# 15-418/ 618 Project Checkpoint

Due Mon, November 21st - 8 AM
Author: Qifang Cai; Xiqiao Shan

**Revised Project Timeline:**

| Time | What we plan to do | Team Member | Status |
|---|---|---|---|
| Nov 1 ~ Nov 7 | Revise proposal<br>Study the design and architecture of the program Understand the binomial model and its requirement | Work Together | Complete |
| Nov 8 ~ Nov 14 | Implement a simple sequential version of the program | Xiqiao: Build SGD Training Algo; Qifang: Build Sequential Model | Complete |
| Nov 15 ~ Nov 21 | Finish sequential program<br>Measure the performance of the simple program<br>Start to use CUDA and parallelize the program<br>Nov 21st Project Checkpoint Report Due | Xiqiao: Build SGD Training Algo; Qifang: Build Sequential Model<br>Work Together: Brainstorm the CUDA Implementation | In Progress<br>Haven't started CUDA Programming |
| Nov 22 ~ Nov 28 | Implement the CUDA program<br>Building peripheral programs<br>Obtain market data from professor<br>Plan how to display result | Xiqiao: Build CUDA DATA Training program;<br>Qifang: Implement CUDA Build;<br>Together: Integration | |
| Nov 29 ~ Dec 5 | Finish CUDA programming and debugging<br>Assess the program performance<br>Building peripheral programs | Work Together | |
| Dec 6 ~ Dec 13 | Finalize the program<br>Compose the paper presentation<br>Dec 13. Parallelism Competition (1:00-4:00pm) and Final Report Due at 11:59pm | Work Together | |

**Project Summary:**

In the past couple weeks, we took time to fully understand the concept of binomial asset pricing model and how to implement the model in a sequential manner. We took steps to simplify the model in math, which helped significantly reduce the calculation load and total memory cost. After researching the model, we used the model's property as a state process and recorded results from recurring calculation. This served the purpose of caching state results. We have programmed a sequential build of the binomial model that had 3 model parameters. To train the model, we first decided to try the Stochastic Gradient Descent (SGD) approach. Later we found out that our gradient steps were too large that the SGD approach didn't always converge. In the coming week, we will look into this issue and see if we can fix the bug. Or, we can approach other ways to training our 3-parameter model.

**Project Due Diligence:**

We are currently falling behind schedule. We originally plan to start the parallel version of our project. However, we encounter some problems (stated below). We would like to use this week to solve these problems and start the parallel version with CUDA. If the problems related to the sequential model and SGD training take too much of our time. We might skip the sequential build and dive into CUDA programming. Currently we have the performance metric of a single call of the sequential model. We believe we are still able to meet the deliverables and goals. It will be challenging, but it's possible. We don't plan to look into model with more than 3 parameters because we are already behind schedule.

**Presentation Plan:**

We would like to show a graph in the poster presentation. The graphs should show case the speed up of the CUDA parallel implementation compared to the sequential implementation. We still need to collect more data on the sequential build to have a meaningful base case. Also, we would like to have a graph comparison between real market quote and our calculated theoretical prices. We should see the market quote and theoretical prices move at similar direction. Also, it would be nice to have a background information section and technology insight section in the poster to help people understand the meaning behind our CUDA program.

**Current Result:**

We have successfully build the sequential model and tested its performance. The 500 level tree structure model takes around 250 millisecond to finish all the calculation needed to solve for theoretical initial value of the financial derivative. However, the model also allocates around 1MB of memory to cache its state calculation result. It goes over all 125751 states sequentially before reaching a final result.

**Issues and Problems:**

- **Problem with the model:**
    - The sequential build performs surprisingly well. After mathematical optimization, the model finish a cycle of calculation in 250 millisecond. If we suppose that the SGD approach calls the function 100 times, the training period is around 25 seconds. However, 25 seconds is a lot of room for improvement. To build the pricing engine similar to the real world implementation, we aims to have the training and calculation time reduced to around 1~5 seconds.
    - We have increased the binomial model to 500 time step. An interesting observation is that the model output is strongly related to the number of model

    - time step. We are confused about this result and want to consult computational finance professor to prove its accuracy.
    - We still need to find more data to verify the model's accuracy.

- **Problem with the training method:**
    - First, choice of training method. First we decide to do a trivial method without any algorithm. For example, manually increase or decrease the training parameters (p, q, r in formula). However, problems with this are (1) it is hard to unify whether to increase or decrease for each parameter which makes a simple method complicated (2) after some research we found out the training parameters used in real life trading is extremely small $10^{-9}$ ~ $10^{-8}$ , which is hard for us to decide steps to change the parameters.
    - Then we considered gradient descent to be good fit for this problem. Also gradient descent is included in our lectures, which has potential to gain great performance improvement applied parallelism. This is actually the method we are currently using, and we did find some problems with it.
        - First, the formula we have is hard to construct an error function and calculate derivatives for it. Since we have multiple levels, value of each level depends on the value of neighbour level. However, as what we did is a backward deduction when calculating theoretical value, changing a training parameter meaning losing accuracy on each deduction level. Therefore, we cannot do gradient descent during deduction (steps of a single iteration) but instead conduct similar process (error functions => changing parameters) between each iteration. Therefore, it is hard to follow standard gradient descent process.
        - Second, learning-rate of parameters. As stated above, each parameter is extremely small in real life. Since learning rate is usually much smaller than training parameter, it is hard to get a learning rate without losing accuracy. If we use definition of derivative (not formula) to calculate gradient of each parameters, we are hard to move forward since we gonna lose much accuracy during division of small number.

- **Logistic Problem:**
    - The coming weeks are really busy. We won't have much progress during the Thanksgiving Holiday week. Our schedules are packed with exams and homework from other classes after we get back from Thanksgiving. We need to be more efficient and move quickly.

**Miscellaneous:**

---

**Program Statistic:**

Function speed:

Average time elapsed for 1 function call: 250 millisecond
Average memory cost: 1 MB
Model Step Periods: 500 (500 levels tree structure)

**Experiment Data:**

| S0 | V0 |
|---|---|
| 219.07 | 0.84 |
| 218.96 | 0.75 |
| 218.79 | 0.64 |
| 218.44 | 0.56 |

**Notes:**