Lab 1: Introduction to SLURM

ECE 455: GPU Algorithm and System Design

Due: Submit completed PDF to Canvas by 23:59 PM 9/19

Overview

This lab introduces the basics of SLURM job submission and resource requests on the Euler cluster. You will learn how to write SLURM batch scripts, submit jobs, and verify their execution.

Euler Instruction

```
~$ ssh your_CAE_account@euler.engr.wisc.edu
~$ sbatch your_slurm_scrip.slurm
```

You should NEVER run your program on the log-in node with the interactive mode. Doing so will risk your account being blocked by the IT. Instead, you should work on your local machine and set up a GitHub repo to transfer code from your local machine to your Euler node, and then compile and run it using a proper `sbatch` script.

Submission Instruction

Specify your GitHub link here:

Note that your link should be of this format: https://github.com/YourGitHubName/ECE455/HW01

Problem 1 — Hello World with SLURM

Task: Write a SLURM script `p1.slurm` that:

- Requests the `instruction` partition
- Runs for a maximum of 1 minute
- Uses 1 CPU core
- Prints `Hello from node <hostname>` to the output file

`p1.slurm`:

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:01:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --output=hello.output

cd $SLURM_SUBMIT_DIR
echo "Hello from node $(hostname)"
```

Problem 2 — Compile and Run a C++ Program

Task: Write a SLURM script `p2.slurm` that:

- Load `gcc/13.2.0` via `module load`
- Compile a C++ program `main.cpp` in the current directory
- Run the compiled binary

`p2.slurm`:

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:01:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --output=compile_run.output

cd $SLURM_SUBMIT_DIR
module load gcc/13.2.0
g++ main.cpp -o main
./main
```

`main.cpp`

```
#include <iostream>
#include <cstdlib>

int main() {
    const char* hostname = std::getenv("HOSTNAME");
    std::cout << "Hello from C++ on node: "
              << (hostname ? hostname : "unknown")
              << std::endl;
    return 0;
}
```

Problem 3 — Multi-Threaded CPU Job

Task: Write a SLURM script `p3.slurm` that:

- Request 4 CPU cores in a single task
- Load `gcc/13.2.0`
- Compile and run a program `parallel.cpp` that uses `std::thread`

`p3.slurm`:

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:02:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --output=multicpu.output

cd $SLURM_SUBMIT_DIR
module load gcc/13.2.0
g++ -std=c++17 parallel.cpp -o parallel -pthread
./parallel
```

`parallel.cpp`

```
#include <iostream>
#include <thread>
#include <vector>

void worker(int id) {
    std::cout << "Thread " << id
              << " running on CPU core" << std::endl;
}

int main() {
    unsigned int n_threads = std::thread::hardware_concurrency();
    std::cout << "Launching " << n_threads << " threads...\n";

    std::vector<std::thread> threads;
    for (unsigned int i = 0; i < n_threads; ++i) {
        threads.emplace_back(worker, i);
    }

    for (auto& t : threads) {
        t.join();
    }
    return 0;
}
```

Problem 4 — GPU Job

Task: Write a SLURM script `p4.slurm` that:

- Request 1 GPU
- Load the `nvidia/cuda` module
- Compile `kernel.cu` with `nvcc`
- Run the program

`p4.slurm`:

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:02:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --gpus-per-task=1
#SBATCH --output=gpujob.output

cd $SLURM_SUBMIT_DIR
module load nvidia/cuda
nvcc kernel.cu -o kernel
./kernel
```

`kernel.cu`

```
#include <iostream>
#include <cuda_runtime.h>

__global__ void vector_add(const float* a, const float* b, float* c, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        c[idx] = a[idx] + b[idx];
    }
}

int main() {
    int n = 16;
    size_t size = n * sizeof(float);

    float h_a[16], h_b[16], h_c[16];
    for (int i = 0; i < n; i++) {
        h_a[i] = i;
        h_b[i] = i * 2;
    }

    float *d_a, *d_b, *d_c;
    cudaMalloc((void**)&d_a, size);
    cudaMalloc((void**)&d_b, size);
    cudaMalloc((void**)&d_c, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);
```

```
    vector_add<<<1, n>>>(d_a, d_b, d_c, n);

    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    std::cout << "Result: ";
    for (int i = 0; i < n; i++) {
        std::cout << h_c[i] << " ";
    }
    std::cout << std::endl;

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    return 0;
}
```

Problem 5

Describe the challenges you encounter when completing this lab assignment and how you overcome these challenges.