Liangzi AUTO: A Parallel Automatic Investing System Based on GPUs for P2P Lending Platform

Gang CHENa,*
Anhui Xiaoma Financial Information Consulting Company Limited
Part A in Blue Business Harbor, No.188, Qianshan Road, Hefei, Anhui, China
a chengang_p2p@163.com
*Corresponding author

Keywords: Parallel computing, GPU, P2P lending, Automatic investing.

Abstract. Nowadays, automatic investing becomes more and more popular in P2P platforms. For the sake of satisfying the lenders' better experiences and offering more flexible automatic investing strategy, in this paper we propose Liangzi AUTO-a parallel automatic investing system which can dynamically decide the investing sequences according to four factors at the moment of starting recruiting of a loan project. We design a dynamic scheduling algorithm for automatic investing and parallelize the scheduling algorithm to exploit Liangzi AUTO with assistance of modern GPUs. Experimental results show that we achieve a maximum speedup of 37.40X.

Introduction

Online peer to peer lending (P2P lending) platforms such as Lending Club [1] directly connect individuals that involve both of lenders and borrowers. As a financial information service medium that forbidden to join the borrowing and lending transactions, P2P platforms eliminate the need for traditional banking services [2]. In recent years, more and more platforms participate in Internet financial information services for borrowing or lending. P2P lending allows lenders to obtain much higher interest income than bank deposit with a credit risk that the loss resulting from defaulting on a borrower is directly born by the lenders. Figure 1 shows a typical P2P lending model.

Figure 1. P2P lending model.

Nowadays, many P2P lenders in China realize that the entire P2P marketplace is facing a reshuffle as a result of being exposed to various kinds of risks. In this context, they have transferred to P2P platforms with high reputation. On the other hand, due to lack of quality borrowers (i.e., quality assets) on these P2P platforms, lenders usually suffer no loan to lend. In order to alleviate the contradiction of assets absence for a P2P platform and time absence for a lender, many P2P platforms develop automatic
investing system to help the lenders to invest even if they do not have time to invest manually.

Up to now, most P2P platforms employ FCFS (First Come First Serve) strategy to design their automatic investing system. However, there is a significant limitation that FCFS does not take into account of many VIP lenders which are more contributable to a platform. That’s to say, when the automatic investing system cannot serve these VIP lenders timely, they may transfer to other platforms owning to the insufficiency to meet their investing demands, leading to unnecessary loss for this platform. For the sake of satisfying the above demands and offering more flexible automatic investing strategy, we propose a parallel automatic investing system (named Liangzi AUTO) in this paper.

Liangzi AUTO dynamically decides the investing sequences according to four factors at the moment of starting recruiting of a loan project. In a large-scale lenders scenario of a P2P platform, efficient automatic investing with a dynamic strategy exhibits time criticalness with data-intensive characteristics and high performance is essential. However, real-time demand is limited by the overall computer system performance to accomplish dynamically automatic investing operations for targeting lenders (i.e., automatic investors) within an accepted time, because of the potential competition with manual investors. We harness the enormous computation power of modern GPUs to achieve performance gains with the purpose of meeting the real-time demands. Our contribution in this paper is: (1) design a dynamic scheduling algorithm for automatic investing; (2) parallelize the scheduling algorithm to exploit Liangzi AUTO with assistance of GPUs; (3) demonstrate the obtained speedup of Liangzi AUTO.

Background and Related Work

GPU Architecture Overview

Modern GPUs offer a raw computing power that is often an order of magnitude larger than even the most modern multi-core CPUs, making them a relatively inexpensive platform for high performance computing [3]. Taking Nvidia GPU architecture as an example, it consists of global memory and an array of streaming multiprocessors (SMs). Each SM comprises an array of in-order streaming processors (SPs) to execute hundreds of threads concurrently. A GPU kernel is a function that is executed M times in parallel by M threads. These threads are divided into thread blocks. A thread block is further divided into thread warps, and a warp is consisted of 32 threads on Nvidia GPUs. Figure 2 shows Nvidia CUDA programming model. For a detailed introduction of GPU computing, we refer to Nvidia CUDA Programming Guide [4].

![CUDA programming model](image_url)
Related Work

There is a significant body of work focusing on P2P lending, with the aim of addressing different P2P problems. Wu et al. [5] present a recommendation model based on intelligent agent in P2P Lending for the borrowers, which helps borrower getting loan more efficiency. Vedala et al. [6] use classification algorithm to classify good and bad borrowers, where the input attributes consist of both core credit and social network information. Kumar et al. [7] use machine learning algorithms and preprocessing techniques to analyze and determine the factors which play crucial role in predicting the credit risk in P2P lending. Shen et al. [8] propose an effective approach of data preprocessing namely key points approximate fitting algorithm to identify different investment patterns. King et al. [9] develop a pricing and data/information service system for financial analytics base on web services and GPUs.

Design of Liangzi AUTO

Motivation

Consider a scenario in a P2P platform: N lenders have enabled automatic investing function. Queue $Q$ represents the automatic investing setting of these N lenders and $L$ represents a specific setting of a lender.

$$Q = \{L_1, \ldots, L_N\}$$

$$L = \{\text{min\_investing\_fund, max\_investing\_fund, investing\_cycle}\}$$

FCFS strategy works as below:

1. Form $Q$ according to the time sequences of enabling automatic investing function for all lenders.
2. When a loan project starts recruiting, the system issues $Q$ to invest according to $L$. Once concurrent head lender of $Q$ successfully invests this load project, it is arranged to the tail of $Q$.
3. Repeat (2) until the amount of this loan project is satisfied.

From the above process, we can see that FCFS strategy only consider the time of enabling automatic investing function. If a VIP lender enables the function too late, the automatic investing system can not serve him, leading to unpleasant user experience. Therefore, it is desirable to develop a novel scheduling strategy.

Dynamic scheduler

In order to design a more considerable scheduling strategy, Liangzi AUTO intends to take some important factors into consideration such as the remaining funds of a lender, tag of have been served recently, VIP ranking level and time sequence of enabling automatic investing function. Based on these four factors, we can score for each lender to dynamically decide how to arrange them in $Q$. Table 1 shows each value and relevant weight of four factors in Liangzi AUTO by default. Note that these factors, values and weights are configurable in the operation and management system of a P2P platform.
According to Table 1, we can calculate a score for each lender:

$$Score_i = \sum_{j=1}^{4} value_j \times weight_j \quad (1 \leq i \leq N)$$  \hspace{1cm} (1)

After calculating a score for each lender, we can form Q according to the scores. That’s to say, a lender with higher score is queued ahead of a lower one.

The next problem is that how much funds to automatically invest for these lenders? In Liangzi AUTO, time slice round-robin in traditional operation system scheduling algorithm is adopted. We firstly divide investing slice for each lender according to their min_invest_fund. Then we sequentially satisfy an investing slice of a lender in Q. When a circle is finished and the amount of the loan project is still not satisfied, we use a very small investing slice (100RMB) for the sake of maintaining a relative fairly scheduling until the amount of the loan project is satisfied. Algorithm 1 describes the work flow of Liangzi AUTO.

**Algorithm1. Sequential Automatic Investing**

**Input**
- Q = {L1, …, LN}, set of automatic investing information of each lender
- Loan_Amount, amount of a loan project that should be satisfied

**Output**
- Automatic investing results: R = {LR1, …, LRN}

Generate Q according to min_investing_fund, investing_cycle in L_i

For each L_i in Q
  - Calculate score, for each lender
  - Sort Q according to score
  - For each L_i in Q after sorting
    - Dequeue to issue the first round investing slice with min_invest_fund_i
  - For each L_i in Q after sorting
    - If (the amount of a loan project is not satisfied)
      - Dequeue to issue the first round investing slice with 100RMB

Repeat the above loop until there is no fund to lend or the amount of a loan project is satisfied
Parallelization

Translating Algorithm 1 to the GPU codes is straightforward: transfer the investing data to the GPU device memory; calculate scores in parallel, sort queue in parallel, invest in parallel and transfer the investing result back to the CPU main memory. The easiest implementation is that each lender is assigned to a GPU thread. In order to fully take advantage of the wide data path to GPU global memory, it is desirable to use multiples of basic data types on GPUs. Sorting on GPU is memory bound and a key to the high performance is that the sort works on groups of four-float values to lower the number of memory fetches. Thus, we can explicitly employ vector data types provided by GPUs. Using vector data type (e.g., float4), a GPU thread can issue a memory request for four float data elements in one cycle versus four separate requests in four cycles. Therefore, the number of memory accesses is significantly reduced.

Before parallel automatic investing, we first initialize thread configuration including the number of thread groups and the number of threads per thread group on the GPU. We group each four lenders into a unit. A GPU thread is responsible for only one unit. Therefore, the number of lenders is equal to the number of threads multiplies four. It is obvious that calculating scores in parallel is very simple to realize. In sorting stage, we randomly choose a pivot to divide the queue and make sure that the values of the left group scores are smaller than the pivot and the values of the right group scores are equal to or larger than the pivot. Then we employ recursion on each group to carry out quick sort until there is only one score in each group. Note that the maximum depth of recursion in CUDA is 64. It is fully adequate to support our application scenarios. After finishing the sorting stage, we assign a GPU thread to invest for four lenders. In our implementation at the investing stage, the thread with the smallest thread ID is responsible for a unit with the smallest, second smallest, third smallest and fourth smallest scores. The arrangement guarantees that the investing result is sorted by scores. Note that the number of investing times per thread is determined on the amount of the loan project and the number of lenders. As the next step, the investing results from all threads are stored into shared memory to perform adjusting stage to check whether a loan project’s satisfied amount has exceeded. If a exceeded amount is found, we adjust the investing results from the smallest thread ID. That’s to say, we reduce the final investing amount of these threads. Up to now, we have finished utilizing the massive thread parallelism of GPU for automatic investing. Note that invoking the GPU kernel is and returning the result of automatic investing are done by the CPU code.

Evaluation

In order to verify the effectiveness of parallel investing, we have conducted a set of experiments on an GTX970 GPU card using the CUDA programming language. As for the computer systems, we used an Intel i5 4590 Quad-Core processor with 8GB RAM. The operation system is Windows 7 and the IDE is CUDA SKD version 7.5. The execution time is measured in three portions: the time to read the data into GPU, the time to execute the kernel function, and the time to write the data back to the CPU main memory. The experimental results are shown in Figure 3.
We can clearly see that parallel investing is much faster than sequential version, with a maximum speedup of 37.40 times. Furthermore, it can be predicted that as the number of lenders increases, we will gain more speedup compared to sequential investing. Note that when the number of lenders is 500, we only obtain a speedup of 0.93 which is less than 1. This is mainly because there are data transfers overheads involved in the execution of a piece of code on the GPU. GPU execution will not be beneficial for all cases especially if the input data size is small. Another factor is that when the number of data items is small, sorting on GPUs does not have an advantage compared with CPUs. Fortunately, the number of lenders in large-scale automatic investing scenarios is usually much more than 1000.

Conclusions
In this paper, we present a parallel automatic investing system-Liangzi AUTO. Based on GPU parallel processing, it dynamically decides the investing sequences according to four factors at the moment of recruiting with real-time. We implement Liangzi AUTO using CUDA and obtain speedups of up to factor 37.40. In the future, we will take scattered investing into consideration to reduce credit risks for the lenders.

Acknowledgement
We would like to thank Dr. Eng. Ştefan Țălu and other anonymous reviewers for insightful feedback that will improve the quality of this paper. This research is financially supported by www.liangzijinrong.com.

References
[1] Information on www.lendingclub.com