Lebanese American University

CSC447: Parallel Programming for Multi-Core and Cluster

Course Project: A Parallel Monte Carlo Simulation for Black-Scholes Option Valuation

Project Report

Author:
Sabrina Azar

Supervisor:
Dr. Haidar Harmanani

April 26, 2018

1 Experimental Setup

For experimental testing I first used my laptop, a Macbook Pro 2016. This works for OpenMP and for testing if the code works or not, but is not good for testing OpenACC because OpenACC does not support the GPU on macOS.

Therefore, I also used the computers in the computer labs at LAU for experimenting and comparing speedups. The computers in the lab run Ubuntu 14.04 LTS, have 15.6 GiB of RAM, a Intel Xeon CPU E5-1620 v4 @ 3.50 GHz x 8 processor and Quadro K420 for Graphics.

For editing the code, I used Visual Studio Code on my Macbook (except when I had to use the computer labs to test OpenACC). For compiling and running the program, I used iTerm:

```
make clean
make all
time ./csc447.x params.txt NUMBER OF THREADS
```

For example (4 threads):

```
make clean
make all
time ./csc447.x params.txt 4
```

2 Experimentation: Black-Scholes Parameters

I tried changing each line in params.txt, one by one (to test smaller and larger values). For example, I would change the first line but keep all the others unchanged. The only two parameters that made a difference were the fourth parameter (gamma) and the last one (the number of trials). When gamma is very high, the simulation is faster. The number of trials also affects the time, of course: when the number of trials is high, the simulation takes more time.

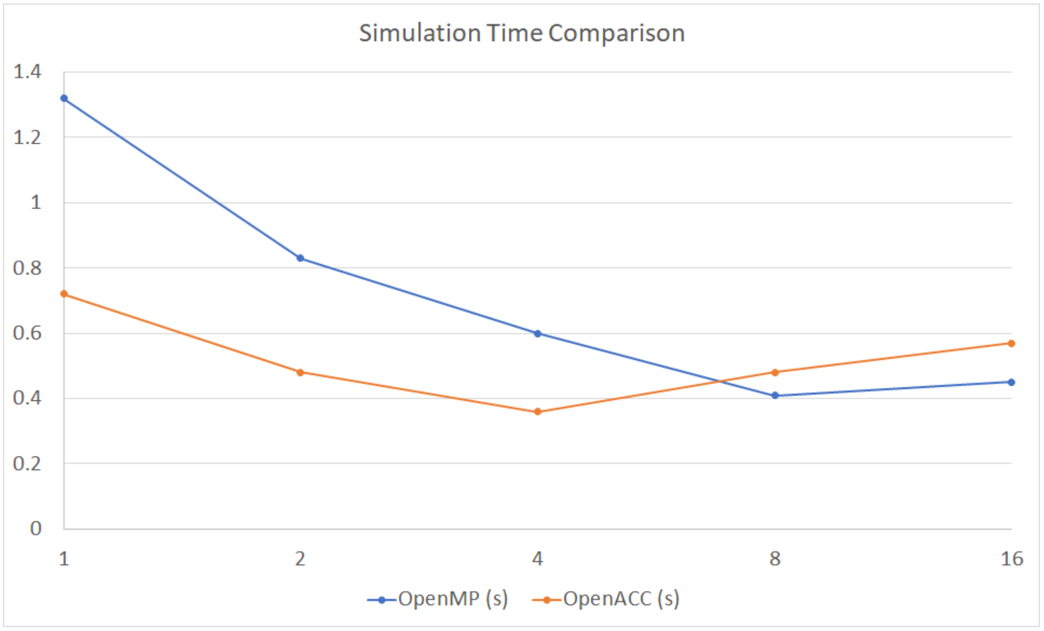
3 Experimentation: Number of Threads

First, we set the number of trials to 10,000,000. Increasing the number of threads leads to a decrease in simulation time. This shows that my parallelization code is working correctly. When I use too many threads (more than there are cores), I get a small speed decrease. This is normal because the cores are then each switching between 2 tasks, instead of each focusing on one task, so some time is lost between the switching.

4 Speedup and performance analysis

Table 1: Simulation Time Comparison

Number of threads	Sim time (OpenMP)	Sim time (OpenACC)
1	1.32 s	0.72 s
2	0.83 s	0.48 s
4	0.60 s	0.36 s
8	0.41 s	0.48 s
16	0.45 s	0.57 s



For discussion, see Section 3. It seems that task switching is more noticeable for OpenACC (it leads to a decrease in speed even when only using 8 threads).

I also ran the serial code, which took 0.96 seconds to run. So when using 1 thread only, the serial code is actually faster than OpenMP code. This makes sense because we are doing extra steps for parallelization, like generating and storing the random numbers first. When using multiple threads, OpenACC looks like the best choice (but we must be careful about the number of used threads), closely followed by OpenMP.

5 Additional Remarks

Please see the code comments for remarks. I also used git commits to show my progress!