Accelerating Virus Scanning With GPU

Thipakar Sabapathipillai, Sinthuja Kopalakirushnan, Dhammika Elkaduwe and Roshan Ragel
Department of Computer Engineering
University of Peradeniya
Peradeniya 20400 Sri Lanka

***Abstract*- We implement the scanning through GPU to obtain an accelerated process. We mainly target to use two types of algorithms such as Aho-Corasick and Boyer-Moore algorithms. For the Aho-Corasick algorithm, we consider the polymorphic virus patterns and do scan where it means the approximate string matching. For the Boyer-Moore algorithm, we consider the non-polymorphic virus patterns that are the actual pattern matching. Experiments conducted on an NVIDIA Tesla C2075 GPU that has 448 cores and 21504 threads running off of a 2 Xeon E5-2670 processors host CPU.**

Keywords: Aho-Corasick, Boyer-Moore, GPU, CUDA

1. INTRODUCTION

Normally, virus scanning is done on CPU (Central Processing Unit). In CPU, the number of cores and threads are lesser (for example, our lab server CPU has 2×8 cores and 2×16 threads). Therefore, it may take hours or days to scan a huge set of files. Since there is a large amount of data, we are expecting a very fast virus scanning. Due to the time-consuming of virus scanning process, we usually skip the scanning process and keep on working. It is not safe to be like that. Viruses could attack our important data. Therefore, the data should be protected from viruses in an efficient way. If the time taken for virus scanning is reduced, then it could be a better solution for this problem. The GPU (Graphics Processing Unit) is a special hardware that is used in this case. It has a large number of cores and threads when compared to CPU (for example; Our lab server GPU has 448 cores and 21504 threads). Therefore, the data processing could be handled through the GPU in parallel, and an accelerated scanning process can be obtained.

For the scanning process, we use an antivirus software. It is computer software that is used to prevent, detect and remove malicious software from the computer. The malicious software is computer software that damages the computer program. They enter into the computer without the permission of the users and run against the wishes of him/her. So, it is necessary to scan the viruses and save the data from virus attack. The antivirus software is used as a virus guard in the computer. It detects the viruses through virus scanning. The virus scanning process is based on pattern matching. Each virus has a pattern. In the process of scanning, the pattern will be matched with the files that are given. A large set of virus database is needed to match with the files. The anti-virus software uses two different techniques to do virus scanning. The first one is the virus dictionary approach that scans the files to search for known viruses from the virus database. The second one is the suspicious behavior approach that is to identify the suspicious behavior of any computer program that is infected.

From the two techniques mentioned above, we focus on the virus dictionary approach. In case of virus dictionary approach, the antivirus software scans the file, and it matches the virus patterns according to the known viruses that are stored in the database. If the virus is found, the antivirus software can delete the file or quarantine it. Then the virus can be removed or inaccessible, and stop spreading. When the computer's operating system creates, opens, closes or downloading files, first it let antivirus software to scan the file. This happens when the user wants to scan the files manually. The virus dictionary approach handles two categories of a pattern set. Those are polymorphic virus patterns and non-polymorphic virus patterns. The polymorphic virus could change its size and shape. However, the non-polymorphic virus has a fixed set of patterns. The polymorphic virus pattern set is handled by the Aho-Corasick algorithm that is a string matching algorithm. The non-polymorphic virus pattern set is handled by Boyer-Moore algorithm that is also a string matching algorithm. However, both algorithms have different functionalities in their implementation. These are the two main set of algorithms that we are going to use in our project.

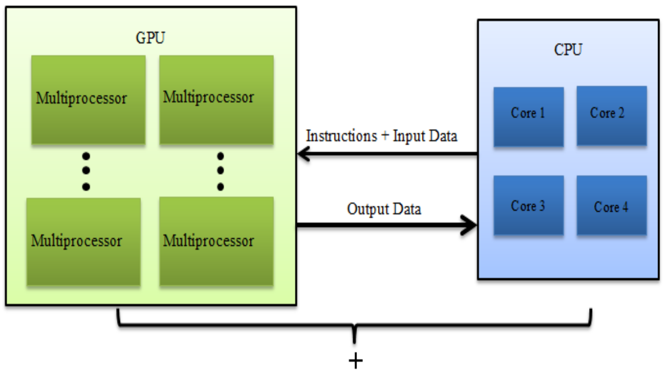

Fig. 1. Methodology of the process

Figure 1 shows the layout of the method of the project work. According to the applied code, the input required and the data instruction will be provided to GPU by CPU. By using the Parallel algorithm in GPU, pattern matching will be done in

parallel and speed up the scanning process. Finally, the CPU gets the results from GPU memory.
From the referred documents, the ClamAV antivirus is said to be open source software. So, we considered the ClamAV antivirus software [3] as sample software to analyze about the virus scanning process. The ClamAV documentation and related references were referred. From the references, the main algorithms, the virus pattern set and the two modules that they used were found out. The Aho-Corasick algorithm and the Boyer-Moore algorithm are the two pattern matching algorithms that they used, and the modules are filtering module and verification module.

ClamAV uses the Aho-Corasick algorithm to detect the polymorphic viruses. The polymorphic patterns (Multi-part patterns) are in regular expression format. It consists of fixed characters and wild characters of detecting viruses. When comparing to non-polymorphic viruses, it is very difficult to detect polymorphic viruses. Because of wildcard characters of polymorphic viruses, it could be able to change the shape each and every time. It will have a different instance for each and every time. The algorithm can handle a large number of patterns and do matching. The algorithm uses trie data structure in its implementation to store the automation generated from polymorphic signatures. Moreover, then the Boyer-Moore algorithm detects the non-polymorphic viruses that are fixed string patterns. There will be no wild characters here.

Wild charge	Meaning	Example
??	Arbitrary character	
*	"??"-it repeats more than zero	XX*YY={XXYY,XX??YY,XX????YY,..}
(XX/YY)	"XX" and "YY" alternation	(XX|YY) = {XX, YY}
{a-b}	"??"-it repeats a or more than a and b or less than b	XX{1-2}YY = {XX??YY, XX????BB}

Table. 1. Wildcard character

Table 1 consists of the wildcard characters that are in between polymorphic virus patterns. Each character has a specific meaning as mentioned in the table above.

The ClamAV uses filtering module to check all the input data files with virus signature and to separate the infected files. After that, in verification module the type of the virus will be identified.

The ClamAV handles different types of a pattern set. So, the patterns and its format were analyzed and found out what pattern set for which purpose. For example, let's consider the three pattern set from ClamAV virus pattern database where the main.db and main.ndb have the polymorphic virus patterns. Moreover, then the main.hdb has the non-polymorphic virus patterns.

In our implementation, the Aho-Corasick and the Boyer-Moore are the two main algorithms. Moreover, from the two modules that analyzed in ClamAV, they did not consider as separately but both modules are combined and used for the scanning process. To implement the GPU functionalities from CPU functionalities, some more information was needed. So, the filtering module, verification module, and the algorithm parts were tried to get from the ClamAV source code. Moreover, also the algorithm implementation, pattern handling, input file handling, and the functionalities of it within the source code were tried to find. However, that information is not available. Some parts of it are in binary files that are not as source code. So we changed the idea to implement the algorithms ourselves.

2. RELATED WORK

In [1], the authors have concentrated on the virus signature detection over real networks. The contribution of their project is scanning only in the downloading process. If there are more signatures of virus available that takes much time to do the scanning in CPU. The CPU cannot stand with the speed of the network. Even though, there are large bandwidth present; it cannot be used completely. When virus scanning is conducted in CPU at the downloading time, the throughput will reduce. Therefore, in their project, they have used GPU for virus pattern matching; speed up the scanning process in internet downloading. In their project, the speed of the scanning process in GPU cannot be increased beyond the bandwidth and the speed of the network. The scanning speed bottlenecks with bandwidth. When comparing to their work, the input file to GPU can be provided according to the speed of CPU in our project. We are trying to increase the speed of the scanning process in GPU as much as possible.

In [2], the authors have focused on pattern matching of Biological sequence data. Their work involves in calculating the probability of Viterbi of a hidden Markov model (HMM). The implementation of Viterbi algorithm on GPUs and the streaming version of the algorithm have been done. HMMer is the open source implementation in their algorithm implementation. The cluster version of HMM-search with linear scalability has been demonstrated. In their project, they do the pattern matching on GPU to do the HMMer-search implementation, but their algorithm is different from our project. They are using MARKOV algorithm, and we are using Aho-Corasick and Boyer-Moore Algorithms. These algorithms can be varied according to the number of calculations and data handling technique.

3. PFAC LIBRARY FOR AHO-CORASICK ALGORITHM

We use PFAC library [4] to implement the Aho-Corasick algorithm. PFAC is an open library for string matching on GPUs. PFAC means; it is a library for the Aho-Corasick algorithm that is implemented for GPU. We obtained a PFAC CUDA toolkit version 2.1 from the internet and tried to make it support for our lab server. However, our lab server has the CUDA toolkit version 6.5. So we try to make it support to that. We did the implementation by updating CUDA functions like cudaBindTexture to cudaBindTexture2D.

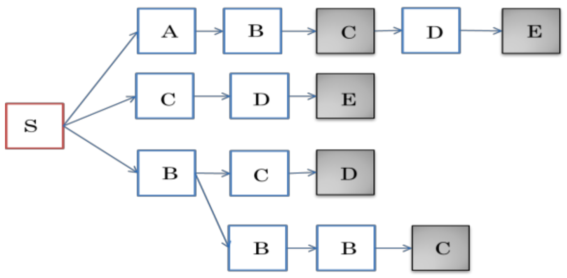

Fig. 2. State diagram for the pattern in PFAC

In Figure 2, it has been shown that if we give the patterns like ABCDE, ABC, CDE, BCD, and BBBC, then the PFAC will draw a state diagram like mentioned above. Here the boxes that highlighted are the end states.

Then the input is needed to give to check for the feasibility of matching with the pattern. If the input is given, it will create threads according to the number of letters in it. Then each thread will start from each letter in the input, and it shares the state machine among them to do string matching.

4. BOYER-MOORE ALGORITHM AND MD5 CHECKSUMS

The Boyer-Moore algorithm handles non-polymorphic viruses that are in simple fixed string signatures. The algorithm uses hash functions to detect non-polymorphic viruses that can be as a malware. A Malware is the short form of malicious software. It can be identified as a program that disrupts computer; gathering sensitive information; or gain access to the private computer system. In this case, the hash value for a malware program is found out and then it is matched with the hash values that are already in the virus database.

The hash values are in MD5 checksum format that is expressed as 32 digits hexadecimal number. An MD5 algorithm handles the MD5 checksum. It is very easy to create the signatures for file hash checksums. The static viruses are detected through this. The hash checksums have the file extension as *.hdb.

5. IMPLEMENTATION

a. IMPLEMENTATION FOR AHO-CORASICK

As we said earlier, we are using PFAC library for the Aho-Corasick algorithm. This algorithm is implemented in PFAC library for the actual string matching. In our case, we are going to use the Aho-Corasick algorithm for polymorphic viruses that are to do approximate string matching. So, we changed the state machine according to that. Let's see how it has been done.

With the help of an example, we will see how the state machine is implemented in PFAC library, and how it is handling the string matching. In that, it uses 256 integer element 1D-array to represent a state. To draw the state diagram we give AB and ABC as pattern input.

No. of element (AB)	= 2
No. of element (ABC)	= 3
So, total no. of elements	= 5

So, it will create a 5x256 integer element 1D array.

No. of pattern = 2

So, it will take the initial state as 3. It will map the patterns in the created 1D array like shown below.

-1	….	2	…	4	…	1	…	-1
0		323		833		1090		1279

The end state will be represented by a number that is less than 3. In the 1D-array mentioned above the 1 represents the ending of the first pattern; the two will represent the ending of the second pattern and so on. Let's consider the two inputs AD and AB. For both inputs, we examine the thread starting with the first letter.

Consider the input AD,

Iteration 1:
Array Index = Initial state x 256 + ASCII of first element(AD)
= 3 x 256 + 65
= 833

So, array index is equal to 833. In the 1D-element array, we mention it has a value as four not a -1 in the index of 833. So the first element of AD has been mapped and after that the next state will be 4.

Iteration 2:
Array Index = State x 256 + ASCII of second element (AD)
= 4 x 256 + 68
= 1092

In the array index 1092, it has the value -1. It will not match if there is -1. So the pattern is not matched.

For the input AB, it will follow the same steps likewise for AD. After the second iteration, it will have the array index as 1090. The value in it is 1. So the second character is also matched. The value 1 is less than the initial state, so it is an end state, and the input is matched with the first pattern. The PFAC library works like mentioned above.

In our case, the virus database has only first 128 ASCII value characters. However, the PFAC library has 256 integer element array. So we change the array representation for the state to 128 ASCII characters to the efficient of memory. Because of approximate matching, sometimes the scanning process may need to remain in the same state for more than one iteration. So, it is necessary to store more information in the same node itself. So that, we tried a structure node instead of the 1D-array. In the PFAC library, they used texture memory to map the state machine (1D- Integer array) from CPU to GPU. The texture memory is a global memory, and it is a type of read-only memory. Moreover, the traffic for the memory access is low here. For the time efficiency, we planned to use this memory. But, it is not possible to bind user structures to texture in it. So we changed the plan and created the 1D-array according to the element inside the structure node as shown below. Then we could able to map all the arrays in texture memory.

Index Flag Actual Max Min	…………….

Fig. 3. (a)The Structure of node we planned

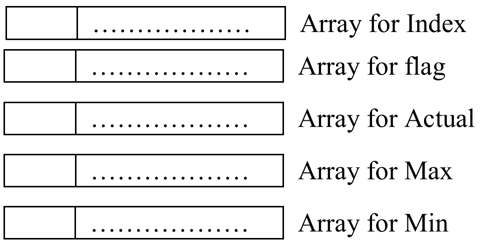

Fig. 3. (b) Array for the elements of structure node

The flag array is used to indicate whether there is a wild character in the node. If there is, then the type of wild character can be identified. According to the details of flag array, the matching process can recognize how much iterations it needs to stay in the same state. According to the flag number it will indicate the state like, if flag 0 – no wild characters after that state, flag 1- fixed number of wild characters after that state, flag 2 – range of wild characters after that state, and flag 3- infinite number of wild characters. Consider the pattern AB{2-4}C. Here it is in state B, it has the flag number 2. The flag 2 indicates that the range of wild characters. So it goes to Max and Min array. According to the data inside those arrays, it will stay in that state.

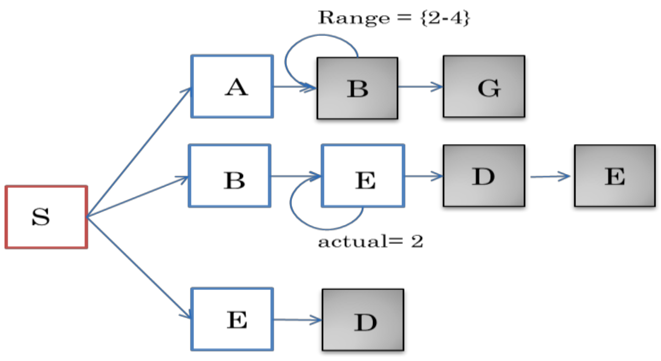

Fig. 4. State diagram for the pattern in PFAC

In Figure 4, it has been shown that if we give the patterns like AB, AB{2-4}G, BE??DE and ED, then the implementation will draw a state diagram like mentioned above. Here the boxes that highlighted are the end states.

Likewise, our implementation for the Aho-Corasick will work, and the implementation is done in CPU as well. **The performance analysis for GPU and CPU implementation is in progress.**

b. IMPLEMENTATION FOR BOYER-MOORE ALGORITHM

Boyer-Moore algorithm is a fast string search algorithm for actual string matching. In our implementation, the algorithm handles exactly fixed strings like checksums. We use the MD5 algorithm to find the checksums. It uses the checksums to find the malware. If there is a huge set of malware, the time was taken to calculate the MD5 checksum in CPU will be high. So, we do that in GPU to reduce the time taken for that. We have implemented the Boyer-Moore algorithm in GPU to do a large set of checksum pattern matching in parallel.

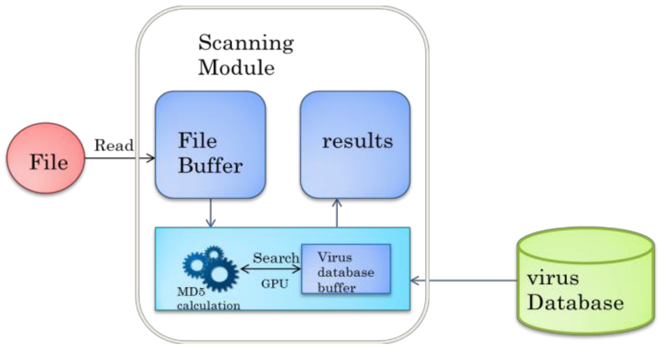

Fig. 5. Implementation of Boyer-Moore algorithm

Our Boyer-Moore implementation has been done as like mentioned in the figure above. **The implementation has been done in CPU, and GPU and the performance analysis for both is in progress.**

6. CONCLUSION

We have implemented Aho-Corasick and Boyer-Moore algorithms in CPU and GPU for the pattern matching of viruses. We use the virus dictionary approach in our implementation. The polymorphic and non-polymorphic virus patterns have been considered. We considered the virus patterns in virus database and matched the patterns with the input files. Our target is accelerating virus scanning with GPU. So, the pattern matching process is done on GPU and, therefore, the time spent for that is reduced. We have completed all the algorithm implementations. Finally, we need to analyze the performance of CPU and GPU. Performance analysis process is in progress.

REFERENCES

[1] Elizabeth Seamans and Thomas Alexander, “Fast Virus Signature Matching on the GPU”, Juniper Network and Polytime, GPU Gems 3, Chapter 35.

[2] Daniel Reiter Horn, Mike Houston and Pat Hanrahan, “ClawHMMER:A Streaming HMMer-Search Implementation”, Stanford University.

[3] Luca Gibelli, Török Edvin, Tomasz Kojm, Alberto Wu, and Nigel Horne, “Documentation”, Available: http://www.clamav.net/doc/install.html

[4] Lung-Sheng Chien, Chen-Hsiung Liu, Cheng-Hung Lin, and Shih-Chieh Chang, “What is PFAC”, Available: http://code.google.com/p/pfac/