Lab 1 - Profiling Parallel Code

High Performance Heterogeneous Computing

University of Minho

The Lab 1 focus on the development of efficient CPU code by revisiting the programming principles that have a relevant impact on performance, such as vectorisation and scalability of multithreaded algorithms. Submit jobs to the mei queue in SeARCH to measure code execution times, but be careful to always use the same node architecture (i.e., the nodes in the *cpar* partition).

1 The SeARCH cluster

In this section, you will learn how to:

- Log in to the university's HPC cluster (SeARCH);
- Authenticate using your university credentials;
- Load software modules;
- Create and edit files on the cluster;
- Compile programs;
- Submit jobs using the scheduler (sbatch);
- Access your cluster files directly on your local machine using sshfs or through Code IDE.

If this is your first contact with HPC systems, take your time to follow each step carefully. By the end of this lab, you will be able to run your first program on the cluster and manage your files easily.

1.1 Prerequisites

Before starting:

- Make sure you have a university account with SeARCH access;
- Have your username and password ready;
- Install an SSH client:
  - On Linux/macOS: the ssh command is preinstalled.
  - On Windows: use PowerShell or PuTTY.

1.2 Logging into the Cluster

The cluster is a set of powerful servers you can access remotely. They are installed here in the Informatics Department and are locally managed.

- Step 1: Open a terminal (Linux/Mac) or PowerShell/Putty (Windows);
- Step 2: Use the SSH command:
  - **ssh username@s7sci.di.uminho.pt**
  - Replace username with your university username.
- Step 3: Enter your password:
  - If the login succeeds, you'll see a welcome message!

Note: To exit/close the connection you can do *CTRL+D*

1.3 Understanding the Cluster Environment

- Login Node: Where you log in and prepare jobs. Do **not** run heavy computations here!
- Compute Nodes: Where your jobs run. You access these via the scheduler (sbatch), not directly (for now).

1.4 Loading Software Modules

At this point, the command line you are using is executing commands in SeARCH. This means that you no longer have access to software that you usually use in your personal computer. Fortunately SeARCH has most of the software that you will need available to use. However, to use these you need to follow proper steps. The cluster uses the module system to manage software.

- Check available modules: *module avail*
- Load module: *module load gcc/11.2.0*
- Check loaded modules: *module list*
- Remove loaded module: *module unload gcc/11.2.0*

1.5 Creating and Editing Files

You can create and edit files directly on the cluster using editors *vim*like or *nano*. For beginners, *nano* is easiest: *nano hello.c* This opens a text editor. Enter the following program:

```
#include <stdio.h>
int main() {
printf("Hello from the HPC cluster!\n");
return 0;
}
```

Save and exit:

- Press *CTRL+O* to save;
- Press *CTRL+X* to exit;

1.6 Compiling Programs

Now compile your program using the compiler you loaded earlier: *gcc hello.c -o hello*
Run it: *./hello*
You should see: Hello from the HPC cluster!

1.7 Submitting Jobs with SLURM (sbatch)

As mentioned earlier, when you log in, you are connected to a login node. This node acts as a front-end interface to the cluster and is intended only for lightweight tasks such as editing files, compiling code, or submitting jobs. It does not have the resources to run heavy computations directly. To run your programs on the actual compute hardware, the cluster uses a scheduler to manage and allocate resources. On SeARCH, the scheduler is SLURM. By submitting a job through SLURM, your program is automatically queued and executed on one of the available compute nodes. Important: All heavy computations must be submitted as jobs via SLURM; never run them directly on the login node.

- Step 1: Create a job script: *nano job.sh* Enter the following:

```
#!/bin/sh
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=00:01:00
#SBATCH --partition=cpar
#SBATCH --output=hello.out
module load gcc/11.2.0
./hello
```

  Save and exit.

- Step 2: Submit the job with: *sbatch job.sh*
- Step 3: Check job status: *queue -u username*

When the job finishes, check the output file: cat hello.out

1.8 Accessing & modifying cluster files from your local computer

Working directly on the cluster can be inconvenient when you want to edit or manage files from your own computer. This section offers convenient ways of addressing this issue.

1.8.1 SSHFS

Install SSHFS:

- Linux:
  - *sudo apt update*
  - *sudo apt install sshfs*
- MacOS:
  - brew install macfuse sshfs
- Windows:
  - Use WinFsp + SSHFS-Win

Create a local folder to mount the cluster wherever you want on your local machine:

```
mkdir ~/my_cluster
```

Mount your cluster home directory:

```
sshfs your_username@s7sci.di.uminho.pt:/home/your_username ~/my_cluster
```

Now, the folder on your local machine contains your cluster files and will sync automatically.

2 Social media network

Goals: to develop skills in the fundamentals of thread programming.

Imagine you work at a company running a large social network (think Twitter, LinkedIn, or TikTok). Each user in the network has a small feature vector (e.g. age, activity level, etc). The connections between users can be represented as a matrix with 0s and 1s. Most users have only a few connections. A few "celebrities" have **millions**. The team responsible for the algorithm that recommends *friends/followers* asks you for data they need to further improve it. They want: *For every user, the average feature vector of its friends.* After studying the problem you arrived to a simple solution. The needed data can be obtain by simply multiplying the matrix of friendships with the matrix of user features and apply the average.

$$\mathbf{followers}_{3\times3} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{user\_features}_{3\times2} = \begin{bmatrix} 0.2 & 0.4 \\ 0.8 & 0.1 \\ 0.3 & 0.9 \end{bmatrix}$$

$$\mathbf{user\_followers\_features}_{3\times2} = \mathbf{followers}_{3\times3} \times \mathbf{user\_features}_{3\times2}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \\ 0.8 & 0.1 \\ 0.3 & 0.9 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.1 \\ (0.2+0.3) & (0.4+0.9) \\ 0.8 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.1 \\ 0.5 & 1.3 \\ 0.8 & 0.1 \end{bmatrix}$$

(if we average each row over the number of follows):

$$\mathbf{user\_followers\_features}^{\text{avg}}_{3\times2} = \begin{bmatrix} 0.8 & 0.1 \\ 0.25 & 0.65 \\ 0.8 & 0.1 \end{bmatrix}$$

Lab / Ext 1.1 Complete the worker algorithm to answer the question asked.

Lab / Ext 1.2 Create a function that follows a master-worker schema to parallelize the code and distributes the workload equally among available threads in the system. The parallelized code should deliver the same result as the original sequential multiplication.

Lab / Ext 1.3 Create two executables resultant from different compilation options:

```
gcc -Wall -O2 -march=ivybridge -g -o gemm_O2 ...
```

```
gcc -Wall -O3 -march=ivybridge -g -o gemm_O3 ...
```

Use *perf* to profile both executables for a dataset size that does not fit in cache. Do not forget to load the necessary modules in both the frontend and compute nodes:

```
module load gcc/11.2.0
```

Profile both executables using *perf* for a single thread (sequential) and 20 threads. Choose the hotspots profiling to get an overall idea of the code bottlenecks.

```
perf stat ./executable
```

Ext 1.4 Extend the code to ensure that each thread writes on a private chunk of memory (the output matrix), and implement a way to efficiently concatenate the results into a single output matrix after each thread processes its data. Compare the execution time of this version of the code with Lab 1.3 for problem sizes that do not fit in cache. Does the performance scale as expected? Profile the code and identify the inefficiencies (CPU, memory, coding inefficiencies), and possibly their causes.