Final Project

Files

- GPUEngine: The updated version of OldGPUEngine. It is made to handle path dependent exotic option pricing using GPU tech. Inspired by the SNN codes in github.
- OldGPUEngine: Classic functional programming.
- OptionSystem: Ultimate goal of my project. Combining all pricing modules into a bascket system. However, it is relatively finished only in c++ part. So, it can be used as an alternative and comparison.

Intent

In my project, the goal is to construct an option pricing system as codes in folder GPUOptionSystem. I have implement the most part of c++ part and some skeleton for GPU part.

However, the GPU part(most of them are skeleton) I wrote in GPUOptionSystem is wrong because CUDA doesn’t support all features of c++. For example, cuda class do not allow dynamic polymorphism which is a base of most pattern.

Inspired from your guidance, I reconstruct a project named GPUEngine. And I defined a class to handle path-dependent option.

Features

- Can price both vanilla options and exotic time dependent options (Asian or knock-out option)
- Transfer the computation of mean pricing into device by reduction. Much more quickly.
- Object-Oriented programming tried on cuda

Results

- GPU engine result:(GPUEngine->kernal)

```
spot=20
strike=20
r=0.02
vol=0.2
timeToMaturity=1y
n_steps=300
n_trails=2000
vanilla call option:1.81008
asian call option:1.09816
up-and-out call option:1.23882
*************************************************************************************
Optimization: using reduction to calculate price generated by MC simulation:
vanilla call option:1.81008
asian call option:1.09816
up-and-out call option:1.23882
```

- Old GPU engine result for vanilla option(OldGPUEngine->Euler_Euro_Option.cu)

```
Call option or put option? (c/p)
c
This is Monte-Carlo simulation using Euler method.
rate: 0.02
volatility: 0.2
time to mature: 1
stock price: 20
exercise price: 20
Trials: 100000
Steps: 300
Time comsumed: 0.361616s
The mean of option value: 1.81101
The std of option value: 2.79994
The mean of underlying asset: 20.3965
The std of of underlying asset: 18.7952
The mean of log underlying asset: 2.99544
The std of log underlying asset: 0.199524
****************************************************************************************************
Comparation:
mean(log(samples of future stockPrices))=2.99544        log(stockPrice)=2.99573
std(log(samples of future stockPrices))=0.199524        volatility*sqrt(time to mature)=0.2
mean error: -0.00990051%        std error: -0.238039%
The number of zero stock price: 0 Ratio: 0%
Press any key to continue . . .
```

- CPU asian engine result(OptionSystem->AsianCallMain):

```
Enter vol
0.2

r
0.02

d
0

barrier
0
Number of dates
300

Number of paths
1000

For the asian call price the results are
0.0767405 2
0.828113 4
0.592598 8
0.835102 16
1.03513 32
1.08369 64
1.07149 128
0.968619 256
0.959884 512
0.973957 1000
```

- CPU up-and-out engine result(OptionSystem->UpAndOutCallMain):

```
Enter vol
0.2

r
0.02

d
0

barrier
30
Number of dates
300

Number of paths
1000

For the up-and-out call price the results are
0.66094 2
2.10711 4
1.46376 8
1.6086 16
1.92293 32
1.59188 64
1.60095 128
1.48332 256
1.34609 512
1.37138 1000
```

Conclusions

1. Although OO programming brings a lot of convenient especially in group project. An unpropered construction of class can cause memory shortage which may restrain our number of trials awfully. Exactly as you mentioned before.
2. GPU engine is much more powerful than cpu engine.

Problems and todo list:

1. my visual studio do not identify function __syncthreads() for some reason(do you know the reason?)
2. If the above problem can be solved. The random number generator can be more efficient by initializing curandState_t array once.
3. The structure remains lots of redundant codes.
4. Switch from geometric brownian motion to stochastic volatility model(Heston Model)

Reference

Engler, Gary. “timeSeriesSNN.” github, 2017, https://github.com/gengler1123/GPU-Computing-For-Finance. Accessed 14 Feb. 2017

Joshi, M. S. (2004). C++ Design Patterns and Derivatives Pricing (Mathematics, Finance and Risk) (2nd ed.). N.p.: Cambridge University Press.

Sanders, Jason, and Edward Kandrot. CUDA by Examplean: an introduction to general-purpose GPU programming. 1 ed., Addison-Wesley Professional, 2010.