TDT4200 Parallel computing

Exercise setup

This document contains information on how to access and use the Snotra cluster, how to install a programming environment for Windows, and how to install a C compiler. Other dependencies needed for solving the programming tasks in this course will be explained as part of the problem sets. Do not hesitate to contact us if you encounter any problems so we can assist you in setting things up.

Using the Snotra cluster

All TDT4200 students have access to the Snotra cluster via SSH. To use this cluster you can follow the procedure described below.

Connecting to the cluster

If you are on campus, you can connect to the server by running

```
ssh <username>@snotra.idi.ntnu.no
```

If you are outside campus, you have to jump through a bastion host to connect to the server by running

```
ssh -J <username>@login.stud.ntnu.no <username>@snotra.idi.ntnu.no
```

When you connect to the cluster you should automatically be allocated a shell on an available server. This ensures that the resources you need for the programming tasks are scheduled and dedicated to you as a user. The shell will terminate after 2 hours, at which point you can simply request a new shell to continue working (assuming there is no queue). Each user gets a maximum of 1 shell.

If you are an employee at NTNU, you might end up on a server called *snotra-login*. This is the login node for the Slurm cluster. The login node is a simple virtual machine and no work should be done on this node. You will be allocated the same shell as regular students by running

```
connect tdt4200
```

Working on the cluster

Viewing and canceling jobs

You can run the `show-jobs` and `stop-jobs` commands via SSH to view or cancel any active jobs in the case you want to be allocated a new shell, but did not properly exit your previous shell.

Show a list of your active jobs by running

```
ssh <username>@login.stud.ntnu.no <username>@snotra.idi.ntnu.no show-jobs
```

or

```
ssh <username>@login.stud.ntnu.no <username>@snotra.idi.ntnu.no stop-jobs
```

Stop your active jobs by running

```
ssh <username>@login.stud.ntnu.no <username>@snotra.idi.ntnu.no stop-jobs -f
```

Note that you still have to pass the `-J` flag to the `ssh` command if you are outside campus.

Working with files

The default home directory on the servers is the official NTNU home directory. This is a directory that you can mount locally on your own machine, so that you can edit files locally and make them available on the remote server to compile and run. NTNU provides guides on how to connect your home directory depending on what OS you are running:

- Linux
- Mac OS
- Windows

You should use the `sambaad.stud.ntnu.no` network directory.

Server specifications

The current list of nodes (subject to change) consists of Selbu and Oppdal with 20 × Nvidia T4 GPUs, 2 × Xeon Gold 6230 and 376GB RAM each. Each user is allocated 4 cores, 8GB RAM and 1 GPU for PS1-PS3, and 2 cores, 8GB RAM and 1 GPU for PS4-PS6.

Setting up a programming environment for Windows

Installing WSL

For completing the programming tasks in this course on a Windows machine, we strongly recommend that you use WSL (Windows Subsystem for Linux). Use this guide to install WSL. It installs Ubuntu Linux by default.
After installing WSL, you can install the programming tools you need and compile programs inside of the WSL Terminal.

Working with files

The files you want to compile will need to be placed in a location that is accessible to the compiler. To do this, execute "explorer.exe" inside of the terminal, as shown here. This will open a Windows Explorer window of the folder in which you reside in the terminal. Move files into this folder to make them available inside the WSL Terminal. To view and navigate the folders interactively in the WSL Terminal, we recommend learning some basic commands. The first 3 commands in this list are useful for the scope of this course.

Workflow

A basic workflow for C programming with WSL can be to move the handout code for a problem set into the WSL folder described above, open these files using some code editor, e.g., Visual Studio Code, and then write your program. When you want to compile your program you can run `gcc` or use the Makefile commands available in the handout code for the problem sets directly in the WSL terminal. For convenience, you can also tie WSL and Visual Studio Code together using this guide.

Installing Homebrew for Mac OS

We recommend that you use Homebrew for package management if you are working on a Mac. Homebrew can be installed by following this guide. Commands needed to use Homebrew can be found in the documentation.

Installing a C compiler

Linux

Update the system:

```
sudo apt update && sudo apt upgrade -y
```

Install the gcc compiler:

```
sudo apt install gcc -y
```

Check the gcc installation (should print the version of the gcc compiler that is installed):

```
gcc --version
```

Mac OS

Update Homebrew:

```
brew update
```

Install the gcc compiler using Homebrew:

```
brew install gcc
```

Check the gcc installation (should print the version of the gcc compiler that is installed):

```
gcc --version
```

Windows (in WSL terminal)

Update the system:

```
sudo apt update && sudo apt upgrade -y
```

Install the gcc compiler:

```
sudo apt install gcc -y
```

Check the gcc installation (should print the version of the gcc compiler that is installed):

```
gcc --version
```