6951-GPU Project Proposal

Title:

Parallel Computation of Binomial Option Pricing Model

Goal

The goal in the model is to use parallel computing to increase the calculating performance of binomial option pricing model.

Abstract

I will implement this process via both C++ code on CPU and CUDA code on Nvidia GPU. In the end, I hope I can produce a prototype for potential real-time high-performance program that serves as kernel for a pricing system that reacts to changes in variables and returns the theoretical prices of options.

Background information

Binomial security pricing model is a fundamental technique in finance industry. The idea is to use a binary tree to depict the evolution of security price. When applied to options, it becomes option binary pricing model. When evolution steps are chosen to be extremely small, this mimics very well price movements as described by the Black Scholes model. As a result, it's a model worth exploring.

For another, I'm also taking FRE6883-- Financial Computing in C++ at the same time this semester. Binomial model is deeply probed in that class, which makes it easier for me to branch this model via parallel computing in GPU because sequential computing in C++ is already covered in that class.

Theory behind model

We denote option price to be `S0` at time 0. The price can evolve into `S1(head)` with an up factor `u` or `S1(tail)` with an down factor `d`. `u+d` could be 2, but doesn't have to. A multiple period binomial model is a standard full binary tree as it would be depicted in a Data Structure class. Pictures can be been in this book, though it's just a normal tree structure.

This is also the model covered in our C++ class. In this very model covered in book, the up factor `u` and down factor `d` are constants through the entire period.

Reference from Stochastic Calculus for Finance: The Binomial Asset Pricing Model by Steven E. Shreve.

To calculate the initial price of option, we take the backward induction technique. Given the

arbitrage-free interest rate `r`, the up factor `u` and down factor `d`, the no-arbitrage price of option at time 0 can be computed recursively backward by:

$V_n(w1w2...wn) = (1/1+r)*[p * V_{n+1}(w1w2...wn\ H) + q * V_n(w1w2...wn)]$;

where p and q are risk-neutral probabilities given by: `p=(1+r-d)/(u-d)` and `q=(u-1-r)/(u-d)`.

Some observations can be drawn from the model.

1. The price of current node depends only on its left, right child and parameters on the tree branch;
2. thus, each branch and price calculation of current node is independent of other branches.
3. Observation 2 is crucial : **because every path are independent of others, we can parallel compute an N-period binomial model instead of sequentially performing this, theoretically reducing time complexity from `O(2^n)` to `O(n)`.**

I'll need a good tuple of `(u,d,r)` to find the best simulation of backward price induction. `(u,d,r)` tuple is given as constants beforehand in the model. I would search on web to find a reasonably good set and eschew finding one by myself because that parameter-fitting process, a machine learning problem, would take me a lot extra time and it's beyond the theme of this project.

Challenges

1. Problem size: The nature of binomial models have this general difficulity. As time periods increases, the number of nodes increases exponentially. Say there are 30 periods, then there will be 2^30 nodes on level30. As a result, some optimizations should be done.
2. Running memory: This is consistent with problem#1. Because the insurmountable size of multiple-period tree, I need to find a way to compress, hash or abandon duplicated information.
3. The project itself is quite hard to implement. GPU and CPU projects are hard to deal with themselves. I will try my best to write those programs, but I cannot say it will be that good.

Resources

Coding: C++/C

Textbook references:

Stochastic Calculus for Finance I: The Binomial Asset Pricing Model, *Steven E,Shreve*.

Procedure

1. Find a nice tuple of `(u,d,r)` to start with;

2. Implement a sequential model in C++ as base for performance comparison;
3. Measure the performance of sequential program;
4. Implement a CUDA parallel model to speed up binomial calculation;
5. Measure the performance of CUDA program;
6. Summary and final report.