Future-Model Simulation using CUDA

Raayan Pillai
CSC 266

November 16, 2017

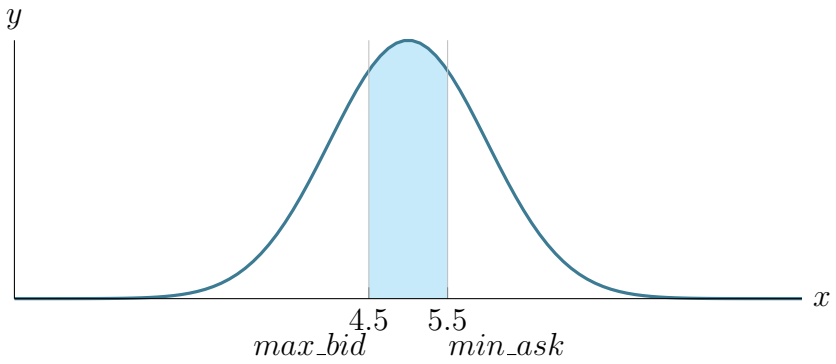
1 Introduction

My inspiration for this project is a side project that I have been working on for the past year. A friend and I have been developing an engine for analyzing and taking advantage of market-making and arbitrage oppurtunities between variouos cryptocurrency exchanges. The data comes in through websockets and REST APIs and is decomposed to an object-structure in Java. A problem with this project is that there are poor response times from some of these exchanges because they do not have perfect stability. Another approach to working with these markets would be to create speculative models. One way of doing this is by simulating markets to predict a future model. Predicting an accurate future model requires many possible futures, generated very quickly.

2 Proposed Implementation

2.1 Market Model

Markets are represented by four qualities, base currency, counter currency, *min_ask* price and *max_bid* price. The base and counter currencies are the ones being traded. The *min_ask* price is the cheapest one being sold and the *max_bid* price is the highest price someone will pay for the exchange. The $min\_ask > max\_bid$. Now these values are all supported by the order book, which is a representation of all the buy orders and all the sell orders. This is represented by some distribution which can be modelled by a uni or multi-modal gaussian.



Most simply by the figure above, this model would propose the *mean_price* is around 5.

2.2 GPU Accelerated Future-Model Simulation

Now that we have a model of the market represented by a gaussian distribution we can simulate a future. Suppose you have program P, it has a market model M which has k transactions, at time t_1. Each transaction T has a $price > 0$, and a $quantity \neq 0$ (negative: selling, positive: buying). Now on this model you have a hypothetical *trader* this object will place buy and sell orders based on the gaussian distribution to simulate actions on the market, corresponding to the order density from time t_1 to t_2. The resulting market model $M - future$ is a possibile future model of the market at time t_2. This simultion is only one possibility derived from relatively simple computations. This process could benefit heavily from parallel processing. Replicating this process among thousands of threads would allow for a variety of normal distribuations which could then be combined to form a *Mixture distribution*. This distribution can later be used to create a prediction of the future min_ask and max_bid prices but that is less of a parallel programming exploration and more of a data science one. This problem will require thorough memory management to find a proper size of market model and simulation time to maximize the amount of threads generating futures because quantity is what is important in this problem. Global memory will be used to store the gaussian and the base market model. Each thread will have its own representation of the market so it will be interesting to see how much can fit without reducing the thread count.

2.3 Algorithm (Python)

```
1  import numpy as np
2  from scipy.stats import norm
3  from scipy.stats import stats
4
5  orders = #populate orderbook from exchange
6
7  orders.sort()
8
9  mu, std = norm.fit(orders)
10 print(mu, std, len(orders))
11
12 for i in range(0, 1000):
13         mu, std = norm.fit(orders)
14         value = np.random.normal(mu, std, size=None)
15         match = False
16         for idx, x in enumerate(orders):
17                 if idx+1 != len(orders):
18                         if orders[idx] >= value and orders[idx+1] > value:
19                                 match = True
20                                 if mu > value:
21                                         del orders[idx]
22                                 else:
23                                         del orders[idx+1]
24                                 break
25         if not match:
26                 orders.append(value)
27 mu, std = norm.fit(orders)
28
29
30 print(mu, std, len(orders))
```

2.4 Multithreaded on GPU

Lines 13 to 28 will be run on individual GPU threads. The quantity of threads will be based on the availible memory divided by the book size, since each thread will need its own copy of the book. After computation of the future models the average mean will be taken as the likely price. This algorithm will be multithreaded on a CPU and compared to the GPU version to measure performance. Both versions will be compared to the actual future price of the book to measure accuracy. The CUDA implimentation will be done in C++ so the python packages being used for normal distributions will need to be implimented.

3 Afterword

I am no longer using the data generated for my CSC 240 project, to avoid any possible complications.

4 Sources

https://www.bitfinex.com/order_book
https://docs.bitfinex.com/v1/reference#rest-public-orderbook
https://en.wikipedia.org/wiki/Normal_distribution
https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables
https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch37.html