Project Proposal for Functional Parallel Programming

By Griffin Newbold (gcn2106),
Sparsh Binjrajka (sb4835),
Anna Christensen (ajc2321)

I. Introduction

After class on the 13th of the current month, I had asked Professor Edwards what kinds of algorithms we should consider for parallelization. He told me and the others that remained to choose an algorithm or process that had grabbed our attention. Nothing fit that criteria for me, I am not one to look at breadth-first search and salivate. To me the joy of programming comes from the result of what I could make using data structures and algorithms, not the data structures and algorithms themselves. Especially after working so hard to land an internship I can simply not find low level joy in those anymore, it only made me more of an applications individual rather than more theory oriented.

That all being said, I am not one to simply overlook all that has passed me by, if I have to choose a topic that stems from something that grabbed my attention. Well the field narrowed quickly, one of the greatest things I have been a continual part of in my time at Columbia since my second semester is being a Teaching Assistant. I have worked each semester so far under Professor Adam Cannon. In the fall he teaches a course called Computing in Context. In honor of this being my last semester as a teaching assistant for that course, I take inspiration from there for my project. The second project given to students who undergo the economics context deals with options pricing, specifically in regards to monte carlo simulations. Primarily students were asked to develop monte carlo simulation algorithms with regards to european and asian call options.

II. Overview

The main premise is a parallelization of monte carlo simulations with respect to options pricing. We would provide implementations for the *european call option*, *the asian call option* as well as the *down and out barrier option*. For reference the european call's exact pricing formula is the following:

$$C_0 = \frac{1}{(1+r)^T} \sum_{k=0}^{T} (T\ C\ k)(p^*)^k (1 - p^*)^{T-k} (u^k d^{T-k} S_0 - K)^+$$

With the following being the definition of p*:

$$p^* = \frac{1 + r - d}{u - d}$$

Of course you can lose some of the details by wrapping it in a monte carlo simulation which is what the students are asked to do along with using this exact formula. I will omit the details for the Asian call option but they will be included in the full report. The more trials of the monte carlo simulation you perform the closer you get to the exact answer by the law of large numbers.

III. Algorithmic Details

As for how I would go about doing this, I would start off with the sequential versions and then as far as parallelization is concerned just going off of initial thinking, I could make use of the parMap rpar function from Control.Parallel.Strategies. Other options may come in handy as well.

As for input sizes and testing. For standard unit testing to make sure the results I get are proper, I will be using the values expected of the students to solve for since I can trust their accuracy and since I have concrete access to them. We can stress test both the sequential and parallel versions of the simulation with increasing the magnitude of the quantity of simulations.

We can also try differing numbers of cores in order to determine where we start seeing diminishing returns. In the event the final presentation requires a demo then for that I will be using a 2020 M1 Macbook air, but all figures and data will come from my desktop which, for reference, and this will be stated again in the report but the machine I will be using to develop and run my simulations is equipped with the following specifications:

- CPU: 11th Gen Intel Core i7-11700k @ 3.60 Ghz (tldr 8 cores 16 threads)
- GPU: NVIDIA GeForce RTX 3060 Ti
- Ram: 16 gigs of 3200 Mhz