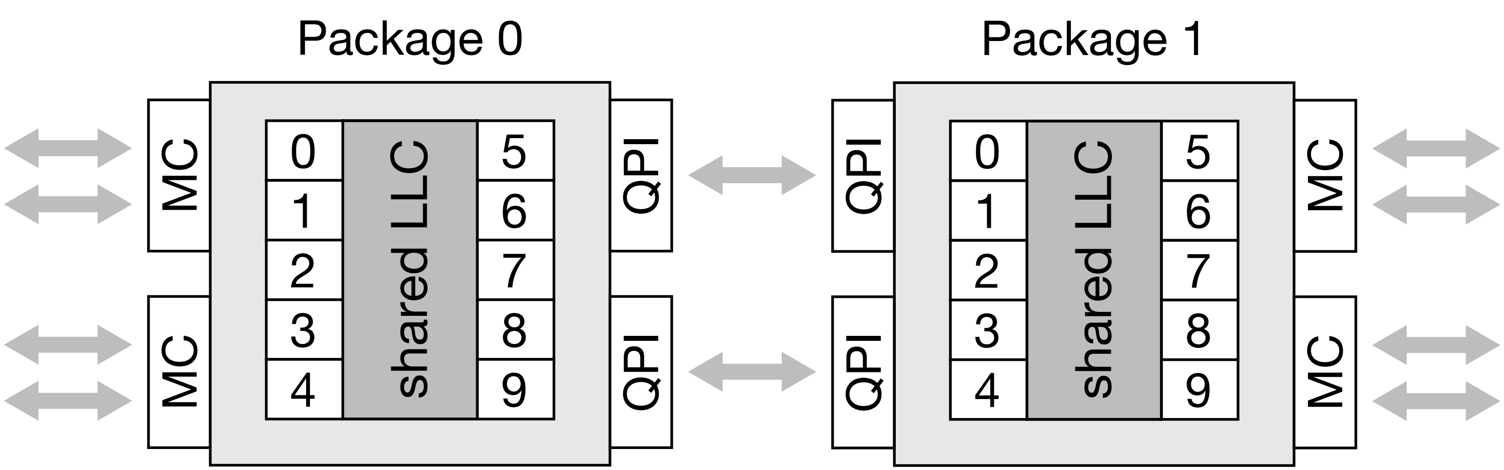
Emmy

- 560 compute nodes
  - 544 regular nodes
  - 16 accelerator nodes (Nvidia K20x GPUs, Intel Xeon Phi coprocessors)
- Dual-socket (i.e., dual processor) Ivy Bridge-EP nodes
- 10 cores per processor, clocked at 2.2 GHz (nominal clock rate)
- 64 GB DDR3-1600 (regular nodes) / DDR3-1866 (accelerator nodes) RAM
- More information available online at https://www.anleitungen.rrze.fau.de/hpc/emmy-cluster/

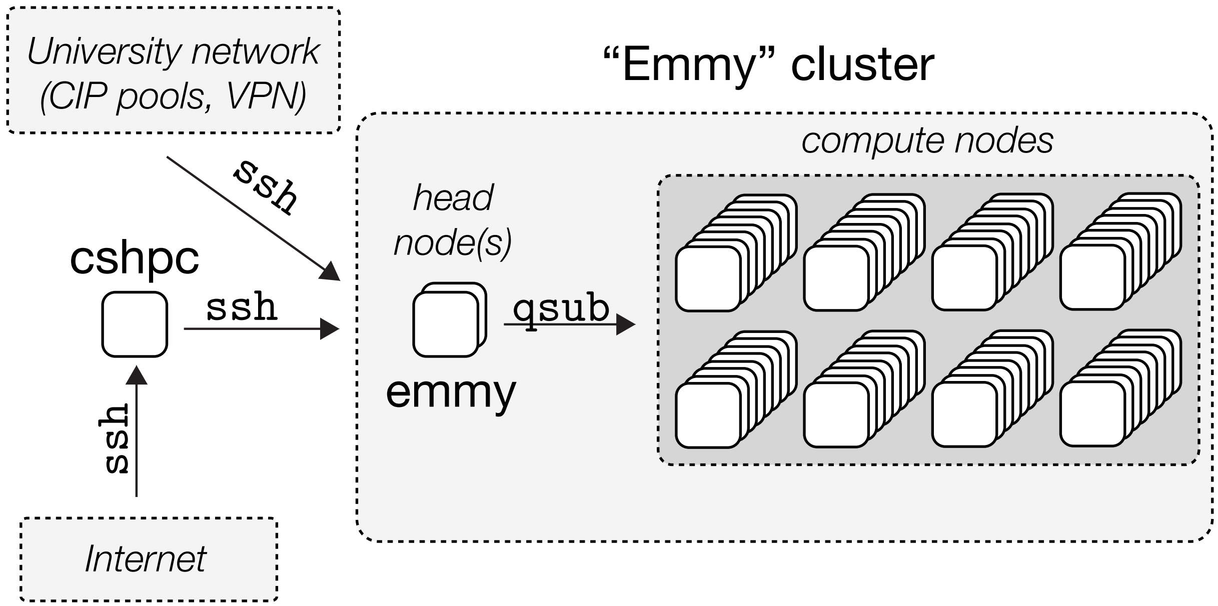
Layout of a single „Emmy“ compute node

Dual-Socket Ivy Bridge-EP nodes

- 2x Xeon E5-2660 v2 **chips** (**CPUs**, **processors, packages**)
- 10 **cores**, 20 **SMT-threads** (**logical cores**) per chip
- CPU clock: 2.2 GHz (nominal), 2.6 GHz (Turbo mode)
- 64GB DDR3-**memory** (**RAM**) clocked at 1.6 GHz

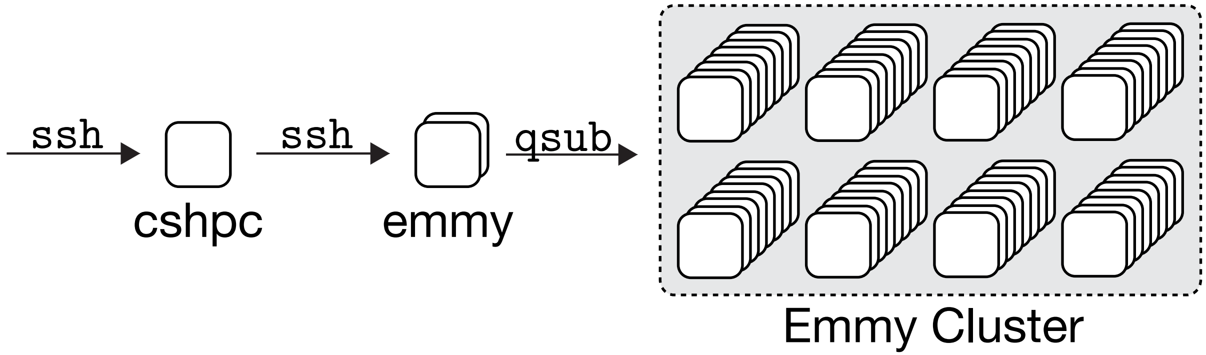


Overview



- Access from inside the university‘s network via head node (`emmy.rrze.fau.de`)
  - Compute nodes are allocated from the head node
- Access from outside the university‘s network via indirection (`cshpc.rrze.fau.de`)

Working with cluster resources



- Try to log in to the head node
    - (`ssh your_HPC_account_name@cshpc.rrze.fau.de`)
    - `ssh your_HPC_account_name@emmy.rrze.fau.de`
- Allocate one compute node for two hours
    - `qsub -lnodes=1:ppn=40:f2.2:ddr1600,walltime=1:30:00 -I`
        - `nodes=1` Number of nodes to request; one in this instance
        - `ppn=40` allocate the full node (2 CPUs/node x 10 cores/CPU x 2-SMT)
        - `f2.2` Fix CPU frequency to 2.2 GHz and do not use Turbo mode
        - `ddr1600` Use regular instead of accelerator node
        - `walltime` The amount of time you want to allocate a node
        - `-I` Interactive login

Module system

- Some software (e.g. compiler) must be explicitly added to the `$PATH`

```
emmy1:~ iwi325$ module avail gcc intel64
----------- /apps/modules/data/development -----------
gcc/4.9.3 gcc/5.4.0 gcc/6.1.0
----------- /apps/modules/data/deprecated -----------
gcc/4.8.1 gcc/4.9.2 gcc/5.3.0 -----------
-----------  /apps/modules/data/development -----------
intel64/12.1up13          intel64/14.0up03
intel64/16.0up03(default) intel64/13.1up03
[...]
```

- You should be using Intel Compiler 19.0.4 for the exercise

`module load` **`intel64/19.0up04`**

Agenda

- General information about exercises

- Introduction to the Erlangen Regional Computing Center's (RRZE) cluster environment
  - Accounts
  - HPC resources
  - Overview
  - Working with cluster resources

- **Reminder: Linux command-line basics**

- This week's (short) exercise

Linux basics: Head node and directories

- Log in to the cluster head node using the secure shell (`ssh`) and X forwarding
  - Open a terminal in your window manager
    `$ ssh -X <your_login_name>@emmy.rrze.fau.de`
    - When logging into a node via `ssh`, your default working directory is your `$HOME` directory (e.g., `/home/hpc/<username>`)

- Create working directory
  - Continue using the terminal you used to log into the head node
  - Create a directory for the CAMA exercise in your `$HOME`; e.g.,
    `$ mkdir archsup_ex_01`
  - Change your current working directory to the newly created one:
    `$ cd archsup_ex_01`

Linux basics: Editing files and compiling code

- Editing your source code
  - Skilled users may use terminal versions of their favorite text editor (e.g., `vim`, `emacs`, etc.)
  - If you require a graphical text editor, make sure to enable X forwarding when establishing your ssh connection using the `-X` command-line parameter, and use, e.g., the `emacs` GUI editor

- Compile code
  - Make sure to load the Intel C compiler from the module system
    ```
    $ module load intel64/19.0up04
    ```
  - Skilled users may create a `Makefile` and use the `make` command
  - You can compile your code manually using the `icc` command:
    ```
    $ icc my_code_main.c my_code_more.c [...] -o my_binary
    ```

Linux basics: Cluster jobs and running your code

- Note: Do **not** run the benchmark on the head node!
  - Contention, reproducibility, etc.

- Submit an interactive job to allocate a node
  - On the head node, use the `qsub` command to request a node
    `$ qsub -lnodes=1:ppn=40:f2.2:ddr1600,walltime=1:30:00 -I`

- Run the benchmark
  - If necessary, change to the correct directory
    `$ cd archsup_ex_01`
  - Run your binary
    `$ ./my_binary`

Linux basics: File systems and copying data

- Note that your “HPC homes” on the clusters are not the NFS-exported “CIP-pool” homes. However, your HPC homes are NFS-exported across the entire HPC domain
  - This means that homes are shared across the head and compute nodes of all RRZE clusters

- Copying files between your local workstation and the HPC domain
  - Copying local files to the HPC home:
    `$ scp path/to/local/file <your_login_name>@emmy.rrze.fau.de:`
  - Getting remote files:
    ```
    $ scp <your_login_name>@emmy.rrze.fau.de:path/to/remote/file \
          path/to/local/file
    ```
  - To copy whole directories use: `scp -r`
  - Make sure to use cshpc when working outside the university network

Agenda

- General information about exercises

- Introduction to the Erlangen Regional Computing Center’s (RRZE) cluster environment
  - Accounts
  - HPC resources
  - Overview
  - Working with cluster resources

- Reminder: Linux command-line basics

- **This week’s (short) exercise**

1st Exercise: Benchmarking skeleton

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- Your first program will measure the memory bandwidth (in GB/s) of a function that initializes a dynamically allocated array of double-precision floating-point numbers with zero
  1. The size of the array (in bytes) is passed as a command-line parameter to your program
     - E.g., `./my_program 1000000` should result in a one-megabyte array
     - Those unfamiliar with the C programming language should start here: https://ece.uwaterloo.ca/~dwharder/icsrts/C/05/
  2. Initializing the dynamically allocated memory
     - Use a `for`-loop to traverse and initialize the array
       - Make sure to put the following statement in the line before the `for`-loop: `#pragma novector`
     - Make sure to put the code to initialize the array into a **separate** function
     - Make sure to put the function to initialize the array into a **separate** file
     - Those unfamiliar with the C programming language should begin by familiarizing themselves with the man-pages system
       - E.g., typing `$ man malloc` will display the man page for the `malloc` function, which is used to dynamically allocate memory

1st Exercise: Benchmarking skeleton

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- Your first program will measure the memory bandwidth (in GB/s) of a function that initializes a dynamically allocated array of double-precision floating-point numbers with zero

   3. Measure the runtime with the help of the `get_time` function provided in StudOn together with the exercise slides
      - Measure the current time (in seconds) **before** calling the function that initializes the memory
      - Measure the current time (in seconds) **after** calling the function that initializes the memory
      - Calculate the attained bandwidth by dividing the amount of data transferred during the initialization (in B) by the amount of time it took to initialize the data (in seconds)
   4. Output measured bandwidth
      - Convert from B/s to GB/s
      - Output result

1st Exercise: Benchmarking skeleton

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- After your code is implemented
  1. Measure the attained bandwidth for a one-megabyte and a one-gigabyte array. What do you notice? Do you have an explanation for what you observe?
  2. Instead of compiling and linking your function to initialize the memory into an executable, convert it into human-readable assembly and examine the macro instructions. Can you establish a connection between your high-level C code and the assembly/macro instructions?

```
$ icc -c —S —masm=intel file_that_contains_init_function.c
```

- `-c` Only create object file, do not link binary
- `-S` Create human-readable assembly
- `-masm=intel` Use Intel assembly format instead of AT&T format (easier to read in my opinion)
- Assemly will be in `file_that_contains_init_function.s`