In-GPU Cache for Acceleration of Anomaly Detection in Blockchain

Shin MORISHIMA† and Hiroki MATSUTANI††, Members

SUMMARY  Blockchain is a distributed ledger system composed of a P2P network and is used for a wide range of applications, such as international remittance, inter-individual transactions, and asset conservation. In Blockchain systems, tamper resistance is enhanced by the property of transaction that cannot be changed or deleted by everyone including the creator of the transaction. However, this property also becomes a problem that unintended transaction created by miss operation or secret key theft cannot be corrected later. Due to this problem, once an illegal transaction such as thief occurs, the damage will expand. To suppress the damage, we need countermeasures, such as detecting illegal transaction at high speed and correcting the transaction before approval. However, anomaly detection in the Blockchain at high speed is computationally heavy, because we need to repeat the detection process using various feature quantities and the feature extractions become overhead. In this paper, to accelerate anomaly detection, we propose to cache transaction information necessary for extracting feature in GPU device memory and perform both feature extraction and anomaly detection in the GPU. We also propose a conditional feature extraction method to reduce computation cost of anomaly detection. We employ anomaly detection using K-means algorithm based on the conditional features. When the number of users is one million and the number of transactions is 100 millions, our proposed method achieves 8.6 times faster than CPU processing method and 2.6 times faster than GPU processing method that does not perform feature extraction on the GPU. In addition, the conditional feature extraction method achieves 1.7 times faster than the unconditional method when the number of users satisfying a given condition is 200 thousands out of one million.

key words: Blockchain, anomaly detection, GPU

1. Introduction

Blockchain is a distributed ledger system composed of P2P network proposed by Bitcoin [1]. In Blockchain, sender and receiver can trade directly without trusted third party such as a bank, unlike existing online money transfer system. The most widely-used applications of Blockchain are crypto currencies. They are used for international transactions because of low cost and fast transactions by the direct transactions. Blockchain has various features, such as fault tolerance, tamper resistance, and anonymity. Applications of the crypto currencies are conservation of assets utilizing fault tolerance and personal transactions that can protect personal information by anonymity. In addition, Blockchain is used for not only crypto currencies but also asset transactions other than currency [2], [3], distributed application [4], [5], document storage system, and registration of land.

The data structure of Blockchain is a chain of hash values of blocks each of which contains a set of transactions. In this structure, an update of a transaction causes changes in all the blocks after the block containing the transaction. With this structure, update or delete of a Blockchain transaction is extremely difficult by everyone including the creator of the transaction; thus tamper resistance is high. However, this feature becomes a problem in which Blockchain system cannot modify fraudulent transactions made by miss operations or stolen secret keys. Because of this problem, once an illegal transaction, such as theft, occurs, the damage will expand. To suppress the damage, we need countermeasures, such as detecting illegal transaction at high speed and correcting the transaction before approval.

Pham et al. proposed an anomaly detection method for Blockchain transactions by using K-means clustering, Mahalanobis distance, and local outlier factor [6]. Using this method, actual theft transactions in past transactions of Bitcoin network were detected. However, abnormal transactions that can be detected by specific feature quantities and algorithms are limited. In reality, we need to repeat anomaly detections by changing feature quantities and algorithms, resulting in heavy computations and long computation time. Figure 1 shows the execution time when anomaly detection is executed using K-means algorithm with four feature quantities (similar approach to [6]).

![Figure 1 Execution time when anomaly detection is executed using K-means algorithm with four feature quantities](Image)
block creation time of Ethereum [4] which is one of typical Blockchain systems. The number of users of one million is almost equal to the number of holders of one Bitcoin (equivalent to the market price about 8,000 dollars) in the Bitcoin network [7] (in actual operation, more users may be targeted). This result shows that the time for a single anomaly detection exceeds the block creation time, which means that multiple anomaly detections cannot be executed within the creation time. In this paper, to accelerate anomaly detections, we propose to cache transaction information necessary for extracting features in GPU device memory and perform both the feature extraction and anomaly detection in the GPUs. Specifically, we propose to cache a graph structure representing transaction status of users as an array-based data structured suitable for GPU processing. By extracting a feature quantity from a graph as an information source, both the feature extraction and anomaly detection can be executed in the GPUs.

The rest of this paper is organized as follows. Section 2 surveys related work. Section 3 illustrates our proposed In-GPU cache based anomaly detection method for Blockchain network. Section 4 shows evaluation results and Sect. 5 concludes this paper.

2. Related Work

2.1 Data Structure of Blockchain

Figure 2 shows an overview of data structure of Blockchain. Upper half of the figure shows the overall Blockchain structure consisting of a chain of blocks. A block is a set of transactions, and detail of a transaction is shown in lower half of the figure. A block contains block ID, ID of previous block, and set of transactions. A block ID is a hash value of its entire block excluding the block ID. Because a block contains the previous block ID, the block is connected to the previous block by the hash value. The connection continues back to the genesis block, and the structure looks like a chain; so it is called Blockchain. In other words, each block ID affects all the subsequent blocks, and an update of a transaction causes changes of all blocks after the block containing the transaction. This feature enhances tamper resistance, because if an attacker tampers a transaction, the attacker must tamper all blocks after the block containing the transaction. Lower half of the figure shows data structure of a transaction. The structure of transaction is similar to the structure of block. That is, it contains transaction ID and transaction detail. A transaction ID is a hash value of its entire transaction excluding the transaction ID.

The write query of Blockchain is always used to generate a new block appended as the latest block of the chain. This means that past blocks and transactions in Blockchain are not updated nor deleted. Blockchain system employs a consensus algorithm, and a block is generated only when the condition based on the algorithm is satisfied, in order to exclude malicious network participants that generate selfish blocks. Regarding the consensus algorithm, several algorithms are proposed, such as PoW (Proof of Work), PoS (Proof of Stake), and PBFT (Practical Byzantine Fault Tolerance) [9].

For approval of a transaction in Blockchain system, the conditions based on the consensus algorithm must be satisfied, and once approved, the transaction cannot be changed even by a creator of the transaction nor system administrators. In addition, illegal transactions (e.g., those by theft secret keys) are not matters of the protocol; thus they cannot be canceled at the time of approval by the consensus algorithm. Please note that there is a time period for a transaction to satisfy the consensus condition (approval of the transaction) after creation of the transaction. Such illegal transactions can be corrected if they are detected during this period. The motivation of this paper is, thus, to detect abnormalities at high speed and high accuracy in order to suppress damage of the illegal transactions.

2.2 Blockchain Search Using In-GPU Cache

In [10], we proposed an acceleration method of Blockchain transaction search using in-GPU cache. In Blockchain systems, most users use Blockchain indirectly via services provided by full nodes, such as exchanges and wallet applications. Therefore, most Blockchain search queries are concentrated to a part of full nodes that provide the services, and thus the search queries at the full nodes become a bottleneck of Blockchain system. We proposed an array-based tree structure taking advantage of GPU processing and the Blockchain-specific characteristics where past transactions are not updated nor deleted. The Blockchain transaction search was accelerated by 3.4 times faster than existing method, which is a general purpose GPU search††. Specifically, there is a possibility that a block is disabled due to a chain split or reconstruction. However, as a block becomes older, probability of invalidation becomes lower. In Bitcoin system, the probability that the 6th block before the latest block is invalidated is considered to be almost zero.
based on key-value store structure [11], [12]. By using the Blockchain transaction search [10] in combination with the Blockchain anomaly detection method proposed in this paper, an entire Blockchain system can be accelerated, because anomaly detection is executed at full nodes that provide the services.

2.3 Anomaly Detection on Blockchain

A method for detecting abnormal transactions in the Blockchain is proposed in [6]. The method uses the user graph that represents user-centric transactions flow. Figure 3 shows an overview of the user graph, where a user is a vertex and a transaction is an edge. As shown in the figure, when there is a transaction from B to A, an edge from B to A is created. The graph structure is expanded by new transactions represented as edges. This user graph is used to extract feature quantities for each Blockchain user. For example, the number of edges that enter a certain vertex is the number of deposit transactions, and the number of edges that leave represents the number of withdrawal transactions. They used six feature quantities, such as the number of deposit transactions, the number of withdrawal transactions, average deposit amount, and average withdrawal amount, for the anomaly detection. Also, they used three anomaly detection algorithms, i.e., K-means clustering, Mahalanobis distance, and Local Outlier Factor. By using the user graph based method, two actual theft cases on June and October in 2011 at Bitcoin network were detected. Please note that many abnormal transactions other than these transactions were detected by the method. However, it is very difficult to judge if they are really stolen cases, because information about most stolen cases would not be disclosed.

The detected theft cases are varied depending on the anomaly detection algorithm used. This means that we need to repeat anomaly detections by changing the algorithms. In addition to the six feature quantities used for the experiment, if we used another feature quantity such as balance, different abnormalities would be detected. The number of all the past transactions in Blockchain is huge and the computation cost for anomaly detection algorithms increases as the number of input samples increases. Thus, in reality, feature quantities are extracted from a range of transactions selected by a given condition for practical anomaly detection. For example, a range of transactions which were created within one year or recent transactions whose cumulated size is less than a certain limit can be used for the feature quantity extraction. Due to various choices of the conditions, computation cost for the transaction extraction becomes large. In this paper, both the feature extraction and anomaly detection are accelerated by GPUs. As a representative algorithm, K-means clustering is implemented and evaluated for the anomaly detection of Blockchain. Please note that our proposal method can be applied to the above algorithms other than K-means clustering.

3. Anomaly Detection Using In-GPU Cache

3.1 Overview of Proposed System

All the transaction information of Blockchain cannot be stored in the GPU device memory which is much smaller than host memory. For example, NVIDIA GeForce GTX 980 Ti has 6GB device memory, while size of Bitcoin transaction data is about 160GB. There are two general approaches for the anomaly detection with a limited GPU device memory. One approach is to repeat the processes in which a part of transactions are sent to GPU and anomaly detection on that part is performed using the GPU. Another approach is that feature quantities are extracted by CPU and then anomaly detection is done by GPU. For the first approach we need to consider the overhead for CPU-GPU data transfers, and for the second approach the issue is high computation cost for feature extractions by CPU. On the other hand, in this paper, we propose a method to perform both the feature extraction and anomaly detection on the GPU. In the proposed method, to extract feature quantities by GPU, a modified user graph that has information of transactions necessary for the extraction is used. The user graph can be cached in the GPU device memory, because the size of user graph can be reduced by eliminating some information unnecessary for feature quantity extraction, such as signature of the sender. By caching the user graph in the GPU device memory, both the feature extraction and anomaly detection can be performed by the GPU.

Figure 4 shows an overview of anomaly detection sys-
tem using the In-GPU cache. As shown in the left side of the figure, all the transaction information is stored in the host main memory. The user graph is created based on the information of the sender and receiver, the necessary information for feature extraction is added to the graph, and the graph is cached in the GPU device memory. Detail about the graph creation method is described in Sect. 3.2. As shown in the right side of the figure, various feature quantities are extracted from the graph cached in the GPU device memory, and then anomaly detection is performed by the GPU. When a different algorithm for anomaly detection is performed using the same set of features, the extracted features can be reused to reduce the overhead of feature extraction. If an anomaly detection is performed using different features but free space to extract new features is not available in the GPU device memory, then old feature quantities are deleted from the In-GPU cache. Thus, both the feature extraction and anomaly detection are performed by the same GPU in the system. The proposed system can accelerate the anomaly detection of Blockchain compared to naive approaches that impose CPU-GPU data transfer overhead or heavy computations for feature extraction by CPU.

### 3.2 Data Structure of User Graph in GPU

Figure 5 shows the data structure of a user graph cached in the GPU device memory. The upper part of the figure shows transaction information in the host. At first, a user graph is created based on the sender and the receiver information from the original transactions. The graph created from the information is shown in the left side of the middle figure. In this graph, a vertex represents a Blockchain address, and an edge represents a transaction from the sender to the receiver. If a transaction includes multiple senders and receivers, edges are created for all the combinations of sender and receiver pairs.

The data structure for caching the graph in the GPU is shown in the lower left side of the figure. The graph is represented using a compressed row storage structure called CSR (Compressed Sparse Row) to be efficiently accessed by the GPU. CSR usually consists of two arrays: an array representing the destinations of edges (DST array) and that pointing sources of the edges (PTR array). The destination of an edge represents a receiver of the transaction and the source represents a sender. In the proposed system, because a receiver can be found by referring to the transaction content, the transaction ID is stored in the DST array instead of the destination of the edge; so we call the array DSTID in this paper. Using these arrays, a withdrawal information from a sender can be found by searching from the PTR array, because transaction information is stored in these arrays in the sender’s point of view. If the PTR array points outside of the DSTID array, it means there is no withdrawal from that vertex. On the other hand, an overall graph search is needed to obtain a deposit information to a receiver. In the proposed system, two additional arrays, where directions of edges in the PTR and DSTID arrays are reversed, are introduced to search a deposit information efficiently. Specifically, SRCID array and its PTR array are added. When we distinguish PTR arrays pointing to DSTID array and SRCID array. The array pointing to DSTID is called DSTPTR and the array pointing to SRCID called SRCPTR. That is, the SRCID array represents senders of transactions and SRCPTR array points receivers of the transactions. Using these arrays, a deposit information to a receiver can be found by searching from the SRCPTR array. In this structure, thus, deposit and withdrawal information can be searched bidirectionally by using the DSTID array for the withdrawal and the SRCID array for the deposit information.

In the example of Fig. 5, the character of each node of the graph means the user, and the number written beside each edge means the transaction ID. In the CSR, each element of PTR array corresponds to the user. For example, the first element of PTR array corresponds to user A, and second element corresponds to user B. User A has three adjacent edges. The edges are one deposit transaction (transaction 0) and two withdrawal transactions (transaction 1 and 2). Corresponding to this, the DSTID array of user A has two elements which have ID 1 and 2, and SRCID array of user A has one element which has ID 0. By repeating this process, the CSRs are created.

Regarding transaction information, only the items related to feature extraction are stored in the In-GPU cache. A cached transaction information is shown in the right side of the middle figure. In this example, the sender, transaction amount, and transaction time are cached. Besides this,
Fig. 6  Processing flow of extracting feature quantities from In-GPU cache

we can cache other information, such as block number, balance, and so on. As the information stored in the In-GPU cache is increased, although the data size increases, various features can be extracted. We can decide which information to be extracted, by considering the capacity of the GPU device memory. This content is created for each edge in the user graph, not for each transaction. That is, when there are multiple senders and receivers in one transaction, all the combinations of sender and receiver pairs are created as edges. The structure is represented as a matrix, where column index is used as transaction ID. Each column represents transaction contents on an edge (i.e., a transaction ID) of the user graph, and each row represents a specific value of the transaction contents such as transaction amount. Let \( i \) be the column index of an interested transaction ID in the matrix. By using \( i \) as the transaction ID in the DSTID and SRCID arrays, the transaction contents on each edge can be obtained from the user graph. The proposed method to extract feature quantities used for anomaly detection using this structure will be described in the next subsection.

3.3 Feature Extraction in User Graph

Figure 6 shows a flow chart for extracting feature quantities from the In-GPU cache with the structure described in Sect. 3.2. First of all, it is judged whether the feature quantities to be extracted are related to deposit or withdrawal. The graph structure of the SRCID array is used in the case of a deposit and that of the DSTID array is used in the case of a withdrawal. Next, GPU threads are launched according to the number of vertices (i.e., the number of users), and each thread searches a single element of the PTR array in parallel. Each thread refers to the SRCID or DSTID array from the corresponding PTR array and acquires a transaction ID representing a column of the transaction content matrix. Then, each thread refers to the row of the transaction content matrix according to the condition, acquires the element from the row related to the feature quantity if the condition is met, and aggregates the feature quantities on the basis of this element. By repeating these steps (4) and (5) in Fig. 6 on all the edges adjacent to the vertex, the feature quantities based on all transaction information related to a specific user can be extracted. These steps represent the processing for extracting one feature quantity. By repeating these steps for each feature quantity, multiple features necessary for anomaly detection can be extracted. When multiple features are necessary, the conditional judgement is performed only in the extraction steps for the first feature; in the extraction steps for the other features, the conditional judgement results for the first feature can be reused. For example, the number of deposits and the average value of the deposits of each user are the features of the extraction, and the condition requires that the transaction value is greater than a threshold. In this case, when the number of deposits is extracted, the conditional judgement of each transaction can be reused, the conditional judgement of each transaction is performed. The result of the judgement can be used for the extraction of the average value of the deposits. Therefore, when the average values are extracted, the judgement is not needed.

Figure 7 shows an example of feature extraction described in Fig. 6. In this example, an average of all the deposit amounts which are greater than 7 is extracted. Each number in the figure is corresponding to each step in the flow chart shown in Fig. 6. First, since the target is payment, the DSTID array is used as user graph. Next, a thread is launched for each element of the PTR array, and each element of the DSTID array is referred to from the PTR array. Since each content of the DSTID array is the transaction ID that represents the column number of the transaction content matrix, the deposit amount can be extracted from the column on the matrix. If the deposit amount is greater than 7, the total deposit amount and the number of deposit transactions are summed up. These steps are repeated until all the edges adjacent to the vertex in charge are completed. Finally, the average deposit amount is obtained by dividing the total deposit amount by the deposit number. Although only one thread is shown in detail in the figure, each thread executes these steps in parallel at all the vertices. When the processing of all the threads is completed, an array can be created that represents an average deposit amount for each user. In the following description, this array is denoted as “feature array”. The number of elements in the feature array.
3.4 Acceleration of Anomaly Detection by Conditional Feature Extraction

As mentioned in the previous section, anomaly detection algorithm is processed for feature arrays. The array size is determined by the number of users because each feature is extracted from all the users. Therefore, the computation cost increases as the number of users increases. In other words, if the number of users which are subject to feature quantity extraction is reduced, anomaly detection can be accelerated. Conditional feature extraction can reduce the number of users by not extracting features of users who do not satisfy the condition.

Figure 8 shows an example of the conditional feature extraction to reduce the feature array size. The condition and user graph are same as Fig. 7. In this figure, the feature array has only two elements even though the number of users is five. There are two transactions (i.e., transactions 3 and 4) from user C, while their transaction values are less than 7. So these features are not extracted. The anomaly detection can be accelerated by reducing the number of elements because computation cost of anomaly detection depends on the number of elements.

Let us introduce another example of the conditional feature extraction. As for Bitcoin balance, users with a lot of balance should be focused in the anomaly detection because a damage of an illegal transaction depends on the balance. For example, it is conceivable to extract users who have 1 Bitcoin or more. Figure 9 shows proportion of addresses (users) for three ranges of Bitcoin balances. As shown, the proportion of users having more than 1 Bitcoin is about 3.1%. This means that if the condition of extraction is “users who have more than 1 Bitcoin”, the size of feature arrays can be reduced to only 3.1% after the extraction.

3.5 Anomaly Detection Using Extracted Feature Quantities

By executing various anomaly detection algorithms for the feature quantities extracted in Sect. 3.3, anomaly users in Blockchain can be detected. In this paper, as an example of the algorithms, an anomaly detection using K-means algorithm is implemented and evaluated. K-means is a well-known clustering algorithm that classifies a large number of data into clusters. When it is used as anomaly detection, feature quantity of a user that is far from the center of gravity of the cluster is regarded for abnormal after the K-means clustering. The anomaly detection using K-means clustering when the number of clusters is $k$ is done by the following steps.

1. An initial cluster is randomly assigned to the feature quantity vector $x_i (i = 1, \ldots, n)$ of each vertex.
2. Calculate the cluster center of gravity $V_j (j = 1, \ldots, k)$.
3. Calculate the distance between each $x_i$ and each $V_j$.
   Then $x_i$ belongs to the cluster $j$ of the nearest $V_j$.
4. Steps 2 and 3 are repeated until the process is converged. The convergence is detected when the number of iterations or total amount of changes falls below a given threshold. Then the clustering is completed.
5. Calculate the distance between $x_i$ and the center of gravity $V_j$ of its cluster. $x_i$ is detected as abnormal if the distance exceeds a given threshold.

When using the feature extraction method described in Sect. 3.3, the feature arrays are created as an array structure, which is suitable for GPU processing. Then GPU threads are launched according to the number of elements (i.e., number of vertices) of the array. Since each thread processes an element in parallel, the procedure can be accelerated by the GPU. Specifically, in the step 2, the center of gravity is obtained by calculating the average of the feature quantities of the respective clusters by the reduction operation. In the distance computations in the steps 3 and 5, the distance for each vertex is computed by each thread in parallel. These steps consist of numerical operations using the array structure. Even if we employ anomaly detection algorithms other than K-means clustering, a similar parallel processing can be applied by changing the operations.
4. Evaluations

4.1 Evaluation Environment

All evaluations are conducted with the same machine. The processor is Intel Xeon E5-2637v3 running at 3.5GHz and memory capacity is 256GB. A single NVIDIA GeForce GTX 980 Ti GPU is used for the GPU processing. Table 1 lists the specification of the GPU. We use CUDA version 7.5. The compiler is nvcc version 7.5.17, and the compile option is “nvcc -std=c++11 -arch=sm_5”. To evaluate the scalability of the proposed system, we use synthetic transaction data in which transactions are generated at random by changing the average number of transactions per user from 20 to 100. We evaluate the number of users as 500 thousands and one million. K-means algorithm is used for the anomaly detection. Four feature quantities are examined: payment number, withdrawal number, average deposit amount, and average withdrawal amount. The following three methods are evaluated in terms of the execution time.

- GPU only: The proposed method using In-GPU cache with feature extraction and anomaly detection by GPU.
- CPU only: A method extracting feature quantities and detecting abnormalities by CPU.
- CPU + GPU: A method extracting feature quantities by CPU, transferring the feature quantities to GPU, and performing anomaly detection by the GPU.

For fairness of evaluation, we create a user graph capable of extracting feature quantities on the host memory to extract feature quantities even when extracting features using the CPU. In addition, to evaluate the influence of adding a condition at the time of feature quantity extraction, we evaluated two cases: cases with and without a condition for extracting feature quantities. In the CPU processing at CPU and CPU + GPU, we use four cores in parallel because the CPU has four cores.

4.2 Execution Time of Anomaly Detection Using Feature Quantities without Condition

Figures 10 and 11 show the execution time in the case of extracting feature quantities without conditions and performing anomaly detection. The numbers of users are 500 thousands and one million in Figs. 10 and 11, respectively. Based on [6], the number of clusters for the K-means algorithm is seven and the number of repetitions of cluster division is 50 in both the cases. As shown in Fig. 10, both CPU + GPU and GPU only are faster than CPU only. However, as the average number of transactions increases, the execution time of CPU + GPU increases, and the speed-up rate of the proposed method increases. Since the features extraction process is performed for each transaction, the time for the feature extractions is proportional to the average transaction number. On the other hand, since the number of extracted features does not change unless the number of users changes, the execution time of anomaly detection does not change.

Comparing Fig. 10 and Fig. 11, the number of users increases and the total transaction volume also increases, so the execution time increases both in feature extraction and in anomaly detection. The tendencies, such as the speed-up rate, are almost the same in Figs. 10 and 11. When the number of users is 500 thousands and the average number of transactions is 100, the proposed method is 8.5 times faster than the CPU only and 2.6 times faster than CPU + GPU. When the number of users is one million and the average number of transactions is 100, the proposed method is 8.6 times faster than CPU only and 2.6 times faster than CPU + GPU.

Table 1  GPU specification used in the experiments

|                                    | GeForce GTX 980 Ti |
|------------------------------------|--------------------|
| Number of cores                    | 2,816              |
| Core clock                         | 1,038MHz           |
| Memory clock                       | 7,010MHz           |
| Memory datapath width              | 256bit             |
| Memory capacity                    | 6GB                |

Fig. 10  Execution time in the case of extracting feature quantities without condition and performing anomaly detection for 500 thousands users

Fig. 11  Execution time in the case of extracting feature quantities without condition and performing anomaly detection for one million users
4.3 Execution Time of Anomaly Detection Using Feature Quantities with Condition

Figures 12 and 13 show the execution time when anomaly detection is performed by extracting the features with the condition that the transaction volume is equal to or more than the threshold value. The transactions are adjusted so that all the users have at least one transaction which satisfies the condition, in order to perform anomaly detection with the same number of users. In Sect. 4.6, the execution time of anomaly detection is evaluated when the number of users is decreased by the conditional feature extraction. The numbers of users are 500 thousands and one million in Figs. 12 and 13, respectively. As shown in Fig. 12, the two GPU based methods (GPU only and CPU + GPU) are faster than the CPU only case, as well as the unconditional case (Fig. 10). Also, as for the CPU + GPU case, the execution time increases as the average transaction number increases.

However, by setting conditions, the computation cost for feature extraction increases, so the increase in execution time is larger than the case without condition. The same tendency can be seen in Fig. 13 where the number of users is one million. As a result, when the number of users is 500 thousands and the average number of transactions is 100, the proposed method is 10.8 times faster than the CPU only and 5.6 times faster than the CPU + GPU. When the number of users is one million and the average number of transactions is 100, the proposed method is 10.7 times faster than CPU only and 5.2 times faster than CPU + GPU. Compared to the case without conditions, the speed-up rate is significantly improved, especially when compared with CPU + GPU.

In this evaluation, we assumed one-to-one transactions between one sender and one receiver, so the number of transactions is equal to the degree of the user graph. In the case of one-to-many or many-to-many transactions, edges are created for all the combinations of sender and receiver pairs. In this case, the computation cost of the feature quantity extraction is proportional to the number of edges of the user graph, not the number of transactions. The average number of transactions in an actual Bitcoin network is about 16 [7], which is smaller than this evaluation. However, the average transaction size is about 500 bytes. In this size, the minimum number of participants (senders and receivers) in the average-sized transaction is five (i.e., four senders and one receiver). Therefore, the average degree of the user graph is estimated as at least 64 (= 16 × 4).

4.4 Relationship between Data Size and Size of GPU Device Memory

Figure 14 shows the memory usage of the GPU device memory in the proposed method in the evaluation. As shown in the figure, the memory usage is proportional to the number of transactions. When the number of transactions per user is 100 and the number of users is 1 million, the total number of transaction is 100 million and the memory usage is 3.36GB. In this case, whole user graph can be stored in the GPU device memory.

4.5 Execution Time When Entire User Graph cannot Fit on GPU Device Memory

If the entire user graph cannot fit on the GPU device memory, the proposed method can be used by storing a part of...

†Information size of a sender is larger than that of a receiver, since a signature is included in the sender information. If we assume more receivers, the number of participants will increase.
the graph. In this case, the feature extraction of the partial graph is performed by GPU and the extraction of the rest of the graph is performed by CPU in parallel. Even in the case, the result of processing is not changed because the whole graph data is stored and processed in CPU and GPU by storing the partial graph in GPU and the rest of graph in CPU. In the proposed data structure, the user graph can be divided by the number of user because the graph stores the transactions collectively for each user.

Figure 15 shows the execution time when the entire user graph cannot fit on the GPU device memory. In the figure, the number of users is 1 million and all cases are unconditional. In the proposed method, the partial graph of which the total number of transactions is about 100 million (precisely, the size is decided by the number of users so that the total number of transactions do not exceed 100 million). The result shows that the acceleration rate is decreased proportion to the number of transactions. However, when the average number of transactions is 500, the proposed method is 18% faster than CPU + GPU method. In the case, the total number of transactions is 500 million which is more than the number of the transactions in whole Bitcoin transaction (as of November 2019). If we need to avoid the decrease of acceleration rate, multiple GPUs should be used, because if the total device memory size of the multiple GPUs exceeds the graph size, whole graph can be processed in the GPUs.

4.6 Effect of Conditional Feature Extraction on Execution Time of Anomaly Detection

Figure 16 shows the execution time of the proposed method when anomaly detection is performed in case of the number of users is decreased by conditional feature extraction. The condition is same as Sect. 4.3 and the transactions is adjusted to reduce the specific number of users matched to the condition. In this figure, execution times of two cases are shown: conditional feature extraction and unconditional feature extraction. In both the methods, the number of users is one million and average number of transactions is 100. In the conditional method, the number of users after feature extraction is changed as shown in x-axis of the figure. In the unconditional method, the number of users is constant.

As shown in Fig. 16, the conditional method is slower than the unconditional method when the number of users is 800 thousands and one million cases, because computation cost for feature extraction increases. If the number of users satisfying the conditions is small, the conditional method is faster than the unconditional method. When the number of users satisfying the condition is 200 thoustands, the conditional method is 1.7 times faster than the unconditional method. This result demonstrates that it is necessary to decide the condition so that the number of users satisfying the condition is small when conditional feature extraction is performed for acceleration.

4.7 Comparison between Proposed Method and KVS-based Method

In the above evaluations, the proposed graph cache based data structure is used for all compared methods (CPU, CPU + GPU, and GPU). Thus, all the evaluations include the proposed data structure. In order to compare the proposed data structure and other data structure, we compare the proposed method and the KVS (Key-Value Store) based method which is used for most of Bitcoin full node imple-
ments. In this evaluation, we implement KVS-based method as a hash table and all processing of the KVS is performed in-memory. Figures 17 and 18 show execution times of these methods for one million users data. Figure 17 shows the unconditional cases and Fig. 18 shows the conditional cases. There is different feature in the KVS based method, that is, the execution time of conditional cases is shorter than the unconditional cases. It is because the bottleneck of feature extraction in KVS based method is the reading of feature data, and the reading is decreased in the conditional cases by not reading data which are not met to the condition. On the other hand, the bottleneck in the proposed method is conditional branch in conditional cases and the execution time is longer than unconditional cases. Even with the differences, as shown in the figures, the proposed method can accelerate the feature extractions in all cases. The proposed method is 13.9 times faster than the KVS-based method in unconditional case and 4.6 times faster than in conditional case when the average number of transactions is 100.

5. Conclusions

Since Blockchain systems cannot modify approved transactions, it is necessary to detect abnormal transactions at high speed and high accuracy in order to suppress the damage of illegal transactions by countermeasures, such as correcting transactions before approval. However, to detect the abnormalities with high accuracy, we need to repeat the detections using various features, so extraction of the feature quantities becomes a bottleneck for high speed abnormality detection. Therefore, in this paper, to accelerate abnormality detection, we proposed to cache transaction information necessary for extracting feature quantities in GPU device memory and performing both the feature extraction and abnormality detection by the GPU. We also proposed a conditional feature extraction to reduce computation cost of abnormality detection. In the experiments, we conducted abnormality detection using K-means clustering with considering a condition of transaction amount. When the number of users is one million and the number of transactions is 100 mil-

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” https://www.bitcoin.com/bitcoin.pdf.
[2] “Counterparty,” https://counterparty.io/.
[3] A. Nordrum, “Wall Street Occupies the Blockchain - Financial Firms Plan to Move Trillions in Assets to Blockchains in 2018,” IEEE Spectr., vol.54, no.10, pp.40–45, Sept. 2017.
[4] “Ethereum Project,” https://www.ethereum.org/.
[5] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making Smart Contracts Smarter,” Proc. ACM SIGSAC Conference on Computer and Communications Security, pp.254–269, Oct. 2016.
[6] T. Pham and S. Lee, “Anomaly Detection in Bitcoin Network Using Unsupervised Learning Methods,” Compution Research Repository, vol.abs/1611.03941, pp.1–5, Nov. 2016.
[7] “Bitcon Rich List,” https://bitinfocharts.com/.
[8] S. Morishima and H. Matsutani, “Acceleration of Anomaly Detection in Blockchain Using In-GPU Cache,” Proc. 16th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA’18), pp.244–251, Dec. 2018.
[9] T.T.A. Dinh, J. Wang, G. Chen, R. Liu, B.C. Ooi, and K.-L. Tan, “BLOCKBENCH: A Framework for Analyzing Private Blockchains,” Proc. International Conference on Management of Data, pp.1085–1100, May 2017.
[10] S. Morishima and H. Matsutani, “Accelerating Blockchain Search of Full Nodes Using GPUs,” Proc. 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP’18), pp.244–248, March 2018.
[11] T.H. Hetherington, T.G. Rogers, L. Hsu, M. O’Connor, and T.M. Aamodt, “Characterizing and Evaluating a Key-value Store Application on Heterogeneous CPU-GPU Systems,” Proc. International Symposium on Performance Analysis of System and Software, pp.88–98, April 2012.
[12] T.H. Hetherington, M. O’Connor, and T.M. Aamodt, “Men-cacheGPU: Scaling-up Scale-out Key-value Stores,” Proc. ACM Symposium on Cloud Computing, pp.43–57, Aug. 2015.

Shin Morishima is an assistant professor in the Faculty of Engineering at Toyama Prefectural University. His research interests include Blockchain and GPU-based systems. He received a Ph.D. in engineering from Keio University in 2018. He is a member of IEICE.
Hiroki Matsutani is an associate professor in the Department of Information and Computer Science at Keio University. His research interests include computer architecture. He received a Ph.D. in engineering from Keio University in 2008. He is a member of IEICE, IPSJ, and IEEE.