Lab 3: Parallel Programming Using OpenMP

ECE 455: GPU Algorithm and System Design

Due: Submit completed PDF to Canvas by 23:59 PM 10/3

Overview

This lab introduces OpenMP for shared-memory parallel programming in C++. You will learn to parallelize loops, sums, and matrix operations using OpenMP directives.

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: https://github.com/wcharmon/ECE455/tree/main/HW03

Note that your link should be of this format: https://github.com/YourGitHubName/ECE455/HW03

Problem 1: Parallel Sum with OpenMP

Task: Convert a sequential sum of an array into an OpenMP parallel version using `#pragma omp parallel for`. Use an array of size 1,000,000 and compute the sum of all elements.

Solution

parallel_sum.cpp

```cpp
#include <iostream>
#include <vector>
#include <omp.h>

int main() {
    const int N = 10000000;
    std::vector<int> data(N, 1); // all ones
    long long sum = 0;

    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < N; ++i) {
        sum += data[i];
    }

    std::cout << "Sum = " << sum << std::endl;
    return 0;
}
```

p1.slurm

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:01:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --output=parallel_sum.output

cd $SLURM_SUBMIT_DIR
g++ -fopenmp parallel_sum.cpp -o parallel_sum
./parallel_sum
```

Problem 2: Parallel Matrix Multiplication with OpenMP

Task: Parallelize a matrix multiplication using OpenMP. Multiply two square matrices of size 512x512.

Solution

parallel_matmul.cpp

```cpp
#include <iostream>
#include <vector>
#include <omp.h>
```

```cpp
int main() {
    const int N = 1024;
    std::vector<std::vector<int>> A(N, std::vector<int>(N, 1));
    std::vector<std::vector<int>> B(N, std::vector<int>(N, 2));
    std::vector<std::vector<int>> C(N, std::vector<int>(N, 0));

    #pragma omp parallel for
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            int sum = 0;
            for (int k = 0; k < N; ++k) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }

    std::cout << "C[0][0] = " << C[0][0] << std::endl;
    return 0;
}
```

p2.slurm

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:02:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --output=parallel_matmul.output

cd $SLURM_SUBMIT_DIR
g++ -fopenmp parallel_matmul.cpp -o parallel_matmul
./parallel_matmul
```

Problem 3: Measure Parallel Speedup

Task: Modify Problem 1 to measure execution time using `omp_get_wtime()`. Compare 1, 2, 4, 8 threads.

Solution

parallel_sum_timing.cpp

```cpp
#include <iostream>
#include <vector>
#include <omp.h>

int main() {
    const int N = 10000000;
    std::vector<double> data(N, 1.0);
```

```
    for (int threads = 1; threads <= 8; threads *= 2) {
        double sum = 0;
        double t0 = omp_get_wtime();

        #pragma omp parallel for reduction(+:sum) num_threads(threads)
        for (int i = 0; i < N; ++i) {
            sum += data[i];
        }

        double t1 = omp_get_wtime();
        std::cout << "Threads: " << threads
                  << ", Time: " << t1 - t0
                  << " sec, Sum: " << sum << std::endl;
    }
    return 0;
}
```

p3.slurm

```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time=00:02:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --output=parallel_sum_timing.output

cd $SLURM_SUBMIT_DIR
g++ -fopenmp parallel_sum_timing.cpp -o parallel_sum_timing
./parallel_sum_timing
```

Problem 4

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

I was very confused for awhile how to do a loop with a changing number of threads but after looking at the solutions and the openmp documentation I was able to figure out how its done. This lab was definitely easier than the previous two but it was still a fun challenge.