Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

High-Performance Computing Lab for CSE **2024**

Student: Benedict Armstrong Discussed with: FULL NAME

Solution for Project 1a

Due date: 11 March 2024, 23:59

1. Euler warm-up [10 points]

1.1. Module System

The module system allows Euler users to quickly and easily configure their environment to use centrally installed software package. A detailed description can be found in the Module System documentation.

There are two systems currently in use. The older system is called `Environment Modules` and the newer system is called `LMOD Modules`. All new software installations are done with LMOD Modules.

```
# List all available modules
module avail

# Load a module
module load <module_name>

# list all loaded modules
module list
```

Listing 1: Module System

1.2. SLURM

The Euler cluster uses SLURM to manage and schedule jobs. To run a job on the cluster, you need to submit a job script to the SLURM scheduler. A detailed description can be found in the SLURM documentation.

1.3. Hello Euler!

We start by compiling and running a simple C program on the Euler cluster. The program is called `hello_euler.cpp` and should print "`Host name: <hostname>`" to standard out.

To run the compiled program on the cluster, we need to submit a job script to the SLURM scheduler. The job script is called `hello_euler.slurm` and should look like this:

The job can then be submitted to the SLURM scheduler with the following command:

The code and output can be found in the `hello_euler` directory.

```
#!/bin/bash
#SBATCH --job-name=hello_euler        # Job name    (default: sbatch)
#SBATCH --output=hello_euler.out      # Output file (default: slurm-%j.out)
#SBATCH --error=hello_euler.err       # Error file  (default: slurm-%j.out)
#SBATCH --time=00:01:00               # Wall clock time limit
#SBATCH --nodes=1                     # Number of tasks
#SBATCH --ntasks=1                    # Number of tasks
#SBATCH --cpus-per-task=1             # Number of CPUs per task
#SBATCH --mem-per-cpu=1024            # Memory per CPU
#SBATCH --constraint=EPYC_9654

srun hello_euler
```

Listing 2: Job script for running hello_euler.cpp

```
    sbatch hello_euler.sh
```

Listing 3: Submitting a job to the SLURM scheduler

```
#!/bin/bash
#SBATCH --job-name=hello_euler_2      # Job name    (default: sbatch)
#SBATCH --output=hello_euler_2.out    # Output file (default: slurm-%j.out)
#SBATCH --error=hello_euler_2.err     # Error file  (default: slurm-%j.out)
#SBATCH --time=00:01:00               # Wall clock time limit
#SBATCH --nodes=2                     # Number of tasks
#SBATCH --ntasks=2                    # Number of tasks
#SBATCH --cpus-per-task=1             # Number of CPUs per task
#SBATCH --mem-per-cpu=1024            # Memory per CPU

srun hello_euler hello_euler
```

Listing 4: Job script for running hello_euler.cpp on multiple nodes

1.4. Multiple Nodes

We can run the same code on multiple nodes using the following job script:
Where we set the number of nodes to 2 and the number of tasks to 2. The output can be found in the `hello_euler_2.out` file.

2. Performance characteristics [50 points]

2.1. Peak performance

The peak performance of a CPU can be calculated using the following formula:

$$p_{core} = n_{super} \times n_{FMA} \times f_{SIMD} \times f_{core} \tag{1}$$

$$p_{CPU} = n_{core} \times p_{core} \tag{2}$$

in our case this is equal to:
For the EPYC_7742 node:

$$p_{core} = 2 \times 2 \times 256 \times 2.25 \times 2.25 = 4095 \text{ GFLOPS} \tag{3}$$

$$p_{CPU} = 64 \times 4095 = 261120 \text{ GFLOPS} \tag{4}$$

For the EPYC_7763 node:

$$p_{core} = 2 \times 2 \times 256 \times 2.45 \times 2.45 = 4768 \text{ GFLOPS} \tag{5}$$

$$p_{CPU} = 64 \times 4768 = 304192 \text{ GFLOPS} \tag{6}$$

2.2. Memory Hierarchies

The output of running `lscpu` and `hwloc-ls` can be found in the `memory_hierarchies` directory. As in the example in the assignment there are also two PDFs detailing the memory hierarchy of the EPYC_7742 and EPYC_7763 nodes. In summary both nodes have 8 NUMA nodes, with 8 cores per NUMA node. More information on NUMA can easily be found in the Wikipedia page. Basically it means that the nodes have faster access to their specific part of the shared memory. The rest of the numbers can easily be read out of the two PDFs detailing the memory hierarchy.

2.2.1. Cache and main memory size

Cache	**EPYC_7742**	**EPYC_7763**
L1d	32KB	32KB
L1i	32KB	32KB
L2	512KB	512KB
L3	16MB	32MB
NUMA	63GB	31GB
Total Machine	502GB	248GB

Table 1: Cache and main memory size for both nodes

2.3. Bandwidth: STREAM benchmark

2.4. Performance model: A simple roofline model