15-418 Project Proposal

Due: October 31, 2016

TITLE:
Parallel Computation of Binomial Security Pricing Model
Team Members: Qifang “Charlie” Cai (qcai), Xiqiao Shan (xshan)
Web Page: https://caiqifang.github.io/binomial-pricing-model-project/

SUMMARY:
Our goal of this project is to use parallel computing techniques to increase the calculating performance of binomial security pricing model. We will implement the project on Nvidia GPU platforms using CUDA API. In the end, we wish we can build a real-time program that reacts to changes in variables and returns the theoretical prices of securities. Hope this pricing engine can serve as the core of a black-box trading system.

BACKGROUND:
Binomial security pricing model is a fundamental technique used in the financial industry. The idea of this model is that we can use a binomial tree to describe the evolution of security price. At time 0, we can denote the security price to be S0. At the next time period, the price S0 can evolve into S1(Head) with an up factor of u and S1(Tail) with a down factor of d. A multiple period binomial model is shown in the graph.

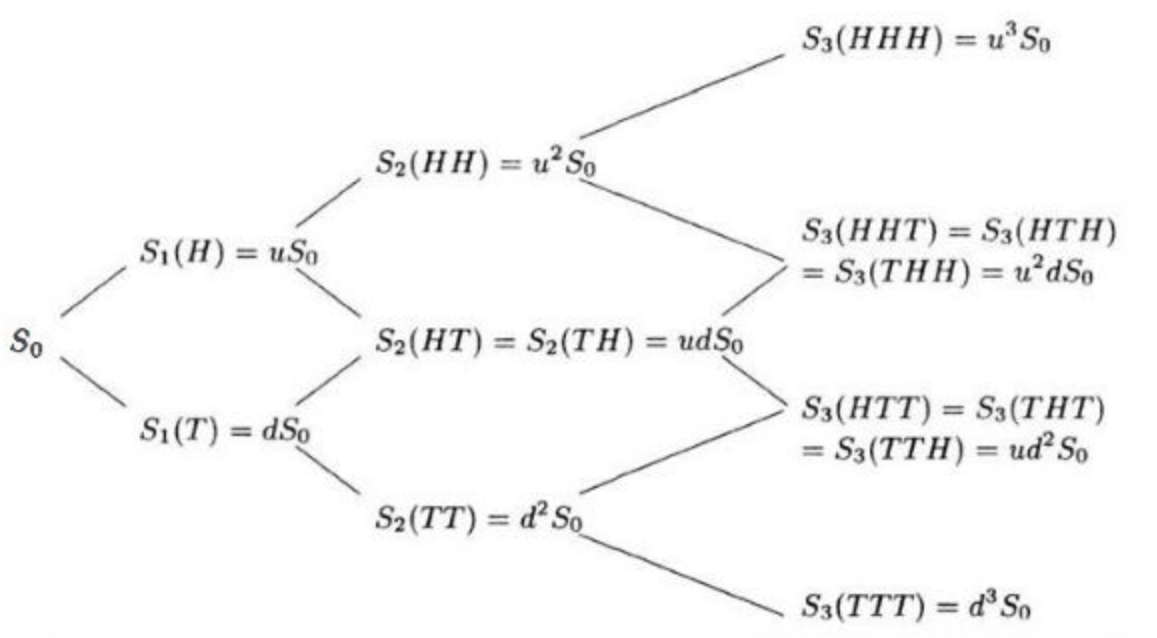

Fig. 1.2.1. General three-period model.

Picture from Stochastic Calculus for Finance I: The Binomial Asset Pricing Model by Steven E. Shreve. (In this case, the up factor and down factor (u and d) are fixed throughout time.)

To calculate the initial price of the security, we introduce the technique of backward induction. Given the arbitrage free interest rate r, the up factor u and down factor d, the no-arbitrage price of the security at time n can be computed recursively backward in time by this equation

$$V_n(\omega_1\omega_2\ldots\omega_n) = \frac{1}{1+r}[\tilde{p}V_{n+1}(\omega_1\omega_2\ldots\omega_n H) + \tilde{q}V_{n+1}(\omega_1\omega_2\ldots\omega_n T)].$$

Where p and q are the risk-neutral probabilities given by

$$\tilde{p} = \frac{1+r-d}{u-d}, \quad \tilde{q} = \frac{u-1-r}{u-d}. \qquad (1.2.15)$$

We can draw some quick observations of this model. The price of each tree node depends on the price of its left and right children nodes and different parameters in the backward induction formula. Different path calculates its price using different children nodes, thus paths are independent of each other. This property allows us to parallelize the calculation of an N period binomial model, theoretically improving its performance from O(2^N) to O(N) time complexity. We will need to use some data to train and calibrate the binomial model and find the best parameters (u, d, r) before we can backward induct the current correct price of the security. If a trading system sees a difference between the market price and theoretical model price, it can further exam the situation and explore potential arbitrage in the mispriced security.

THE CHALLENGE:

Some identified challenges include:

Problem Size -

The nature of the binomial model presents some difficulties. As the time period increases, the total nodes in the binomial tree grows exponentially. (Total of 2^N nodes) Initially, we are aiming to have a 30-period binomial model, but we need to do some math and computer science optimizations. There is simply no platform that can calculate and store the 2^30-level binomial tree.

Memory Management -

Assume that each node only contains a single floating number, because we have 2^30 nodes in total, there will be no reasonable storage device that can store this much data. We need to find a way to compress, hash and even abandon duplicated information. At the same time, we will be accessing value in the memory frequently. There's no way we can fit everything in the cache, so we need to find a way to access data that best utilizing the GPU memory and cache.

Smart Work Distribution -

All the security price calculation involves the backward induction equation. The calculation time does not vary too much. However, we still need to manage to distribute the calculate in a clever way to best utilize the GPU.

Model Training -

For simplicity, we use the binomial pricing model with three variables (up factor u, down factor d, and interest rate r). At the beginning, we don't have a concrete idea about the value of different variables. First thing to do in the program is to use training data to tune the model and find the best fitting variable value. Only then we can use the variable to produce correct theoretical price.

Time/ Speed -
One of the goals is to use this program as the pricing engine to a black-box trading system. This goal poses a significant challenge and has a strict time requirement. To catch the mispricing arbitrage opportunities in the market, the system needs to react to a variety of products and provide pricing suggestions almost in real time. We will try our best to speed up the program and see if we can complete calculation in single digit second or even less than a second.

RESOURCES:
Computer Platform:
CPU and Nvidia GPU platform (potential benefit from GPU parallel computing)
Starter Code:
As for now, we do not have any starter code. We plan to start from scratch.
Coding Language:
C/C++
Textbook Reference:
Stochastic Calculus for Finance I: The Binomial Asset Pricing Model by Steven E. Shreve.
Faculty Advisers:
Dr. David Handron (CMU-21370)
Prof. William Hrusa (CMU-21378)
Prof. Todd C. Mowry (CMU-15418)
Prof. Brian Railing (CMU-15418)

GOALS AND DELIVERABLES

PLAN TO ACHIEVE
In this project, we want to develop a parallelized CUDA program for binomial security pricing model. This parallelized program should to the best of its capability use the Nvidia GPU computing power and greatly increase the calculation performance. The program should be able to adapt to market changes and adjust its parameters in runtime. This allows the program to evolve along with the market to find the best theoretical price of securities. The preliminary design of the workflow is as follow:

1. Define binomial pricing model
2. Train the model with predetermined data
3. Run model to suggest correct price
4. Adjust parameters of the model over time
5. Repeat step 2

To achieve this, we will implement

- A sequential build of the binomial model as base performance reference
- A CUDA parallelized build to best speed up the binomial model calculation
- Some peripheral program that helps fetching market data and monitoring status

HOPE TO ACHIEVE
If we are ahead of schedule and have the capacity, we want to fully optimize the program and shrink the runtime to be less than a second. We want to be able to achieve better accuracy and extend the model to more than 30 periods. If possible, we want to introduce more variables to the model (more than just u, d, and r) and make the model versatile enough to fit a wide range of financial products.

DEMO
We will be demonstrating our pricing model and hopefully an application that uses the power of this pricing engine. We wish to show how this system can fetch market data, calculating theoretical values and discover mispriced securities. Also, we will be comparing the sequential implementation and GPU parallelized implementation in terms of speedup and memory cost.

PLATFORM CHOICE:
We choose to use CUDA and C++ as the main programming techniques. After considering all the parallel platform we have coded in this semester, we think CUDA is the best choice. First of all, we do not need to do much communication between several computers. OpenMPI seems not a good choice here. Second, there are lots of index mapping and memory access operations within our program. If we use OpenMP, synchronization could become a big problem for us. Given the time we have in this semester, we think using CUDA is a clean and efficient solution. Last but not least, we have done “parallel scan” using CUDA in previous assignments and this inspired us of some optimization of parallel operations within our program. We are happy to try to implement what we learned from the assignment and make an improvement on our program based on the knowledge.

SCHEDULE:

Time	What we plan to do	Status
Nov 1 ~ Nov 7	Revise proposal Study the design and architecture of the program Understand the binomial model and its requirement	In progress
Nov 8 ~ Nov 14	Implement a simple sequential version of the program	
Nov 15 ~ Nov 21	Finish sequential program Measure the performance of the simple program Start to use CUDA and parallelize the program Nov 21st Project Checkpoint Report Due	
Nov 22 ~ Nov 28	Implement the CUDA program Building peripheral programs	
Nov 29 ~ Dec 5	Finish CUDA programming and debugging Assess the program performance Building peripheral programs	
Dec 6 ~ Dec 13	Finalize the program Building peripherals Compose the paper presentation Dec 13. Parallelism Competition (1:00-4:00pm) Final Report Due at 11:59pm	