CS 575 Parallel Programming

Project #0

Simple OpenMP Experiment

Name: Rachel Xing (#934229189)

Email Address: xingru@oregonstate.edu

Description: This project is a simple OpenMP program to test the speedup of parallel processing by comparing performance between 1-thread and 4-thread execution. The program initializes two large floating-point arrays, each containing 20,000 elements, performs pairwise multiplication between them, and stores the results in the third array of the same size. To ensure reliable performance data, the multiplication operation and the writing of the result into the third array are executed 20 times for each thread configuration. The peak performance from each set of trials is then used to calculate the 1-thread-to-4-thread speedup factor and parallel fraction.

Commentary:

1. **Tell what machine you ran this on.**

   I ran the program on two machines. The first is the **OSU Flip3 server**, which runs Enterprise Linux 9.5 and is equipped with an Intel® Xeon® Gold 6126 CPU @ 2.60GHz. The second is **my personal laptop**, a 2024 13-inch MacBook Air with an 8-core Apple M3 chip running macOS Sequoia 15.3.2.

2. **What performance results did you get?**

   Flip machine:
   Peak Performance with 1-thread = **486.02 MegaMults/Sec**
   Peak Performance with 4-thread = **1781.11 MegaMults/Sec**

```
flip3 ~/CS575ProjectArchive/Project0 1003$ bash Project0.bash
NUMT = 1
OpenMP version 201511 is supported here
For 1 threads, Peak Performance =   486.02 MegaMults/Sec
NUMT = 4
OpenMP version 201511 is supported here
For 4 threads, Peak Performance =  1781.11 MegaMults/Sec
```

   MacOS:
   Peak Performance with 1-thread = **716.98 MegaMults/Sec**
   Peak Performance with 4-thread = **1331.53 MegaMults/Sec**

```
[ 50%] Building CXX object CMakeFiles/Project0.dir/Project0.cpp.o
[100%] Linking CXX executable Project0
[100%] Built target Project0
OpenMP version 202011 is supported here
For 1 threads, Peak Performance =   716.98 MegaMults/Sec
```

```
[ 50%] Building CXX object CMakeFiles/Project0.dir/Project0.cpp.o
[100%] Linking CXX executable Project0
[100%] Built target Project0
OpenMP version 202011 is supported here
For 4 threads, Peak Performance =  1331.53 MegaMults/Sec
```

3. **What was your 1-thread-to-4-thread speedup S?**

Flip machine: $S = \frac{(Performance\ with\ four\ threads)}{(Performance\ with\ one\ thread)} = 1781.11\frac{MegaMults}{Sec} / 486.02\frac{MegaMults}{Sec} = \mathbf{3.665}$

My macbook: $S = \frac{(Performance\ with\ four\ threads)}{(Performance\ with\ one\ thread)} = 1331.53\frac{MegaMults}{Sec} / 716.98\frac{MegaMults}{Sec} = \mathbf{1.857}$

4. **Your 1-thread-to-4-thread speedup should be less than 4.0. Why do you think it is this way?**

There are two main factors I can come up with that prevent the program from achieving a perfect speedup factor of 4.0 when increasing from one thread to four. First, only the portion of the code for pairwise multiplication and writing into the new array is parallelized. Other parts such as macro definitions, array initialization, and the for-loop over trials still run sequentially even during parallel execution. This sequential work limits the overall speedup that can be achieved through parallel processing. Second, as we learned from this week's material, each thread has its own state and stack. The overhead involved in initializing and switching among multiple threads can introduce extra time and resource costs that are not present in single-threaded processing. This is analogous to energy use in the physical world, where some loss or inefficiency is inevitable even with optimization and perfect utilization is impossible.

[Extra Comment: An interesting observation is that the 1-thread performance of the Apple M3 chip is better than that of the Intel Xeon Gold 6126 processor, but its overall 1-thread-to-4-thread speedup is about half of the Xeon's. This difference is due to the inherent design goals of the two processors. The Intel Xeon Gold 6126 is designed for servers and workstations with a focus on multi-core performance, while the consumer-grade Apple M3 chip prioritizes single-threaded performance and energy efficiency. Besides, although I've attempted to quit all background applications, the helper processes spawned by them don't always appear clearly in the activity monitor. These processes can still run or restart automatically even after quitting the main program. Furthermore, other necessary system processes are competing for CPU time, slowing down parallel processing along with the unkilled helper processes. In contrast, the remote server is dedicated to the computing task, where CPU and memory resources are specifically allocated to minimize contention. The Flip machine also runs on a Linux system, which is more optimized for multiprocessing compared to other operating systems.]

5. **What was your Parallel Fraction, Fp? (Hint: it should be less than 1.0, but not much less.)**

Flip machine: $Fp = \frac{4}{3} * \left(1 - \frac{1}{S}\right) = \frac{4}{3} * (1 - \frac{1}{3.665}) = \mathbf{0.970}$

My macbook: $Fp = \frac{4}{3} * \left(1 - \frac{1}{S}\right) = \frac{4}{3} * (1 - \frac{1}{1.857}) = \mathbf{0.615}$