Exercises for Architectures of Supercomputers

1st Exercise, Nov 2, 2023

Farhad EbrahimiAzandaryani

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

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

General information about exercises

- Exercises are **not** mandatory, but its contents are **part of the exam**

- Typical exercise schedule
  - Solution to last week's exercise is presented
  - New assignment for the next week is given

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

Accounts

- All students enrolled in the StudOn course by 6pm on Oct 24 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: hpcv274h – hpcv377h
  - 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 "Woody" and "Meggie" clusters

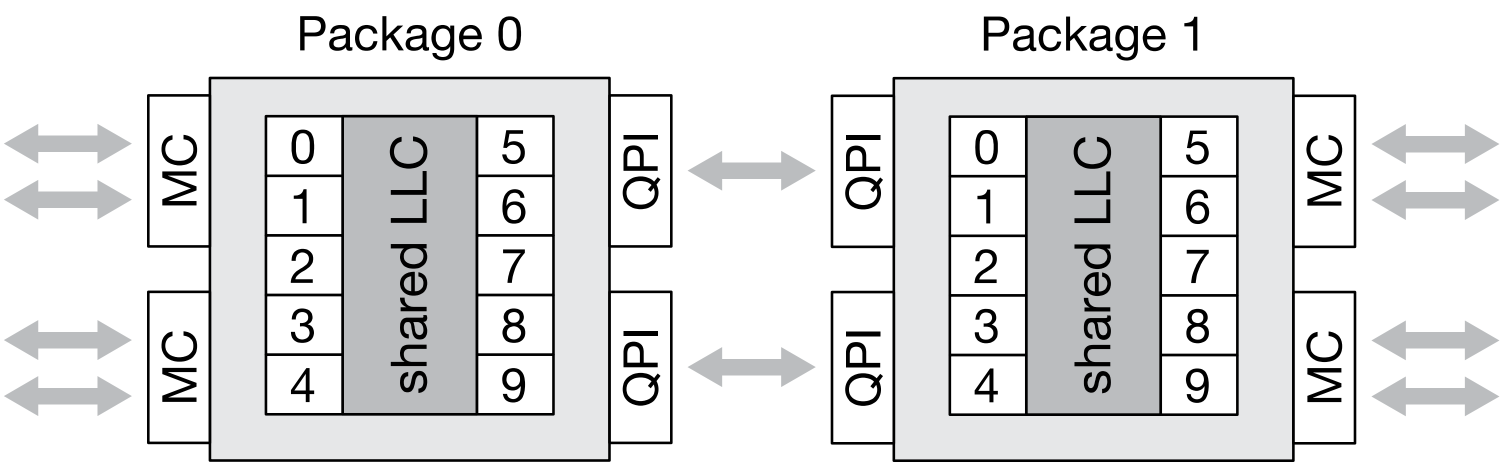
Meggie

- 728 compute nodes
- Dual-socket (i.e., dual processor) Broadwell nodes
- 10 cores per processor, clocked at 2.2 GHz (nominal clock rate)
- 64 GB DDR3-1600 (regular nodes)
- More information available online at https://hpc.fau.de/systems-services/documentation-instructions/clusters/meggie-cluster/

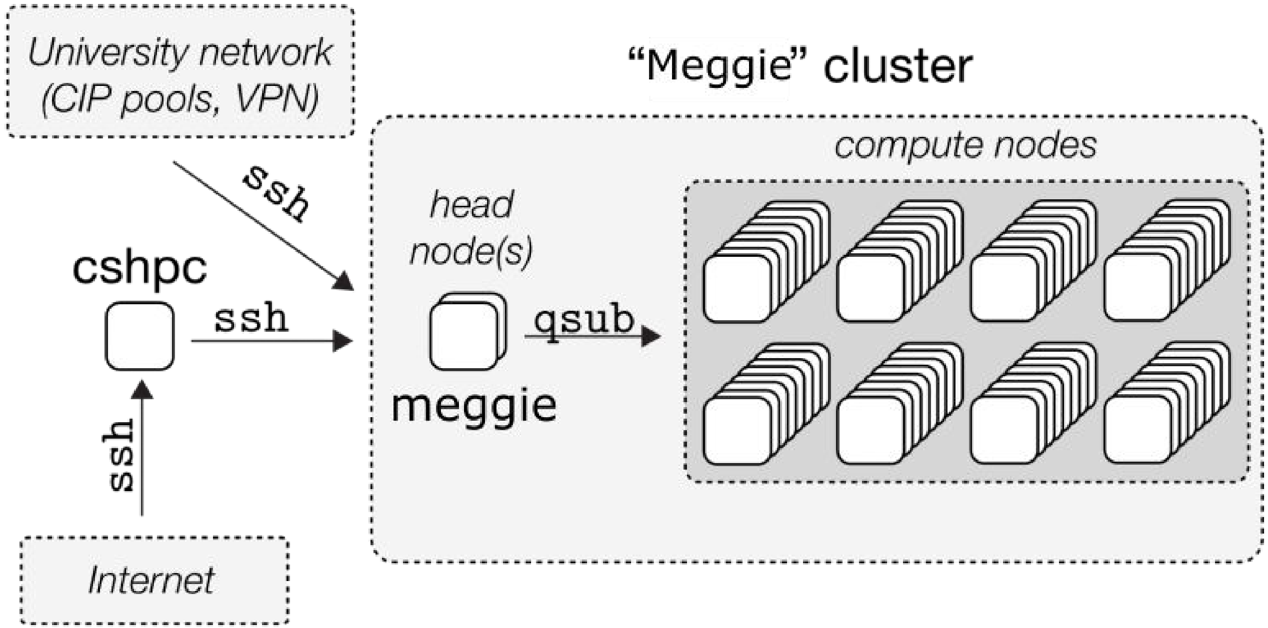
Layout of a single „Emmy“ compute node

Dual-Socket Ivy Bridge-EP nodes

- 2x Xeon E5-2630 v4 **chips** (**CPUs**, **processors, packages**)
- 10 **cores**, 20 **SMT-threads** (**logical cores**) per chip
- CPU clock: 2.2 GHz (nominal), GHz (Turbo mode)
- 64GB DDR3-**memory** (**RAM**)

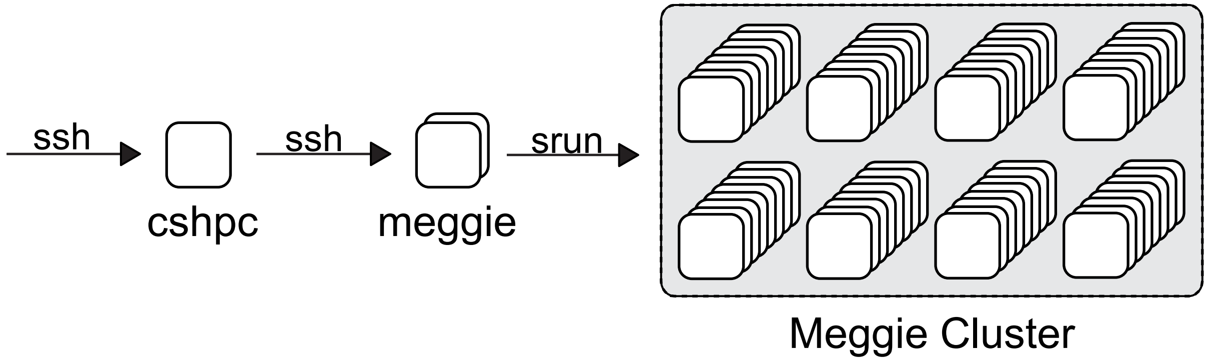


Overview



- Access from inside the university's network via head node (meggie.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@meggie.rrze.fau.de

Working with cluster resources

- Allocate one compute node for one hour
  - srun -p work --time=1:00:00 --ntasks-per-node=1 --cpus-per-task=20 --nodes=1 --exclusive --constraint=hwperf --cpu-freq=2200000 --pty bash -l
    - p, partition Choose node partition (work=default, devel=higher priority)
    - time The amount of time you want to allocate a node
    - ntasks-per-node Maximum number of tasks per node
    - cpus-per-task Number of allocated CPU cores per task (for multithreading)
    - N,nodes Minimum amount of nodes to be allocated
    - exclusive Grants exclusive access to the allocated node
    - C,constraint hwperf grants access to hardware performance counters (necessary for running likwid-perfctr), only use when needed
    - cpu-freq Sets the allocated CPU cores frequency to requested number, if possible
    - pty Execute task in specified terminal mode
    - l,label Prepend task ID to output

Module system

- Some software (e.g. compiler) must be explicitly added to the $PATH
  meggie1:~$ module avail gcc intel
- ---------- /apps/modules/data/compiler ---------------
  gcc/11.2.0 gcc/12.1.0
  ---------- /apps/modules/data/compiler ----------------
  intel/2021.4.0 intel/2022.1.0
  ---------- /apps/modules/data/development -----------
  intelmpi/2021.6.0
  [...]

- You should be using Intel Compiler 2021.4.0 for the exercise
  module load **intel/2021.4.0**

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

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 your_login_name@meggie.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 ArchSup 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

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- 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 -Y command-line parameter, and use, e.g., the emacs GUI editor
  - You can also write your own ssh-config and/or use local text editors like vscode

- Compile code
  - Make sure to load the Intel C compiler from the module system
    $ module load intel/2021.4.0
  - 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 srun command to request a node
    $ srun --time=1:00:00 --cpus-per-task=20 --nodes=1 -l

- 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

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- 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 computer 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

- Your first program will measure the memory bandwidth (in GB/s) of a function that initializes a dynamically allocated array that contains double-precision floating-point numbers initialized to a value of 1.0
  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. Dynamically allocate memory
     - 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
  3. Initializing the dynamically allocated memory
     - Use a for-loop to traverse and initialize the array
     - Make sure to put the function to initialize the array into a **separate**

1st Exercise: Benchmarking skeleton

- Your first program will measure the memory bandwidth (in GB/s) of a function that initializes a dynamically allocated array that contains double-precision floating-point numbers initialized to a value of 1.0

  4. 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)
  5. Output measured bandwidth
     - Convert from B/s to GB/s
     - Output result

     - USE PRAGMA NOVECOR
       - Removed it because with novector small dataset was slower than large one; this is also the case with vector (reason: function-call overhead; BUT without novector, assembly is vectorized and more complex!)

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**