PROCESSES VS THREADS TESTING & REPORT

COMP 8005

SPENSER LEE A00925785

Contents

Executive Summary

This program uses the Taylor series to calculate an approximate value of Pi. This is accepted to be an inefficient way to approximate Pi; however, the main purpose of this program is to compare the performance of processes and threads, so this task is satisfactory.

After extensive testing between threads and processes, I have not found any significant performance difference for this particular application.

Introduction

This program uses the Taylor series to approximate the value of Pi. The Taylor series to compute pi is as follows:

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots\right)$$

It is understood that using the Taylor series to approximate Pi is very inefficient. Since the main purpose of this program is to compare the performance of threads and processes, this task is suitable.

The program was written using C++, using the std::thread library from C++11 and fork() for processes.

The program accepts arguments for the number of terms in the infinite series to compute, along with how many processor workers or how many thread workers per process.

For I/O operations, each worker will write its current series result to a file for every single loop iteration. Additionally, the parent process will write the final calculation to a results file, along with the execution time in milliseconds.

Parallel Operation

I split up the work between processes by taking the total number of series calculations (1/3, 1/5, 1/7, etc.) and dividing that up between the total number of workers. Each worker would perform the exact same loop, but each one starting at a different offset within the series.

For example, if I had 2 workers with a total number of series calculations of 3, worker 1 would start the series calculation at 1/3 and end at 1/5, then worker two would start the series calculation at 1/7 and end at 1/9. Then, each worker would add their series result into a global result that can be used in the formula 4 (1 – globalresult).

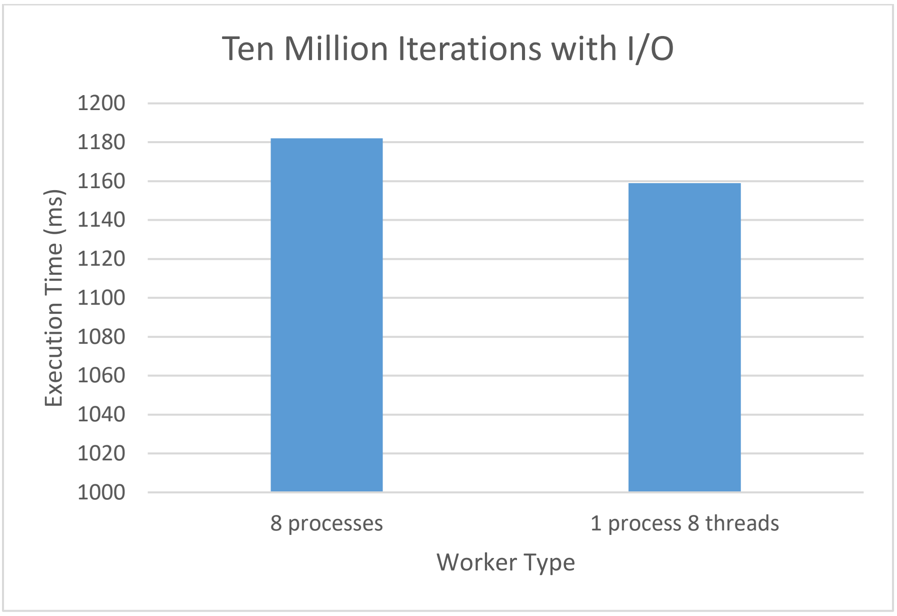
Analysis

In order to compare threads and processes accurately, I ran through several testing scenarios on an Intel 4790K which has 4 cores with hyperthreading, so total 8 compute cores.

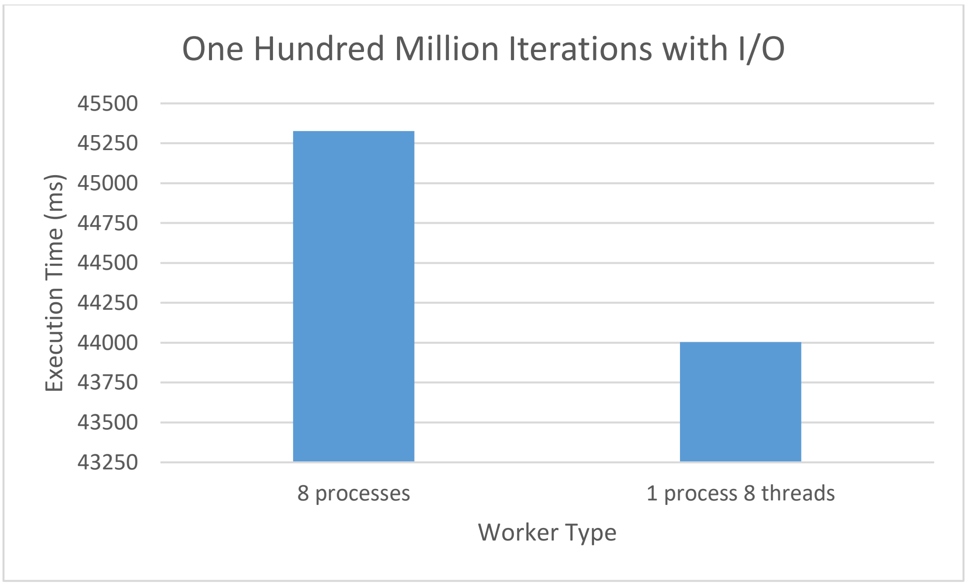
1. Execute ten million (10000000) terms of the Taylor series with I/O operations ENABLED using:
   a. 8 Processes
   b. 1 Process with 8 threads
2. Execute one hundred million (100000000) terms of the Taylor series with I/O operations ENABLED using:
   a. 8 Processes
   b. 1 Process with 8 threads
3. Execute ten billion (10,000,000,000) terms of the Taylor series with I/O operations DISABLED using:
   a. 8 Processes
   b. 1 Process with 8 threads
   c. 4 Processes with 2 threads process
   d. 2 Processes with 4 threads per process

Findings

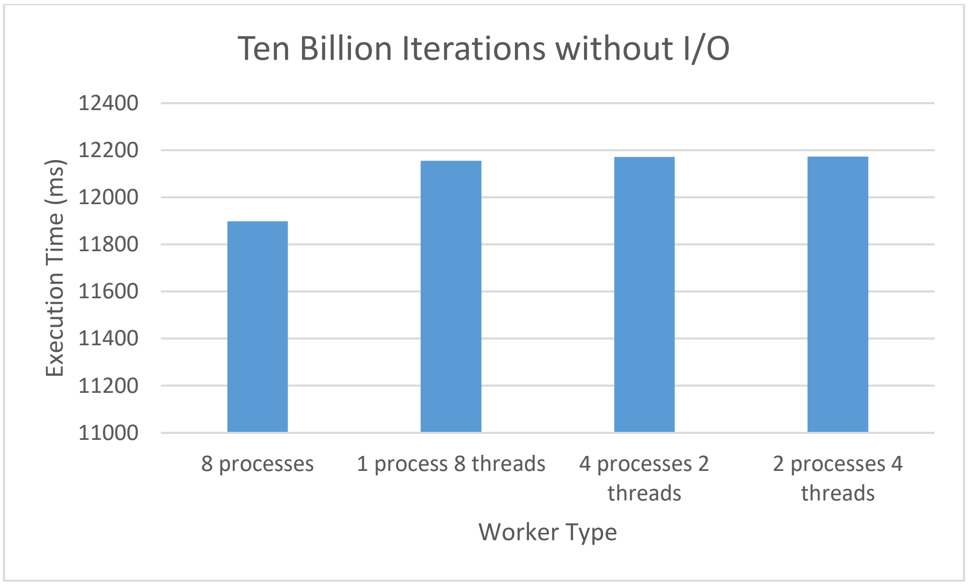
1. After running the program with each work type 10 times and finding the average, I found that threads were very slightly faster than processes (by 23 ms); however, this seems to be within the margin of error.



2. With this larger workload, the performance difference was more apparent, showing that threads were on average 1324 ms faster. However, this difference is still relatively small compared to the entire execution time of about 45 seconds. It appeared to me that I/O operations were bottlenecking the performance and was the main source of execution time difference.



3. In this test, I disabled the I/O operations because they seemed to be the main source of execution time increasing. After doing so this allowed me to increase the iteration count. Disabling I/O operations gave results that were opposite to the previous tests; the processor only worker type performed on average the fastest, whereas before the threaded worker type was just beating it. However, the difference between the fastest (8 processes) and the slowest (2 processes 4 threads) was only 275 ms.



Conclusion

From all of the tests that I ran, it has shown me that for this particular work task, the performance difference between threads and processes is very minimal. All of my results were in the margin of error, so it is difficult to say anything conclusive without developing different kinds of work tasks.

For this particular work task, which involves a lot of division and adding, it appears that processes and threads have about the same performance characteristics.