Exploring Parallelism for Automatic and Scattered Investing on P2P Lending Platform

Gang Chen
Anhui Xiaoma Financial Information Consulting Company Limited
Room 521, Part A in Blue Business Harbor, No.188, Qianshan Road, Zhengwu District
Hefei, Anhui, China
e-mail: chengang_p2p@163.com

***Abstract*—The features of P2P lending are wide coverage, low threshold, relative simple procedure, easy to satisfy the borrowers' requirement. In order to spread credit risk for lenders, automatic and scattered investing become more and more popular nowadays. In this paper, we take advantage of the computing power of modern GPUs to accelerate scattered investing in large-scale lender scenarios. By abstracting the scattered investing as map-reduce operation, we extend two stages map-reduce to four stages according to our scattered investing business and exploit a novel parallel investing system based on GPUs. Experimental results show that we achieve a maximum speedup of 27.52 times.**

Keywords-parallel processing; GPU; P2P lending; scattered investing

I. INTRODUCTION

The development of Internet technology paves the way for reducing the cost and raising the efficiency of modern financing activities through World Wide Web. As a promising innovative Internet finance model, peer-to-peer (P2P) lending marketplace allows individuals to conduct lending and borrowing directly among each other under the intermediary services providing by P2P network lending platforms instead of traditional banking services [1]. With the rapid growth of P2P lending marketplace in recent years, more and more people participate in online transactions for borrowing or lending.

Up to date, most P2P lending platforms still only focus on the basic investing needs, and are insufficient to meet higher demand of lenders. For the sake of offering lenders more flexible investing manner, several P2P lending platforms have developed creative products to further reduce the credit risk. These products can achieve scattered investing automatically, such as Renren Dai's U program [2], which brought outstanding experiences for lenders. In large-scale lender scenarios of a P2P platform, automatic and scattered investing exhibits time criticalness with data-intensive characteristics and high performance is essential. However, such real-time requirement is limited by the overall computer system performance to accomplish investing operations for targeting lenders of the product (i.e., automatic investors) within an accepted time, because of the potential competition with manual investors.

In this paper, we investigate how to harness the parallel processing capability of GPUs to achieve performance gains with the purpose of meeting the real-time demands for scattered investing. Our contribution in this paper is: (1) abstract parallel investing processes as map-reduce operations; (2) extend two stages map-reduce to four stages according to our scattered investing business to exploit a novel parallel investing system with assistance of GPUs; (3) demonstrate the obtained speedup of the parallel scattered investing system. The rest of this paper is organized as follows: Section 2 introduces the background and related work. Implementation details of the parallel scattered investing are presented in Section 3. Experimental performance is evaluated in Section 4. Finally, conclusions are summarized in Section 5.

II. BACKGROUND AND RELATED WORK

A. P2P Lending Model

P2P network lending is a mode which brings the traditional lending transactions to the online web. Under the support of modern network technology, P2P lending platforms can offer private lenders and borrowers a more user-centric and interactive digitization of their finance operations. The whole lending process can achieve through the platform, such as choosing borrowers, investing funds, generating contracts and repaying loans. Online transactions must be completed through a kind of medium, which called the P2P lending platform [3]. From the structure of the transactions, we can find that there are three principals, which are the borrowers, the lenders (investors) and the P2P platform. The investors lend their funds to the borrower and the platforms offer service for them. In order to understand P2P lending model more easily, we abstract it as shown in Fig.1.

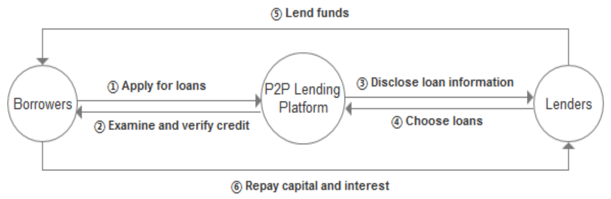
⑤ Lend funds
Borrowers
① Apply for loans
② Examine and verify credit
P2P Lending Platform
③ Disclose loan information
④ Choose loans
Lenders
⑥ Repay capital and interest

Figure 1. P2P lending model.

B. Scattered Investing

Probability of default (PD) is a financial term describing the risk that the borrower will be unable or unwilling to repay its debt in full or on time [4]. It provides an estimate of the likelihood that the default event will occur. Assuming that a lender lends funds to 10 borrowers at a time and 3 borrowers do not meet their debt obligations, hence the PD value is 30%. With respect to a lender, expected loss is the value of a possible loss times the probability of that loss occurring (i.e., PD). We can easily see that scattered lending (i.e., lend to more borrowers as possible) can decrease the potential expected loss. Paipai Dai is the first P2P networking lending platform in China. It carries out a capital guarantee program for lenders [5]: when a lender lends funds to more than 50 borrowers and each lending amount is less than 5000RMB (also must be less than one third loan amount of a single borrower), Paipai Dai guarantees the safety of the lender's capital. It is obviously that based on law of large numbers, scattered investing can work well in P2P lending platform.

C. 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 [6]. 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. A warp is the smallest scheduling unit on an SM. The threads in a warp run in lock-step, one instruction at a time. Therefore, a thread warp implicitly synchronizes at every instruction. Different thread warps execute asynchronously. This mode of parallel execution is called single-instruction multiple-thread (SIMT). Fig.2 shows Nvidia CUDA programming model. For a detailed introduction of GPU computing, we refer to Nvidia CUDA Programming Guide[7].

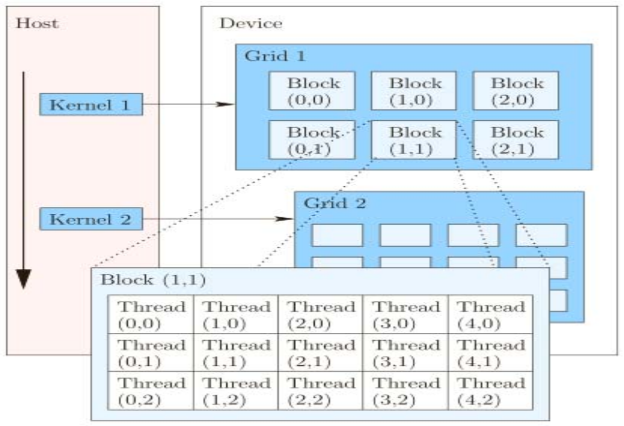

Figure 2. CUDA programming model [7].

D. GPU Architecture Overview

There is a significant body of work focusing on P2P lending, with the aim of addressing different P2P problems. To solve the problem of P2P market state modeling, Zhao et al. [8] present a focused study on market state modeling and analysis for online P2P lending. Wu et al. [9] present a recommendation model based on intelligent agent in P2P Lending for the borrowers, which helps borrower getting loan more efficiency. Vedala et al. [10] use classification algorithm to classify good and bad borrowers, where the input attributes consists of both core credit and social network information. Kumar et al. [11] 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. [12] propose an effective approach of data preprocessing namely key points approximate fitting algorithm to identify different investment patterns. Yang et al. [13] develops an integrated model to better understand the critical factors that influence lending intention through online P2P lending platform. King et al. [14] develop a pricing and data/information service system for financial analytics base on web services and GPUs.

III. PARALLEL SCATTERED INVESTING SYSTEM DESIGN

A. Sequential Implementation

Problem Description. Scattered investing system works as the following rules:

(1) The number of investing portfolio must be equal or greater than P for each lender in order to realize enough scattering.

(2) The number of right and liabilities for each borrower is as least as possible.

(3) Scattered investing strategy is relatively fair for each lender.

(4) Scattered investing strategy can satisfy at most loan requirement for a borrower.

Intuitively, the above problem looks like a multi-objective solution. However, in our business scenario strictly satisfying rule (2) – (4) is not a must and we do not intend to apply genetic algorithms to search the optimal parameters for scattered investing. A comprehensible sequential implementation can be described as algorithm 1.

Algorithm1. Sequential Scattered Investing

```
Input
$MI_{before}$ = {$MI_1$, ..., $MI_n$}, set of investing amount of each
        lender
$ML_{before}$ = {$ML_1$, ..., $ML_m$}, set of unsatisfied loan amount of
        each borrower
P, the number of investing portfolio
Output
Scattered investing results: $MI_{after}$, $ML_{after}$
for each $MI_i$ in $MI_{before}$
    slice = $MI_i$ % P, note that the remainder is also a slice
array element
endfor
Sort slice array according to ascending order and enqueue
$Q_1$
Sort $ML_{before}$ according to descending order and enqueue $Q_2$
for each element in $Q_2$
    dequeue element in $Q_1$ (we called issue) to satisfy the
loan
    if the loan (i.e., head element of $Q_2$) is satisfied one
time
```

```
            check whether the issued slice element has remaining amount
                if has remaining amount
                    continue to try to satisfy next loan in $Q_2$
                endif
        endif
        else continue to issue slice element in $Q_1$ to satisfy the loan
        endif
        Repeat the above loop until there is no fund to lend (i.e., each slice element in $MI_{after}$ is 0) or all the loans are satisfied (i.e., each loan element in $ML_{after}$ is required loan amount)
endfor
```

From the above sequential algorithm, we can see that there exhibits parallelism in scattered investing process. However, as GPUs are specially designed for massive data-parallel computing, their performance is subject to the presence of condition statements. When a GPU kernel runs, on a conditional branch where the threads diverge in which path to take, the threads taking different paths have to run serially, causing serious performance degradations [15]. In order to avoid the performance limitation of conditional branch in our sequential algorithm, we have to reform our algorithm from another perspective.

B. *Abstract Scattered Investing to Map-Reduce Procedure*

As a domain-specific programming model, map-reduce is a parallel processing paradigm for large scale data proposed by Google on distributed clusters [16]. Developers only require to define the associated map-reduce operations and the map-reduce runtime deals with of partitioning, inter-communication, task scheduling and fault tolerance. Map-reduce provides two primitive operations: map function processes input key/value pairs (key_1, $value_1$) and generates intermediate key/value pairs (key_2, $value_2$), and reduce function merges all intermediate pairs associated with the same key_2 to form a list of $value_3$. The two primitive operations are defined as follows:

$$\text{Map: } (key_1, value_1) \rightarrow \text{list } (key_2, value_2)$$
$$\text{Reduce: } (key_2, \text{list } (value_2)) \rightarrow \text{list } (value_3)$$

As mentioned earlier in Algorithm 1, we issue Q_1 and satisfy Q_2 sequentially. Assume that we define slice element in Q_1 two attributes, namely s_lender and s_amount, and define loan element in Q_2 two attributes, namely l_id and l_amount, the scattered investing operations can be converted to map-reduce operations as follows:

$$\text{Map: } (s\_lender_i, s\_amoun_{ti}) \rightarrow \text{list } (l\_id_j, s\_amount_k)$$
$$\text{Reduce: } (l\_id_j, s\_amount_k) \rightarrow \text{list } (l\_id_j, s\_amount_j)$$

Intuitively, we can direct adopt the above map-reduce operations to parallelize scattered investing. However, as map-reduce is domain-specific, we have to rethinking it according to our actual business. For example, before we start map operation, we must generate slices for each lender. More importantly, after reduce operation we must ensure each loan being invested does not exceed its required funds. Therefore, we extend traditional two-stage map-reduce as four stages, namely Slice (S), Map (M), Reduce (R) and Adjust (A). In fact, stage S can be incorporated into stage M. We isolate these two stages because one lender have the possibility of investing more than once with respect to the same product. When stage S is finished, generated slices can be directly accepted by M stage. When the whole map-reduce operations are finished, we launch stage A to adjust the loan amount of each loan. Hence, our four stages map-reduce operations can be depicted as follows:

$$\text{Slice: } (s\_lender_i, P) \rightarrow \text{list } (s\_lender_i, s\_amount_k)$$
$$\text{Map: } (s\_lender_i, s\_amount_i) \rightarrow \text{list } (l\_id_j, s\_amount_k)$$
$$\text{Reduce: } (l\_id_j, s\_amount_k) \rightarrow (l\_id_j, \text{list } (s\_amount_j))$$
$$\text{Adjust: } (l\_id_j, \text{list } (s\_amount_j)) \rightarrow \text{list } (l\_id_j, s\_amount_j)$$

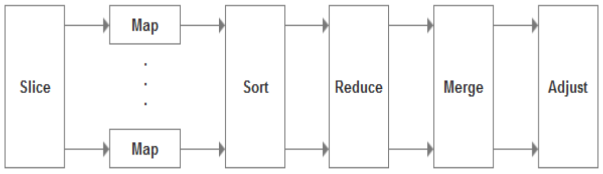

Figure 3. Work flow of four-stage processing model.

Fig. 3 depicted the work flow of our four-stage processing model. Extended four stage processing model can handle the basic scattered investing. However, just running a single pass of four-stage procedure may not address some problem. For example, when we have finished stage A, we find that some borrowers' loan requirements are not satisfied. At the same time, there are some new lenders come to invest. In this case, we need to iterate another pass to deal with their scattered investing. Therefore, we define each stage can be assembled dynamically before terminate condition is satisfied (i.e., there is no left amount of funds of the product), making it flexible to handle complicated situations, as shown in Fig. 4.

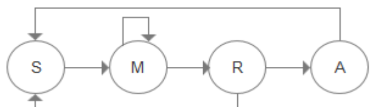

Figure 4. Dynamically assembled of each stage.

C. *Parallel Implementation*

CPU-based Map-reduce implementation is straightforward: Firstly, slice the investing data and map them to generate loan/amount pairs. Then, sort these pairs by load_id and reduce all the same loan_id pairs and gather the final output. Finally, adjust the exceeded amount loads. Translating CPU-based implementation to the GPU version is also direct: transfer the investing data to the GPU device memory; make Slice Map, Sort and Reduce kernel calls; and transfer the investing result back to the CPU main memory.

Because current GPUs do not support dynamic thread scheduling effectively, we need to distribute map-reduce work evenly across threads on the GPU to efficaciously exploit abundant thread parallelism[17]. The easiest implementation is that each slice data in Map and Reduce is assigned to a GPU thread, and processing many slice data items with a single kernel call. In order to fully take advantage of the wide data path to GPU global memory, which is on the order of hundreds of bits, it is desirable to use multiples of basic data types on GPUs. Hence, it is possible for to read adjacent address of block data when GPU threads are executing simultaneously [18]. 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 scattered investing, we first initialize thread configuration including the number of thread groups and the number of threads per thread group on the GPU. In the map stage, we group each four slice pairs into a block. A GPU thread is responsible for only one block. After finishing the map stage, we sort the intermediate (loan_id, invested amount) pairs so that the pairs with the same loan_id are stored consecutively. In the reduce stage, we also group the sorted intermediate pairs into a block and make the pairs with the same loan_id belong to the same block. Therefore, the number of blocks is equal to the number of threads multiplies four. In our implementation, the thread with the smallest thread ID is responsible for a block with the smallest, second smallest, third smallest and fourth smallest loan_ids. The arrangement guarantees that the investing result of the reduce stage is sorted by loan_id. As the next step, the investing results from all threads are stored into shared memory to perform adjust stage. In adjust stage, each thread is responsible for checking whether a loan's satisfied amount has exceeded or has not reached its required amount. If a exceeded loan is found by a thread, this thread writes to the pair (lender, excessive amount) to a specified location in shared memory. On the other hand, if an unsatisfied loan is found by a thread, this thread reads pair (lender, excessive amount) to the above specified location and updates its pair. CUDA function __threadfence() makes this inter-thread communication comes true. After adjust stage, the final result of scatter investing has been generated. Up to now, we have finished utilizing the massive thread parallelism of GPU for our scattered investing task. Our detailed four-stage parallel scattered investing process is depicted in algorithm 2. Note that invoking the GPU kernel is and returning the result of scattered investing are done by the CPU code. The input and output of algorithm 2 is as the same as algorithm 1.

Algorithm2. Parallel Scattered Investing

```
Copy lender investing data from the CPU main memory to the GPU global memory
Invoke kernel slice;
Invoke kernel map and sort the intermediate pair by loan_id
Invoke kernel reduce and store the result into the shared memory
Invoke kernel adjust
Copy the final scattered investing result back to the CPU main memory
```

IV. Evaluation

In order to verify the effectiveness of parallel scattered investing, we have conducted a set of experiments on an GTX970 GPU card using the CUDA programming language. This GPU consists of 1664 cores clocked and 4GB GDDR5 memory. 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 results reported are based on the system time measurements, collected from the program execution, using the processor's hardware performance counters. The time is measured in three portions: the time to read the data into the card, the time to execute the function, and the time to write the data back to the system's memory. The experimental results are shown in Fig. 5. We can clearly see that parallel scattered investing is much faster than sequential version, with a maximum speedup of 27.52 times. Furthermore, it can be predicted that as the number of lenders increases, we will gain more speedup compare to sequential scattered investing.

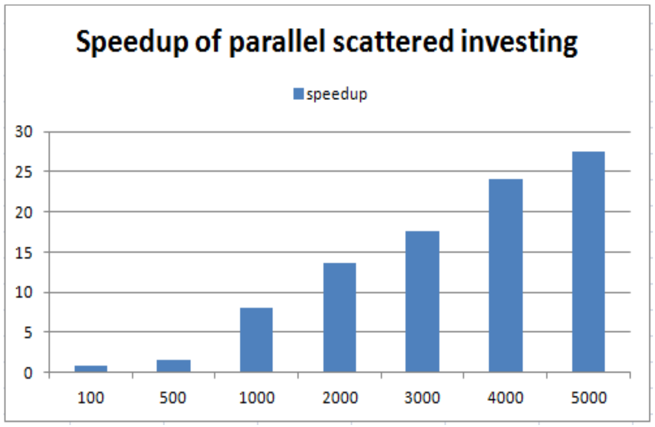

Figure 5. Speedup of parallel scattered investing.

Note that when the number of lenders is 100, we only obtain a speedup of 0.75 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 the input data size is small. Fortunately, the number of lenders in large-scale automatic and scattered investing scenarios is usually more than 1000.

V. Conclusion

In this paper, we have introduced a parallel scattered investing system by extending two stages map-reduce to four stages based on our business character. We implement parallel scattered investing using CUDA and obtains speedups of up to factor 27.52. We do not use the shared memory of GPUs in map stage and reduce stage because the number of slice may be very large and the size of all slices may exceed the size of shared memory. In order to fully take advantage of GPU memory hierarchy, we will develop shared memory staging strategy in the future.

Acknowledgment

We would like to thank the anonymous reviewers for insightful feedbacks that will improve the quality of this paper. This research is financially supported by www.liangzijinrong.com.

References

[1] N. Chen, A. Ghosh, N. Lambert. Social Lending. In Proceedings of the ACM Conference on Electronic Commerce, 2009, pp335-344.

[2] Renren Dai website. www.we.com

[3] H. Wang, M. Greiner ,J.E. Aronson. People-to-People lending: the emerging E-commerce transformation of a financial market. Lecture Notes in Business Information Processing, 2008, 36:182-195.

[4] M. Kim. Downturn LGD，Best Estimate of Expected Loss，and Potential LGD under Basel II. Journal of Economic Research. 2006, 11(2): 203-223.

[5] Paipai Dai website. www.ppdai.com

[6] J. Nickolls, W. J. Dally. The gpu computing era. IEEE Micro, 2010, 30(2): 56-69.

[7] NVIDIA. CUDA C Programming Guide. Available online at http://docs.nvidia.com/cuda/cuda-c-programming-guide

[8] H. K. Zhao, Q. Liu, H.S. Zhu, et al. A Sequential Approach to Market State Modeling and Analysis in Online P2P Lending. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017, Issue 99, pp1-13.

[9] J. H. Wu, R. Fu. An Intelligent Agent System for Borrower's Recommendation in P2P Lending. In proceedings of the International Conference on Multimedia Communications (Mediacom), 2010, pp179-182.

[10] R. Vedala, B. R. Kumar. An application of Naive Bayes classification for credit scoring in e-lending platform. In proceedings of the International Conference on Data Science & Engineering (ICDSE), 2012, pp81-84.

[11] V.L. Kumar , S. Natarajan, S. Keerthana, et al. Credit Risk Analysis in Peer-to-Peer Lending System. In proceedings of the International Conference on Knowledge Engineering and Applications (ICKEA), 2016, pp193-196.

[12] F. Shen, N.L. Luo. Investment pattern clustering based on online P2P lending platform. In proceedings of the International Conference on Computer and Information Science, 2016, pp1-6.

[13] Q. Yang, Y-C. Lee. Critical Factors of the Lending Intention of Online P2P: Moderating Role of Perceived Benefit.In proceedings of the International Conference on Annual International Conference on Electronic Commerce-e-Commerce in Smart connected World, 2016, pp1-8.

[14] G.H. King，Z.Y. Cai，Y.Y. Lu, et al. A High-Performance Multi-user Service System for Financial Analytics Based on Web Service and GPU Computation. In proceedings of the International Symposium on Parallel and Distributed Processing with Applications, 2010, pp327-333.

[15] A. Bakhoda, G. L. Yuan, W. W. L. Fung, et al. Analyzing CUDA workloads using a detailed GPU simulator. In Proceedings of the Symposium on Performance Analysis of Systems and Software, 2009, pp163-174.

[16] J. Dean, S. Ghemawat. Mapreduce: simplified data processing on large clusters. In proceedings of the Conference on Symposium on Operating Systems Design & Implementation, 2004, 51(1), pp137-150.

[17] B. He, W. Fang, Q. Luo, N. K. Govindaraju, et al. Mars: a mapreduce framework on graphics processors. In Proceedings of the International Conference on Parallel Architecture & Compilation Techniques, 2008, pp260-269.

[18] S. Ryoo, C. I. Rodrigues, S. S. Stone, et al. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In Proceedings of the 13th ACM symposium on principles and practice of parallel programming, 2008, pp73-82..