Accelerating Blockchain Search of Full Nodes Using GPUs

Shin Morishima
Dept. of ICS, Keio University,
3-14-1 Hiyoshi, Kohoku, Yokohama, Japan
Email: morisima@arc.ics.keio.ac.jp

Hiroki Matsutani
Dept. of ICS, Keio University,
3-14-1 Hiyoshi, Kohoku, Yokohama, Japan
Email: matutani@arc.ics.keio.ac.jp

Abstract

Blockchain is a distributed ledger system based on P2P network and originally used for a crypto currency system. The P2P network of Blockchain is maintained by full nodes which are in charge of verifying all the transactions in the network. However, most Blockchain user nodes do not act as full nodes, because workload of full nodes is quite high for personal mobile devices. Blockchain search queries, such as confirming balance, transaction contents, and transaction histories, from many users go to the full nodes. As a result, search throughput of full nodes would be a new bottleneck of Blockchain system, because the number of full nodes is less than the number of users of Blockchain systems. In this paper, we propose an acceleration method of Blockchain search using GPUs. More specifically, we introduce an array-based Patricia tree structure suitable for GPU processing so that we can make effective use of Blockchain feature that there are no update and delete queries. In the evaluations, the proposed method is compared with an existing GPU-based key-value search and a conventional CPU-based search in terms of the throughput of Blockchain key search. As a result, the throughput of our proposal is 3.4 times higher than that of the existing GPU-based search and 14.1 times higher than that of the CPU search when the number of keys is 80×2^{20} *and the key length is 256-bit in Blockchain search queries.*

1. Introduction

Blockchain is a distributed ledger system built on P2P (Peer-to-Peer) network and was originally proposed in bitcoin [1]. Recently, Blockchain has become popular, because it has been applied to not only crypto currency (e.g., bitcoin) but also the other applications, such as transaction of non-currency asset [2][3] and smart contract [4][5].

In Blockchain, transaction data are propagated to all the nodes that join the P2P network and are verified by these nodes. The nodes are classified into two types: full nodes that verify all the transactions in the network and SPV (Simplified Payment Verification) nodes that verify a part of transactions that involve themselves. Full nodes propagate transaction for the other full nodes and SPV nodes when requested. From a viewpoint of Blockchain users, being a SPV node is much lightweight compared to a full node, because the SPV node can significantly reduce the storage requirement to store transaction data and the number of transactions to be verified. In addition, if a user trusts a specific full node for deposit, the user can use Blockchain without any SPV node operations by deposit of asset to the full node operator. In the case of bitcoin, many users of bitcoin applications (e.g., exchanges, settlements, and wallets) are using bitcoin services without any node operations. For example, the number of Coinbase [6] users that use its wallet service is about 11 millions, while that of full nodes is about 9 thousands as of October 2017. It means that a bitcoin network is accessed by a vast number of users and responded by a relatively small number of full nodes. Therefore, Blockchain search queries, such as confirming balance, transaction contents, and transaction histories, from many users go to full nodes and search throughput by the full nodes would be a new bottleneck of Blockchain system.

In this paper, we propose an acceleration method of Blockchain search using GPUs (Graphics Processing Units), because GPUs have high computational power and can be used for acceleration of various databases. A major difference between Blockchain and conventional databases is that there is no update and delete query for old blocks. We accelerate Blockchain search by taking advantage of this feature. More specifically, we propose to use an array representation of Patricia tree that can make use of both the Blockchain and GPU features.

2. Related Work

2.1. Data Structure of Blockchain

Figure 1 shows an overview of data structure of Blockchain. As shown in the upper half of the figure, Blockchain consists of multiple blocks, each of which has information of block ID, previous block ID, and multiple transaction contents. A block is linked to its previous block via the previous block ID, and this relationship continues to the first block. A block ID is generated as a hashed value of data included in the block, and it affects all the successive blocks. Thus, if a block content is modified, all the successive block IDs should be changed accordingly. This means that tamper resistance of Blockchain is very high, because an attacker needs to update all the successive block IDs even when it tampers with only one transaction in a block. The lower half of the figure shows an overview of data structure of a transaction. Each transaction has transaction ID

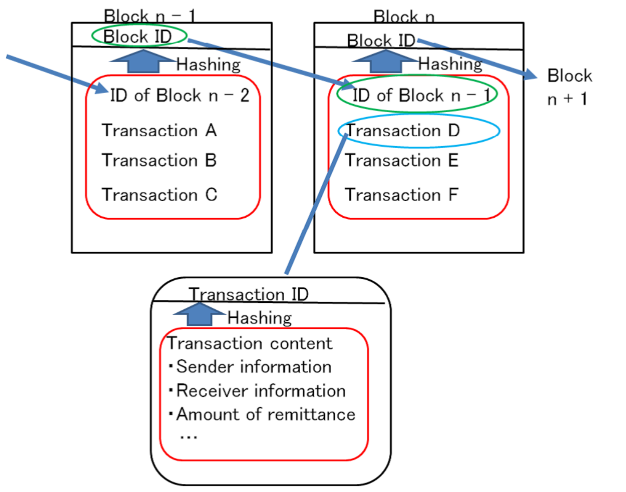


Figure 1. Overview of data structure of Blockchain

and transaction contents. The transaction contents include information of sender, receiver, amount of remittance, and so on. The transaction ID is a hashed value of the transaction contents. A write query in Blockchain appends a new block to the last block in Blockchain. It does not update nor delete any blocks including transaction contents, transaction ID, and block ID

2.2. Acceleration of KVS Using GPUs

KVS (Key-Value Store) is a simple database that stores pairs of a key and a value. KVS typically achieves high performance and high horizontal scalability, thanks to the simplified structure and functions that support simple read and write queries (e.g., GET and SET queries). Full nodes of Blockchain use KVS to store and search transaction data. For example, as a bitcoin system, Bitcoin Core [7], which is one of the most famous full node implementations, uses LevelDB [8], which is a KVS. An acceleration method of KVS using GPUs is proposed in [9]. The proposed acceleration method is enhanced by using NIC (Network Interface Controller) in [10]. In these methods, KVS read queries are accelerated by offloading key hashing and hash table in GPUs. To handle collisions of multiple hashed values, a chain structure is often employed. As a typical software-based implementation of the chain, dynamic allocation of memory and pointer of each key are used. However, such hash table implementation is not very suitable for GPUs, because dynamic memory allocation would degrade the performance [11] and the chain creates a non-deterministic number of elements in each hashed value. To address this issue, a fixed-size set-associative hash table is used in [10] so that GPUs can use statically-allocated array and process a deterministic number of elements in each thread. As a result, up to 13 million requests per second throughput is achieved using the above-mentioned method and direct memory access from NIC to GPUs by GPUDirect [12]. In addition to KVS, document-oriented data stores are accelerated by using GPUs in [13].

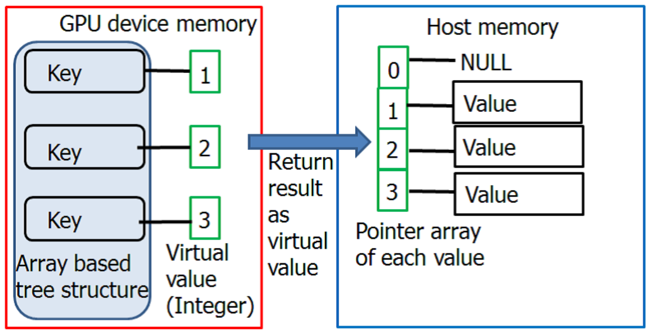


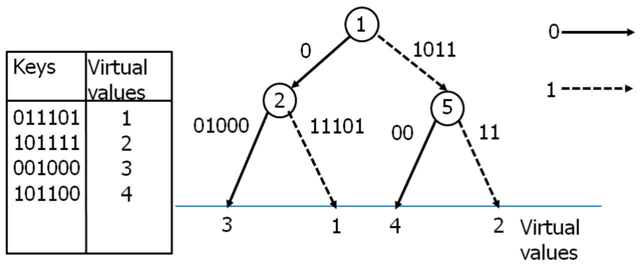
Figure 2. Overview of proposed Blockchain search method using GPUs

3. Blockchain Search Using GPUs

3.1. Overview of Blockchain Search System

Blockchain search queries typically request transaction contents, transaction histories, and balance. KVS can be used for such queries if transaction ID or user address is used as a key and corresponding data is used as a value. Thus, KVS is suited for the Blockchain search queries. However, unlike most applications that use KVS, Blockchain data are not updated nor deleted, as mentioned in Section 2. Using this feature, Blockchain search can be implemented efficiently than regular KVS systems. Figure 2 illustrates the proposed Blockchain search method using GPUs. The key-value pairs are divided into keys and values. The keys are stored in GPU device memory and the values are stored in CPU host memory. An index number is assigned to each value, and it is stored in the GPU device memory as a “virtual value” together with the corresponding key. A host CPU maintains a pointer array that stores host CPU memory addresses where the values are stored. The pointer array is indexed by the virtual values stored in the GPU device memory. That is, a memory address of a value is stored in an element of the pointer array indexed by the corresponding virtual value. The virtual value of the first key-value pair is zero and it is incremented by one whenever a new key-value pair is added. This is because Blockchain is append-only and there are no update and delete queries in Blockchain. Please note that virtual value 0 is reserved as NULL which means that there is no matched key in the search query.

In GPU, a search query searches a given key from all the keys stored in the GPU device memory. If the given key is found in the GPU device memory, it returns the corresponding virtual value; otherwise it returns NULL. Then, in CPU, the target value can be retrieved from a host CPU memory based on the pointer array and the virtual value returned by the GPU search. In this method, CPU-GPU transfer is very small. That is, only a single integer value (i.e., virtual value) is transferred from GPU to CPU as a query result. To accelerate the key search in GPU, keys are organized as an array representation of a tree structure suitable for the GPU processing. Detail will be discussed in the next subsection.



Keys	Virtual values
011101	1
101111	2
001000	3
101100	4

Figure 3. Example of Patricia tree representing four keys

3.2. Data Structure for GPU Key Search

As mentioned above, Blockchain search uses a transaction ID or a user address as a search key. In fact, a transaction ID is generated as a hashed value of the transaction contents and a user address is generated as a hashed value of user's public key. In bitcoin system, length of a transaction ID is 256-bit and that of address is 160-bit (272-bit if header and checksum are included). An existing GPU-based acceleration of KVS uses a fixed-size set-associative hash table [9] as mentioned in Section 2. However, there are two problems in this approach. First, a cache miss occurs when the number of keys that have the same hashed value is more than the number of ways of the set associative. Second, the number of key comparisons between a given key and a cached key increases as the number of ways increases. Multiple comparisons per a query increase the GPU computation cost. Our proposed data structure can address these problems by exploiting the Blockchain feature.

In this paper, we propose to use Patricia tree where each bit in a key represents a node (i.e., radix is two). Figure 3 shows an example of Patricia tree that represents four keys. A tree structure in the right side of the figure is constructed from four keys shown in the left side of the figure. Bit patterns of the four keys are mapped to this tree from their left-most bit to the right-most bit. Starting from the left-most digit, when two or more keys have different bit values (0 or 1) in the selected digit, a vertex diverges. A number in each vertex indicates the digit where a divergence occurs. In this example, the root vertex corresponds to the first digit (denoted as 1). The first digit of the 1st and 3rd keys is 0, while that of the 2nd and 4th keys is 1. Thus, a divergence occurs at the root vertex. Because the first four bits of the 2nd and 4th keys are the same (1011), the root vertex diverges with two edges: edge 0 and edge 1011. The left child node corresponds to the second digit (denoted as 2) and diverges to the 3rd and 1st keys. The right child node corresponds to the fifth digit (denoted as 5) and diverges to the 4th and 2nd keys. Overall, the number of vertexes is $n-1$ when the number of keys is n, because the vertexes are introduced at different bits and all keys are unique. When a new key is added to the tree, the following two steps are performed. First, the tree is traversed from the root based on the bit pattern of a new key and the first unmatched bit is detected. Second, a new vertex is introduced at the first unmatched bit and the new key is assigned to this vertex.

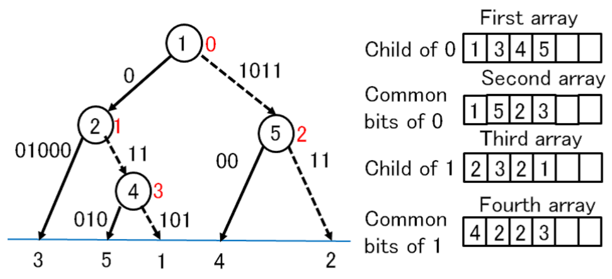


Figure 4. Example of array structure representing the tree structure

Algorithm 1 Proposed key search on GPU

```
1: $ptr = 0$ // Vertex currently visited (initial value is 0)
2: $searchptr = 0$ // Digit of key currently examined
3: $P_0[], P_1[]$ // The first and third arrays
4: $C_0[], C_1[]$ // The second and fourth arrays
5: $length$ // Key's bit length
6: $key[]$ // Bit pattern of the key (key[0] is the first bit)
7: while $searchptr < length$ do
8:     $p = searchptr$
9:     $searchptr = searchptr + C_{key[p]}[ptr]$
10:    $ptr = P_{key[p]}[ptr]$
11: end while
12: return $ptr$ // Virtual value of the key
```

In tree structures, an edge between two vertexes are typically implemented as a pointer. As a result, many pointers are used in a tree-based search, but such implementation is not suited for GPU kernels. In this paper, Patricia tree is represented as an array structure suitable for using GPUs. The array structure is simply built by storing each vertex in order added, because write queries of Blockchain are append-only and do not update nor delete existing keys. More specifically, four arrays are created. The length of each array is the same as the number of vertexes. Each vertex is corresponding to elements at the same position in the four arrays in order added. Figure 4 shows an example of the array structure representing the tree structure. Four arrays in the right side of the figure represent the tree structure shown in the left side of the figure. Each element in the first array indicates the left child vertex and that in the second array indicates the digit of the left child vertex (i.e., a number in the circle). In the same way, each element in the third array indicates the right child vertex and that in the fourth array indicates the digit of the right child vertex. Please note that corresponding virtual values are stored in the tree as leaf vertexes. For example, vertex 1's left child vertex is virtual value 3 as shown in the second element in the first array. Vertex 3's left and right child vertexes are virtual values 5 and 1 as shown in the fourth elements in the first and third arrays. Thus, the virtual value of a given key can be efficiently searched by using GPUs.

3.3. Key Search Using GPUs

In this paper, we use CUDA [14] as a developmet platform of NVIDIA GPUs. A CUDA thread is created for each search query. Many threads can be executed in parallel to process many queries. Algorithm 1 shows the search process in each thread. ptr in the 1st line is a currently visited vertex and $searchptr$ in the 2nd line is a currently examined digit in bit pattern of the key. Four arrays in the 3rd and 4th lines are the proposed array representation of Patricia tree. $length$ in the 5th line is bit length of the key. For example, it is 256 when a transaction ID of bitcoin is used as a key. The condition in the 7th line checks if the currently examined digit is less than bit length of the key. If it is true, it means that the search is not completed; otherwise the search is terminated. In the 8th, 9th, and 10th lines, a digit in the key is searched. $key[p]$ is a value (0 or 1) of the currently examined digit in the key (e.g., $key[0]$ is the first bit). The currently visited vertex and the currently examined digit in the key are updated in the 9th and 10th lines, respectively. These update steps execute only addition and substitution without any conditional branches; thus these update steps can be effectively parallelized in GPUs. As each leaf vertex represents a corresponding virtual value, in the 12th line, ptr to be returned is the search result.

As a search result, the corresponding virtual value of a given key is obtained if the key is included in the tree. Otherwise, the search result is not matched to the given key. For this reason, a verification step that compares a given key and the key searched is required. This verification step returns 0 (i.e., NULL) as the virtual value when the comparison is not matched. It can be implemented as a simple bit string comparison in GPUs. In this case, bit patterns of all the keys must be stored in the GPU device memory for the comparisons. On the other hand, if this verification step is executed in the host CPU, we do not have to store bit patterns of all the keys in the GPU device memory, and thus we can save the GPU device memory usage.

4. Performance Evaluations

In this section, we compare the following four methods in terms of performance.

- GPU: The proposed method using GPU only. Verification of result is done by GPU.
- GPU+CPU: The proposed method using both GPU and CPU. Search is done by GPU, while verification is done by CPU.
- HASH: An existing GPU-based approach that uses fixed-size set associative hash table [10]. The number of ways is 16.
- CPU: A software approach that uses hash table with chaining for conflict resolution.

The above hash table based implementations (i.e., HASH and CPU) use a hash function. In the evaluations, a part of bits of keys are used as hashed values, because the original keys (e.g., transaction ID and user address) are already hashed values in Blockchain system. All the methods are executed at the same machine. The processor is Intel Xeon E5-2637v3 running at 3.5GHz and memory capacity is 256GB. A single NVIDIA GeForce GTX980Ti GPU is used for the GPU processing. The

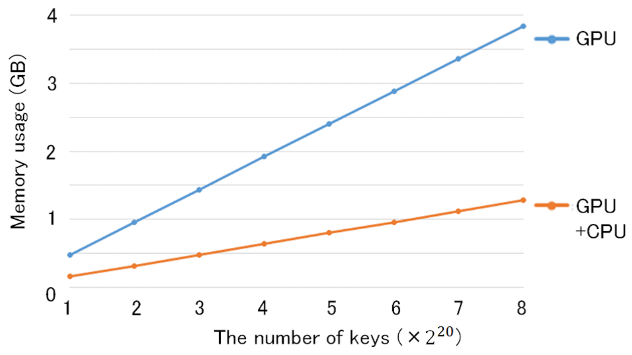


Figure 5. Relationship between the number of keys and device memory usage

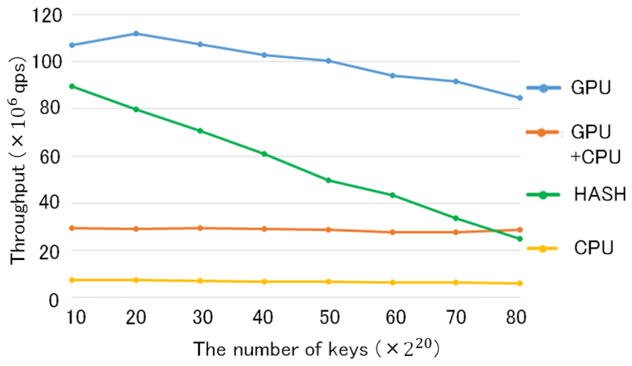


Figure 6. Relationship between throughput and the number of keys ($\leq 80 \times 2^{20}$)

number of cores is 2,816, the core clock is 1,038MHz and the memory capacity is 6GB of the GPU.

4.1. GPU Device Memory Usage

As mentioned in Section 3.3, the memory usage of GPU processing in our proposal depends on by which the verification is done (i.e., GPU or GPU+CPU). Figure 5 shows the memory usage of the tree structure when the key length is 256-bit and the number of keys is varied from 10×2^{20} to 80×2^{20}. The memory usage increases in proportion to the number of keys in both the methods. GPU+CPU can save the memory usage and the usage is one third of the GPU-only when the number of keys is 80×2^{20}. In the bitcoin system, the number of all the transactions until now is about 260 millions. They cannot be stored in a single GeForce GTX980Ti when the GPU-based key verification is used. In this case, GPU+CPU method should be used.

4.2. Relationship between Throughput and Number of Keys

Figure 6 shows the relationship between throughput and the number of keys when key length is 256-bit and

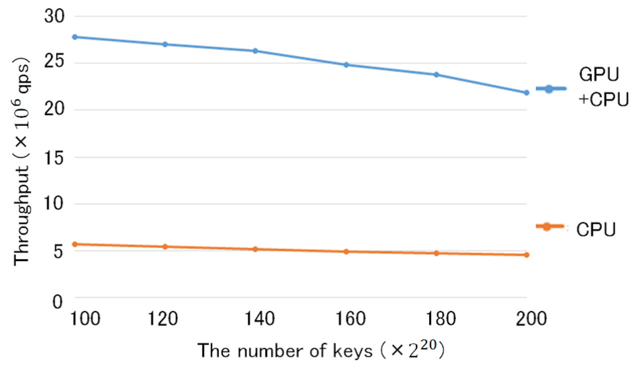


Figure 7. Relationship between throughput and the number of keys ($\geq 100 \times 2^{20}$)

the number of keys is varied from 10 millions to 80 millions. In the tree-based key search, the number of vertexes is proportional to the number of keys. It is a binary search, so the order of the computation cost of the search is $O(logn)$ when the number of keys is n. In Figure 6, the throughput is degraded as the number of keys increases in GPU method. However, the throughput is almost constant in all the cases in GPU+CPU method; this is because the verification time of GPU+CPU method is constant regardless of the number of keys and it is significant in the whole process. In the hash-based key search, the order of the computation cost of hash search is $O(1)$. However, in Figure 6, the throughput is degraded as the number of keys increases in HASH method. This is due to collisions of hashed values. In the fixed-size set associative hash table, as the number of keys increases, due to the collisions, the number of comparisons for verifying a key increases up to 16. On the other hand, the throughput in CPU method is almost constant, because of a low probability of collisions because the hash table can utilize abundant host CPU memory. In summary, when the number of keys is 80×2^{20}, the throughput of GPU method is 3.4 times higher than that of HASH method. It is 14.1 times higher than that of CPU method.

Due to the GPU device memory limitation, when a huge number of keys are searched, GPU+CPU method is a reasonable option for the GPU processing. To discuss such a situation, we evaluate the cases when the number of keys is more than 100×2^{20}. Figure 7 shows the results when the number of keys is more than 100×2^{20}. Although the throughput of GPU+CPU method is degraded as the number of keys increases, it is still 4.8 times higher than that of CPU method even when the number of keys is 200×2^{20}. This result demonstrates a good scalability of GPU+CPU method that does not store all the keys in the GPU device memory.

5. Summary

In this paper, we proposed an acceleration method of Blockchain search at full nodes using GPUs. We proposed an array-based Patricia tree structure for the Blockchain key search using GPUs. The structure consists of four arrays and optimized for the GPU processing. It can make effective use of Blockchain feature, that is, there are no update and delete queries. Furthermore, the structure can search a given key without storing all the existing keys in the GPU device memory; as a result, a huge number of keys can be managed within a single GPU. In the evaluations, the proposed method is compared with an existing GPU-based key-value search and a conventional CPU-based search in terms of the throughput of Blockchain key search. As a result, the throughput of our proposal is 3.4 times higher than that of the existing GPU-based search and 14.1 times higher than that of the CPU search when the number of keys is 80×2^{20} and the key length is 256-bit in Blockchain search queries. When all the keys cannot be stored in a single GPU, the throughput of our proposal is 4.8 times higher than that of CPU search.

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 Spectrum*, pp. 40–45, Sep. 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," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2016, pp. 254–269.

[6] "Coinbase," https://www.coinbase.com.

[7] "Bitcoin Core," https://bitcoin.org.

[8] "LevelDB," http://leveldb.org/.

[9] T. H. Hetherington, T. G. Rogers, L. Hsu, M. O'Connor, and T. M. Aamodt, "Characterizing and Evaluating a Key-value Store Application on Heterogegenenous CPU-GPU Systems," in *Proceedings of the International Symposium on Performance Analysis of System and Software*, Apr. 2012, pp. 88–98.

[10] T. H. Hetherington, M. O'Connor, and T. M. Aamodt, "MemcachedGPU: Scaling-up Scale-out Key-value Stores," in *Proceedings of the ACM Symposium on Cloud Computing*, Aug. 2015, pp. 43–57.

[11] X. Huang, C. I. Rodrigues, S. Jones, I. Buck, and W. mei Hwu, "XMalloc: A Scalable Lock-free Dynamic Memory Allocator for Many-core Machines," in *Proceedings of the IEEE International Conference on Computer and Information Technology*, Jun. 2010, pp. 1134–1139.

[12] "Developing a Linux Kernel Module using GPUDirect RDMA," http://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[13] S. Morishima and H. Matsutani, "Performance Evaluations of Document-Oriented Databases using GPU and Cache Structure," in *International Symposium on Parallel and Distributed Processing with Applications*, August 2015, pp. 108–115.

[14] "NVIDIA CUDA," https://developer.nvidia.com/cuda-zone.