

PROJECT #0

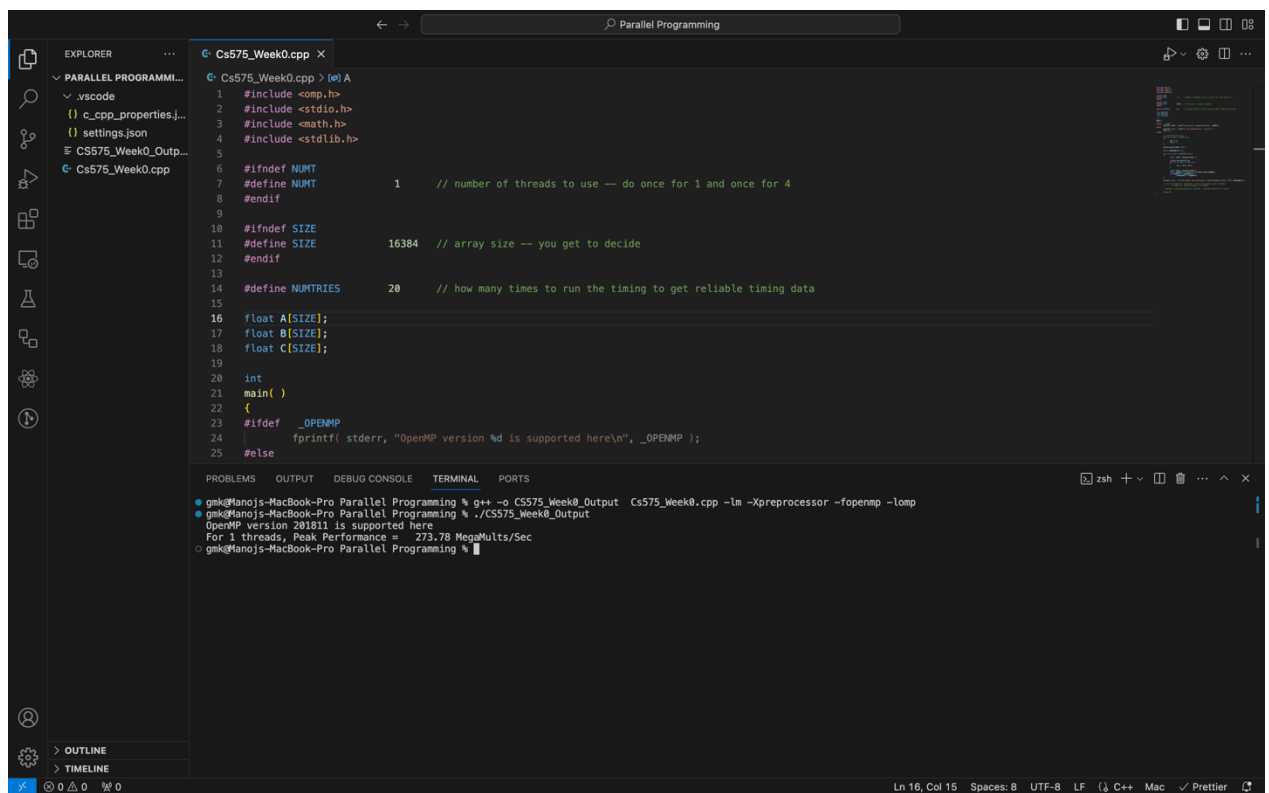
Simple OpenMP Experiment

Name: Manoj Kumar Gummadi

Email: gummadm@oregonstate.edu

Date: 08/04/2024

1) Performance or Execution time results for 1 thread



The screenshot shows a Visual Studio Code editor window with a file named `Cs575_Week0.cpp` open. The code is a C++ program that uses OpenMP for parallelization. It defines a number of threads (NUMT) as 1, an array size (SIZE) as 16384, and the number of trials (NUMTRIES) as 20. The program declares three float arrays (A, B, and C) of size SIZE. In the `main` function, it checks if OpenMP is supported and prints the OpenMP version. The terminal output shows the command to compile and run the program, and the resulting output, which includes the OpenMP version and the peak performance for 1 thread.

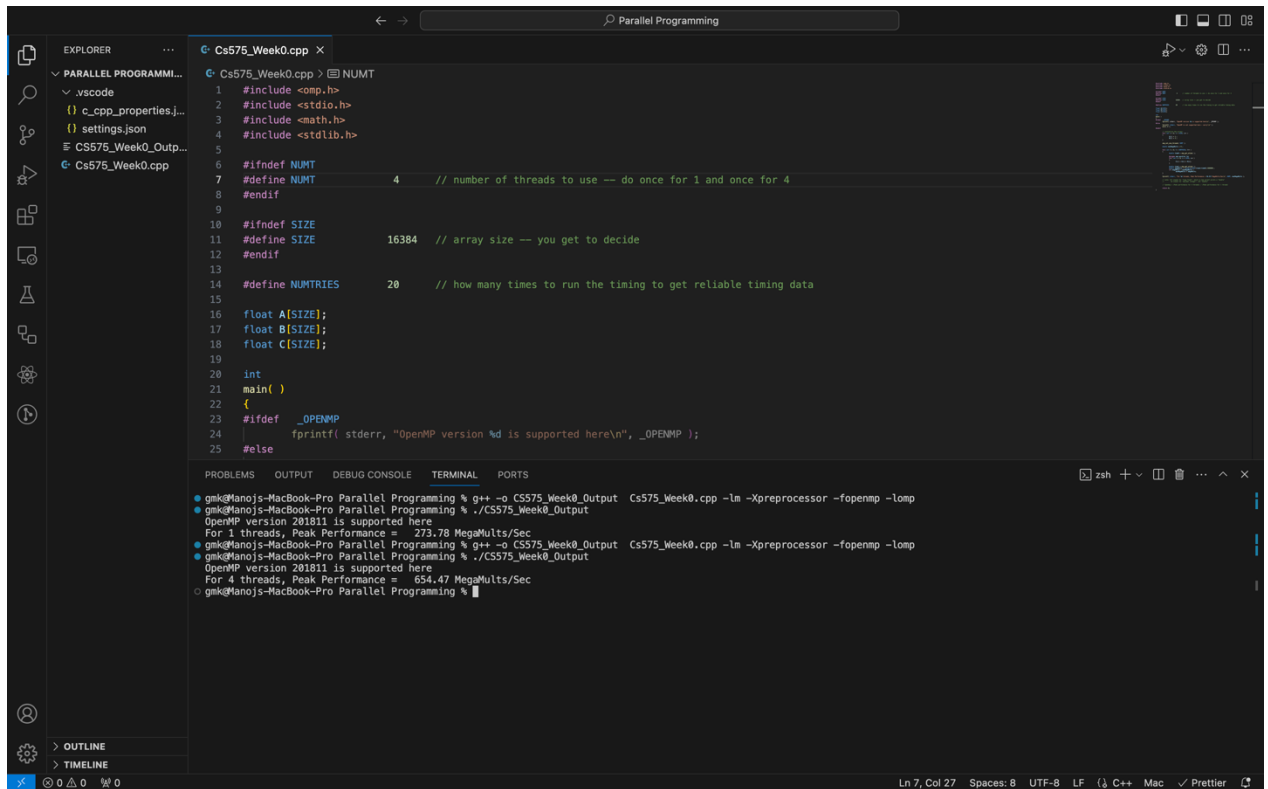
```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 #ifndef NUMT
7 #define NUMT 1 // number of threads to use -- do once for 1 and once for 4
8 #endif
9
10 #ifndef SIZE
11 #define SIZE 16384 // array size -- you get to decide
12 #endif
13
14 #define NUMTRIES 20 // how many times to run the timing to get reliable timing data
15
16 float A[SIZE];
17 float B[SIZE];
18 float C[SIZE];
19
20 int
21 main( )
22 {
23     #ifdef _OPENMP
24         fprintf( stderr, "OpenMP version %d is supported here\n", _OPENMP );
25     #else
```

TERMINAL OUTPUT:

```
gsk@Manoj's-MacBook-Pro: Parallel Programming % g++ -o Cs575_Week0_Output Cs575_Week0.cpp -lm -xpreprocessor -fopenmp -lomp
gsk@Manoj's-MacBook-Pro: Parallel Programming % ./Cs575_Week0_Output
OpenMP version 201811 is supported here
For 1 threads, Peak Performance = 273.78 MegaMults/Sec
gsk@Manoj's-MacBook-Pro: Parallel Programming %
```

Ans: 273.78 MegaMults/Sec

2) Performance or Execution time results for 4 threads



The screenshot shows a VS Code editor with a C++ file named `Cs575_Week0.cpp`. The code defines a parallel program using OpenMP. It sets the number of threads to 4, the array size to 16384, and the number of trials to 20. The program calculates the peak performance for 4 threads. The output window shows the execution results:

```
gmk@Manojs-MacBook-Pro Parallel Programming % g++ -o CS575_Week0_Output Cs575_Week0.cpp -lm -xpreprocessor -fopenmp -lomp
gmk@Manojs-MacBook-Pro Parallel Programming % ./CS575_Week0_Output
OpenMP version 201811 is supported here
For 1 threads, Peak Performance = 273.78 MegaMults/Sec
gmk@Manojs-MacBook-Pro Parallel Programming % g++ -o CS575_Week0_Output Cs575_Week0.cpp -lm -xpreprocessor -fopenmp -lomp
gmk@Manojs-MacBook-Pro Parallel Programming % ./CS575_Week0_Output
OpenMP version 201811 is supported here
For 4 threads, Peak Performance = 654.47 MegaMults/Sec
gmk@Manojs-MacBook-Pro Parallel Programming %
```

Ans: 654.47 MegaMults/Sec

3) One-thread-to-four-threads Speedup (>1.)?

Ans: (Peak performance for 4 threads) / (Peak performance for 1 thread)

$$\text{Speedup (S)} = 654.47 / 273.78 \Rightarrow 2.39$$

4) Parallel Fraction (Fp)?

Ans:

$$(4.0/3.0) * (1.0 - (1.0/S)) = (4.0/3.0) * (1.0 - (1.0/2.39))$$
$$= 1.333 - 0.582 = 0.751$$

5) Commentary

a) Tell what machine you ran this on.

Apple M3 Macbook Pro, OS: Sonoma 14.4.1

b) What performance results did you get?

1 thread - 273.78 MegaMults/Sec

4 threads - 654.47 MegaMults/Sec

c) What was your 1-thread-to-4-thread speedup?

$S = 2.39$

d) Your 1-thread to 4-thread speedup should be less than 4.0. Why?

Managing multiple threads incurs overhead due to setup, synchronization, and teardown, diminishing the benefits of parallelization. Additionally, Amdahl's Law dictates that some parts of a program must run sequentially, limiting the overall speedup achievable through parallelization and resulting in less than linear improvement. Furthermore, when threads compete for shared resources like CPU caches and memory bandwidth, it can lead to contention, causing delays and reducing overall efficiency. Ensuring data consistency among threads through synchronization mechanisms such as locks and barriers introduces additional processing overhead, detracting from the desired speedup. Moreover, load balancing challenges arise when tasks are unevenly distributed among threads, leading to inefficient resource utilization and hindering optimal parallel execution.

e) What was your Parallel Fraction, F_p ?

Ans: 0.751