Parallelism in Signature Based Virus Scanning with CUDA

Andrej Dimitrioski(✉), Marjan Gusev, and Vladimir Zdraveski

Faculty of Computer Science and Engineering,
"Ss. Cyril and Methodius" University, Skopje, Republic of Macedonia
andrej.dimitrioski@students.finki.ukim.mk,
{marjan.gushev,vladimir.zdraveski}@finki.ukim.mk

Abstract. Information security is playing big role in the computer technologies. Its job is to detect unauthorized violation of the information integrity, secure it and also recover it, if the integrity was violated. One of the things that can alter an information are computer viruses. One of the task of the information security is also to detect these malicious applications and prevent their goal. This can be achieved in various techniques and one of them is signature based virus scanning. This technique uses a virus database (virus signatures) to detect if a file or application is infected with a specific virus. In this paper we are going to see in more details how is this implemented, which algorithm are mostly used and also try to improve its performance by parallelizing it on GPU by using CUDA. We are also going to see how CUDA utilizes large number of threads to solve a specific problem and use it to implement a parallel signature based virus scanner. Later we are going to see the performance benchmarks of the conducted experiments and discuss them and give a final conclusions for the usage of a GPU in signature based virus scanning.

Keywords: Virus · Scanning · CUDA · GPU

1 Introduction

Computer viruses are malicious applications that can harm the computer in various ways and they can be written in different programming languages. Their first appearance starts in 1970's and through the development of computer technologies they were getting more and more advanced and now we know a plethora of different types of computer viruses. In the early 80's computer viruses were primitive, destructive and were mostly distributed through software and transferred from a computer to a computer by using floppy disks. When the computer networking hit mainstream users it opened whole new ways of infecting computers with viruses. In that time there was the first appearance of computer worms which they are able to replicate itself and spread to other computers. This was

V. Poulkov (Ed.): FABULOUS 2019, LNICST 283, pp. 413–422, 2019.
https://doi.org/10.1007/978-3-030-23976-3_36

achieved through emails, hidden in attachments in a form of an application or any type of a media file. Spreading through network usually was aided by flaws in the network stack of the operating systems or flaws in software that relied on the network. This trend continued in the next years till today and it is unimaginable to encounter a virus which does not rely on the computer network. Virus scanning or namely the anti-virus software starts in the late 1980's as measure to deal with viruses. At that time anti-virus software was primitive relaying on a simpler techniques to detect and handle virus infections. These most include techniques like signature based virus scanning and they were quite effective since metamorphic viruses did not exist back then. Also its usage wasn't trivial because it had a complicated interface. As of today that has changed drastically, more detection techniques were developed which was caused from the streamlined improvement of the viruses. These include polymorphic, unusual behavior, heuristic and cloud based detection.

From the start of the development of anti-virus software till today the CPU is the main resource that is used to execute these detection techniques. The CPU mostly relies on sequential execution, though processors of this era are capable to execute several instructions in parallel in the same time due to the larger number of cores and threads. As applications get more and more demanding and techniques get more and more complex this means big performance impact in systems where can it mean more than the actual security. We know that GPUs are capable of utilizing large number of cores and that is proven to have great performance on graphical computing. The possibility to exploit the potential of these devices in more general problems was introduced with the GPGPU [6] or General-purpose computing on graphics processing unit which is extensively used in parallel programming paradigm.

Right now there are several platforms where a end-user can develop GPGPU application. CUDA [7] being the one of the most popular and developed by NVIDIA provides an API to utilize their graphic cards. Having this in consideration we can see that the problem we introduced above may be transformed in a way that we can use the large number of cores in the GPU to scan programs for a malicious code which frees the CPU and we can use it in executing other tasks.

2 Related Work

Virus detection has a wide area of development and research. Since CPU processing still has major role in virus scanning, most of the researches and related work lies on it. In a particular the specific topic about signature based detection looses its popularity because of the lack of a success rate when it comes dealing with polymorphic viruses. Because of that the shift is now towards using heuristic scanning techniques [11], [10] that can deal with self-modifying code. On other hand, related work for a GPU aided virus scanning appeared in late 2009 where NVIDIA posted [3] about Kaspersky Lab using their GPUs and CUDA. This is achieved by uploading the suspected malicious file to Kaspersky Lab data

center where with the usage of NVIDIA CUDA and complex virus detection algorithm it can quickly detect if the file is really malicious and give suggestion to the user what to do next. Kaspersky is claiming that performance increase reached even 360 times over the Intel Core2 Duo processor. NVIDIA released a white-paper [4] on their GPU Gems site showing how pattern matching technique in virus scanning can provide better performance with the use of a GPU. Intel on the other hand announced [5] that their 6th, 7th, 8th and their future CPU generations will offload virus scanning from the CPU to their GPU so it will help the performance and battery life. Although this is not really related to CUDA platform, it shows that work on using the GPU for virus scanning is leaning towards a successful future. Similar work [2] on the paper's area has been done, where by using GPU and NFA [8] improved performance of virus scanning compared to open source anti-virus software ClamAV [9]. Another research on signature based detection has been done where by using highly-efficient memory compression technique and CUDA [12] showed improvement in memory usage and performance in pre-processing and run-time stage. A research [14] on virus detection by using Big Data [13] and Hadoop based comparison and pattern matching was aided by use of a GPU.

3 Architecture and Design

3.1 Specifying the Problem

The problem that we are covering here is a signature based virus scanning. In computer systems files are represented with bytes, that way it is easier for the CPU to work with them. A variable length of bytes may represent instruction, some data or something similar and also they may represent malicious segment of the a file. A old technique that is still being used is a signature based scanning. This technique searches for specific set of bytes. These specific set of bytes (sometimes named Signature) are commonly stored in a file called Virus Definition Database. These databases are updated on regular basis by the anti-virus vendors. So altogether this technique takes the file, and scans if it consists a known signature. In this paper we are going to simplify the problem, and make an assumption that programs that we are going to scan are not going to be polymorphic or in other words, no variations of a signature may end up in a file causing not to be detected by anti-virus.

3.2 Algorithm that We Are Going to Use

Solving the scanning problem on the first hand seems easy and if we consider that we have one pattern (virus definition) to scan. We will certainly have a great scanning performance. The problem gets harder if we have more than one virus definition. A brute force implementation of that will mean that we will need to do as many single searches as the number of virus definitions and that for certain will be slow if we have thousands of registered definitions in the database. We will

have performance improvement if we use some known string searching algorithm as Rabin-Karp, Knuth-Morris-Pratt or Boyer-Moore but still we wont get the results that we want. So we need to find a way to search concurrently for every virus definition in one take. This gets even harder if we have different lengths in virus definitions. That's where Aho-Corasick [1] algorithm comes to play. The idea behind that algorithm is to build a finite state machine represented with a trie structure. This structure contains nodes with links between them which helps in pattern matching by fast transitions. Functions that this algorithm use are: Goto, Failure and Output function. Goto function gives us a next state for the current state and character. Failure function represents when the current character does not have an edge. And output function is used to map every pattern that ends at some state.

3.3 CUDA Programming Model

CUDA utilizes a large number of threads to run on a processor or in this case streaming multiprocessor. It achieves this by running them grouped in a thread blocks, and one or more (highly dependent on the GPU specifications) thread blocks can run simultaneously on streaming multiprocessor. These thread blocks on other hand form a grid of blocks. Having this, CUDA gives opportunity to the programmer to let him define the structure of the grid or thread blocks depending on the problem. Since we can't do computation without data, with CUDA we can also reserve space on the GPU RAM for transferring the required data from the system memory. The memory model in the GPU is divided to 3 kinds of memory: Local - used only by a single thread (also named cache), very limited and the fastest one. Shared - used only by threads defined in a thread block and it is slower than a local memory. Global - used by every thread in the grid and it is the slowest and the largest one.

3.4 Implementation and Restrictions

Aho-Corasick algorithm to work needs first to build the trie as we mentioned before. From the observation this process seems to be highly sequential as we can't simply run a multiple threads for each of the signatures and build the trie. This is because as mentioned earlier it uses all of the patterns to create the structure. But on the other hand after the state machine is built, the algorithm takes the input text (file in our case) processing character by character (byte by byte). This on the CPU will require starting from the start of the file and scanning byte by byte all the way to the end.

In previous subsection we mentioned that GPU can utilize large number of threads. So instead of using the CPU to scan through whole file, we can divide the file in section of a bytes and give each thread to pass a one section. Since we have blocks and each one has same number of threads running on it, we are actually going divide a larger section of the file to each block, and divide that section into smaller sections and give it to each thread in the block. We can see

this illustratively with thread blocks and the threads in it in Fig. 1 and also with Eq. 1 we denote how much bytes each GPU thread will have to scan.

$$bytesPerThread = \frac{totalNumOfBytes}{numOfBlocks * threadsPerBlock} \tag{1}$$

$$bytesPerThread \geq max(L(vd_1), L(vd_2), ..., L(vd_n)) \tag{2}$$

Because we can have a different sizes of a files and also we can have different sizes of blocks and threads, this can vary. But we need to make sure that every thread will scan equal or more bytes than the largest virus definition. We denoted that in Eq. 2

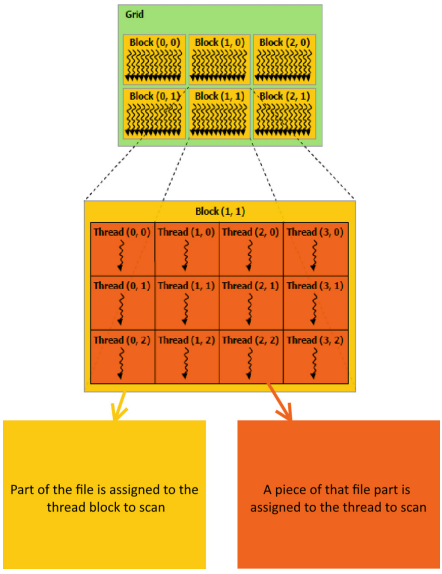

Fig. 1. GPU execution of scanning algorithm.

For the virus scanning we also need the file to reside in the memory of the GPU as well as the data structure (trie) used for scanning the file. We mentioned that the GPU uses global, shared and local memory and they have own memory limits. Keeping this in mind, we need to decide where every piece of the data will reside. Local and shared memory are extremely small so we can store the variables that are used in scanning and also some parts of the trie (Failure and Output function), so every thread will have own variables while scanning and every thread block will have own copy of the Failure and Output function. In global memory we are going to store the Goto function and the file since their size is pretty larger than the rest of the data. Every thread on the GPU will have access to this data. This is where we have limitations of

running the scan more optimal as possible. Access to global memory is slow and even slower when more than one thread wants to access same piece of data. This although is not a problem for accessing the file since every thread has own chunk to process but Goto function since it is used by all threads can lead to high number of memory access conflicts thus decreasing the performance of the scanning. Also since shared memory is small, we cannot always store the Failure and Output functions in it because of their size which depends of the size of the virus definition file, so in some cases if the virus definition is large we have to store them in the global memory which further can decrease the performance. Another restriction we have to mention is the limit of the actual virus definitions which in this implementation is 32 because of the Integer data structure (4 bytes) used for the Output function. CUDA still does not have wide range support for more complex data structures, in our case Bitset data structure which makes it possible to use larger virus definitions.

4 Experiment and Discussion

We conducted a experiment on different files sizes, each with different number of blocks and threads and maximum size of 32 virus definitions. We need to point out that these are random files, and virus definitions were generated with random section of these files. System we used has Intel i5-5200U (up to 2.70 GHz), 8 GB of RAM and NVIDIA GeForce 940M (2 GB of RAM). We benchmarked the time that it takes the CPU and GPU to process the files on their respective algorithms. The goal of this experiment is to show how the GPU and CUDA are capable of executing a signature based scanning on multiple file sizes and comparing them with CPU scan counterpart. We scanned a files with sizes 1670, 1400, 600, 100 and 10 MB. Also we made sure we don't give a thread a number of bytes to scan which is smaller than the largest virus definition we generated.

We ran the scanning on 1670 MB file on the GPU multiple times with different configuration of blocks and threads and also on CPU for reference. The speedup started from 256 blocks and 256 threads per block and kept rising till 1024×1024 which had around 5.7 times speedup over the CPU. With 32×32 and 64×64 configurations we encountered less performance due to a threads having to scan large chunks of the file. That is because performance of a single thread on a GPU is much worse than the performance of a CPU. The results are shown in more detail in Fig. 2 and Table 1 below. The first column of the table represents the CPU result, next ones are GPU results noted by NumberOfBlocks $\times$ NumberOfThreads.

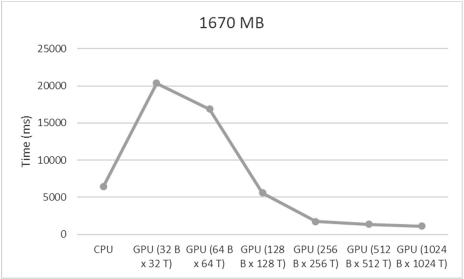

Fig. 2. Running the scan on 1670 MB file.

Table 1. Running the scan on 1670 MB file

	CPU	(32 × 32)	(64 × 64)	(128 × 128)	(256 × 256)	(512 × 512)	(1024 × 1024)
1670 MB	6430 ms	20350 ms	16879 ms	5597 ms	1750 ms	1369 ms	1126 ms

Similar results were encountered on 1400 MB file where we had around 6 times speedup over the CPU. We encountered speedup in the rest of the tests shown below, and they had better results than the CPU on every configuration we tested. The results are shown in more detail in Fig. 3 and Table 2 below.

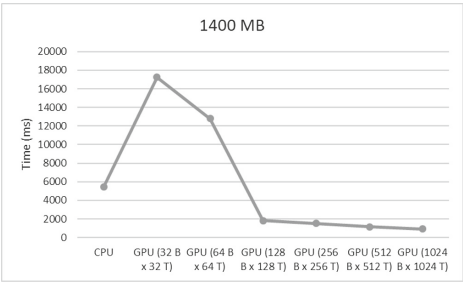

Fig. 3. Running the scan on 1400 MB file.

Table 2. Running the scan on 1400 MB file

	CPU	(32 × 32)	(64 × 64)	(128 × 128)	(256 × 256)	(512 × 512)	(1024 × 1024)
1400 MB	5433 ms	17234 ms	12784 ms	1815 ms	1533 ms	1155 ms	915 ms

In the 600 MB file scan we had steady improvement as we increased the blocks and threads peaking to maximum of 6 times speedup over the CPU. The results are shown in more detail in Fig. 4 and Table 3 below.

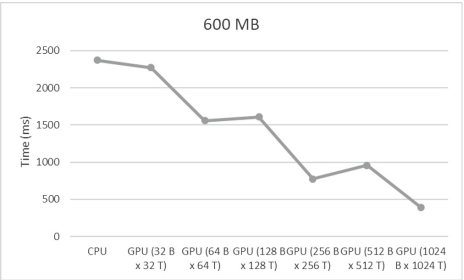

Fig. 4. Running the scan on 600 MB file.

Table 3. Running the scan on 600 MB file

	CPU	(32 × 32)	(64 × 64)	(128 × 128)	(256 × 256)	(512 × 512)	(1024 × 1024)
600 MB	2372 ms	2273 ms	1558 ms	1610 ms	773 ms	957 ms	394 ms

In the 100 MB file scan, the 32 × 32 configuration gave immediate speedup over CPU but as we started to increase them, the performance degraded till 256 × 256 as from that point again to see steady improvement but not much from the 32 × 32 configuration. The results are shown in more detail in Fig. 5 and Table 4 below.

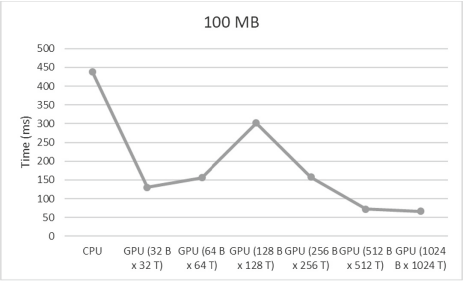

Fig. 5. Running the scan on 100 MB file.

Table 4. Running the scan on 100 MB file

	CPU	(32 × 32)	(64 × 64)	(128 × 128)	(256 × 256)	(512 × 512)	(1024 × 1024)
100 MB	439 ms	130 ms	156 ms	302 ms	158 ms	72 ms	66 ms

10 MB file scan was no different than 100 MB one, same sharing a strange degradation of a performance with 64 × 64 and 128 × 128 configurations. The results are shown in more detail in Fig. 6 and Table 5 below.

Fig. 6. Running the scan on 10 MB file.

Table 5. Running the scan on 10 MB file

	CPU	(32 × 32)	(64 × 64)	(128 × 128)	(256 × 256)	(512 × 512)	(1024 × 1024)
10 MB	56 ms	13 ms	33 ms	32 ms	10 ms	7 ms	5 ms

5 Conclusion

The implementation of the problem even if it was fairly simple and accounting the restriction for the limited usage of the faster shared memory we still managed to get better results than CPU across all of the file sizes we tested. From the experiment we can conclude two types of usages for GPU virus scanning:

(a) Use of large number of threads to scan a very large file.
(b) Use of large number of threads and distribute them across many small files.

This is because our experiment showed a improvement across all file sizes. For the larger ones we saw significant improvement if we used large number of threads. On smaller sizes even the smallest number of threads showed improvement over the CPU. We have to bear in mind that in our test bench we used a graphics card that has small VRAM (video RAM) size and also has limits when it comes storing shared data for every thread block. That gives us greater chance of having a memory access conflict while running a scan and degrading our performance and also preventing us to scan even large files. So in a scenario where it is possible to put the whole scanning data structure in shared memory (better graphics card), we can decrease the memory conflicts and yield even more performance improvements. With this we showed that signature based virus scanning is actually possible to do on GPU as it can be applicable in situations where we want to speedup the scanning of the file or just offload some work of the CPU to the GPU. We also have to mention that this is not the only scanning virus technique out there, there are several others like anomaly detection with machine learning that as well can utilize the power of a GPU. But still as of today signature based scanning is intensively used as first set of checks in the virus scanning procedures or combined with other techniques. Further development and research in this field can enable the opportunity to completely offload the virus scanning on the GPU so we can free up the CPU for the tasks where the performance is most needed.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18**, 333–340 (1975)
2. Vicente Dias, A.N.: Detecting Computer Viruses using GPUs
3. New Virus Scanning Solution Uses NVIDIA CUDA. https://blogs.nvidia.com/blog/2009/12/15/new-virus-scanning-solution-uses-nvidia-cuda/
4. Chapter 35: Fast Virus Signature Matching on the GPU. https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch35.html
5. Intel offloads virus scanning to the GPU for better battery life and performance. https://www.pcworld.com/article/3268985/security/microsoft-intel-virus-scanning-gpu.html
6. GPGPU. https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units
7. NVIDIA Inc.: CUDA. https://developer.nvidia.com/cuda-zone
8. NFA (Nondeterministic finite automata). https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton
9. ClamAV. https://www.clamav.net/about
10. Gao, D., Yin, G., Dong, Y., Kou, L.: A Research on the Heuristic Signature Virus Detection Based on the PE Structure
11. Alberto, C., Gonzlez, N.: Polymorphic Virus Signature Recognition via Hybrid Genetic Algorithm. https://github.com/carlosnasillo/Hybrid-Genetic-Algorithm/blob/master/README.markdown
12. Pungila, C., Negru, V.: A highly-efficient memory-compression approach for GPU-accelerated virus signature matching. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 354–369. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33383-5_22. https://link.springer.com/chapter/10.1007/978-3-642-33383-5_22
13. Big Data. https://en.wikipedia.org/wiki/Big_data
14. Panigrahi, C.R., Tiwari, M., Pati, B., Prasath, R.: Malware detection in big data using fast pattern matching: a hadoop based comparison on GPU. In: Prasath, R., O'Reilly, P., Kathirvalavakumar, T. (eds.) MIKE 2014. LNCS (LNAI), vol. 8891, pp. 407–416. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13817-6_39. https://link.springer.com/chapter/10.1007/978-3-319-13817-6_39