symmetry

MDPI

Article

Endpoint Security in Networks: An OpenMP Approach for Increasing Malware Detection Speed

Igor Forain [1,†], Robson de Oliveira Albuquerque [1,†], Ana Lucila Sandoval Orozco [2,†], Luis Javier García Villalba [2,*,†] and Tai-Hoon Kim [3,†]

1 Decision Technologies Laboratory-LATITUDE, Electrical Engineering Department (ENE), Technology College, University of Brasília (UnB), PO Box 4386 CEP 70910-900, Brasília, Brazil; iforain79@gmail.com (I.F.); robson@redes.unb.br (R.d.O.A.)

2 Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), Universidad Complutense de Madrid (UCM), 28040 Madrid, Spain; asandoval@fdi.ucm.es

3 Department of Convergence Security, Sungshin Women's University, 249-1 Dongseon-Dong 3-ga, Seoul 136-742, Korea; taihoonn@daum.net

* Correspondence: javiergv@fdi.ucm.es; Tel.: +34-91-394-7638

† These authors contributed equally to this work.

Academic Editors: Stefanos Gritzalis and Young-Sik Jeong
Received: 14 May 2017; Accepted: 22 August 2017; Published: 27 August 2017

Abstract: Increasingly sophisticated antivirus (AV) software and the growing amount and complexity of malware demand more processing power from personal computers, specifically from the central processor unit (CPU). This paper conducted performance tests with Clam AntiVirus (ClamAV) and improved its performance through parallel processing on multiple cores using the Open Multi-Processing (OpenMP) library. All the tests used the same dataset constituted of 1.33 GB of data distributed among 2766 files of different sizes. The new parallel version of ClamAV implemented in our work achieved an execution time around 62% lower than the original software version, reaching a speedup of 2.6 times faster. The main contribution of this work is to propose and implement a new version of the ClamAV antivirus using parallel processing with OpenMP, easily portable to a variety of hardware platforms and operating systems.

Keywords: antivirus; ClamAV; OpenMP; parallel processing; security

1. Introduction

Nowadays, malware has caused the unavailability of computer systems because of file corruption and may also lead to a loss of information availability, confidentiality and integrity by introducing errors in the operating systems (OS) or their applications. In addition, some types of malware can take remote control of computers to increase the damage caused by the attacker. It has been seen in the literature that the amount of malware targeting Windows OS (Microsoft Corporation, Redmond, WA, USA) is larger than targeting other OS [1,2].

Malware detection programs act as a function whose domain is the inspected code, and the image is a set of two outputs, i.e., malicious or benign [3]. As detailed in the paper of Idika and Mathur [4], there are two main approaches to malware detection programs: (a) signature detection, and (b) identification of abnormal behavior. The first creates a database of malware signatures from previously identified malicious codes, indicating a possible attack when the application or computer files match the signature. The second approach builds a software behavior model, showing a possible malicious code in programs that does not match the model. When compared, the former has the advantages of accurately identifying known attacks and lower incidence of false positives, but it is not

appropriate to detect new types of attacks. The second can detect new attacks but presents a higher rate of false positives [5].

Antivirus (AV) programs can be used on different computing systems, ranging from personal computers and smartphones to servers. These last, e.g., email and proxy servers, may be capable of detecting the malware before it reaches the target system, blocking the threat. However, a significant amount of data can overload the servers, particularly in a scenario of growing speed of network links and the increasing number of data queues. Furthermore, antiviruses based on a proxy or email servers probably cannot assess files created by the users on their workstation. Server based antiviruses are not supposed to detect malware inserted either via removable media (USB flash drives and external hard drives) or through wireless access not controlled by network administrators before it reaches the target system. Furthermore, there is not a known antivirus that can detect an encrypted virus in files, and this approach has been used lately by attackers [1,2,6].

Increasingly sophisticated antivirus software added to the growing amount and complexity of malware demands more processing power from personal computers, more specifically from the central processor unit (CPU) [7].

The main contribution of this research is focused on the modification of ClamAV engine to a parallel approach to speed up its detection time. With such assumption in mind, this work proposed different techniques of using ClamAV with the Open Multi-Processing (OpenMP) library. Results show a significant reduction in detection time depending on the approach used.

This paper is organized as follows: Section 2 introduces ClamAV and parallel processing with OpenMP. Section 3 discusses some works related to malware detection improvement and algorithm optimization with OpenMP. Section 4 presents bottlenecks of ClamAV and the approaches of this work to parallelize ClamAV with OpenMP. Section 5 analyses the performance results obtained with multi-core CPU parallelization. Section 6 concludes this paper and presents future works.

2. Parallel Processing and Clam AntiVirus

Parallel processing is related to the capability to use multiple processors with the objective of running a program in less time, and there are several approaches to achieve such tasks. For example, the parallel processing software approach handles the execution of a program on parallel processing hardware with the purpose of obtaining scalability and reducing execution time. Thus, some systems are designed to speed up the execution of programs by dividing the program into multiple fragments and processing these fragments simultaneously.

On the other hand, ClamAV is an open source antivirus engine for detecting malware and other malicious threats. It is mainly deployed on mail servers as a server-side email virus scanner.

The following subsections detail some work about parallel processing and ClamAV.

2.1. Parallel Processing

Parallel processing in computer science refers to the execution of different parts of the same program on different processor units simultaneously. Parallel systems can be divided in task and data parallelism. Task parallelism refers to the system capable of executing different tasks simultaneously, while data parallelism indicates the execution of separated instances of the same task over different blocks of data at the same time [8]. In a simplistic view, parallel processing may have three main objectives: execute a work faster, increase the throughput of a program or reduce the latency of bottleneck codes.

Multi-core CPU is a type of hardware classified as shared memory multiprocessor (SMP) because its architecture is based on multiple processor cores sharing the same block of random access memory (RAM). In order to add parallel capacity for systems based on multi-core CPU, there are different platforms and libraries available—for example, Intel (Folsom, CA, USA) Thread Building Blocks (TBB) [9,10], Pthreads [10], Message Passing Interface (MPI) [11], OpenMP [11,12], Open Computing Language (OpenCL) [13] and Open Accelerators (OpenACC) [14]. All of them can be used for

programming multi-core CPUs, but MPI is more appropriate for distributed shared memory (DSM) systems, such as computational clusters [11]. Among TBB, Pthreads and OpenMP, the last has been more used to programming SMP systems because its performance and portability for different CPUs, compilers and OS [10,12,15,16].

OpenCL may be considered as a programming model to explore the parallel capacity of a variety of hardware platforms, like CPU and graphics processing unit (GPU) [13]. Because of its complexity when compared to OpenMP, OpenCL has been more used to GPU programming [15], but it still has been used to the parallel programming of multi-core CPU [15,17]. In addition, there are OpenCL implementations for a range of devices and vendors, including Intel multi-core CPUs, Intel Xeon Phi coprocessor, and Advanced Micro Devices (AMD) (Sunnyvale, CA, USA) multi-core CPUs.

Similar to OpenCL, Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model, but it is implemented only for GPU of NVIDIA (Santa Clara, CA, USA) vendor. Despite its popularity, stable implementations and pioneering, CUDA only targets one type of device (GPU) from one vendor (NVIDIA) [17].

OpenMP is a library containing data types, functions, and directives used for parallel programming in Fortran, C, and C++ language compilers. OpenMP programming model allows the combination of parallel and sequential code in the same application. Although presenting the advantage of a unified view of the system memory space, OpenMP does not have a better performance than MPI when used on heterogeneous architectures with non-uniform memory access (NUMA) [11,12]. Since version 4.0, OpenMP has supported programming for GPU [14].

OpenACC is a directive programming model and platform like OpenMP. It supports a broad range of devices, e.g., Intel multi-core CPUs, Intel Many Integrated Core (MIC) Architecture that is used by Intel Xeon Phi coprocessors and NVIDIA GPUs. OpenACC is a standard with members from the OpenMP community. Despite the growing number of supported devices and the potential to suppress OpenMP shortly, OpenACC still does not offer so many loop parallelization features when compared to OpenMP for programming multi-core CPUs [18].

2.2. The ClamAV Program

ClamAV is an open source antivirus program developed in C language on Unix OS (BSD, Berkeley Software Distribution, Berkeley, CA, USA) [19]. Originally, that AV was designed to inspect attachments in e-mail servers as part of the concept of protecting client users. This approach is still valid as it considers the main use of ClamAV. ClamAV also has versions for Linux (Ubuntu, Debian and others) and Windows OS (versions 10, 8, 7, Microsoft Corporation, Redmond, WA, USA) besides other platforms. Nowadays, there are ports of the original code for Windows OS, named ClamWin. This version is intended for command line use and has no graphical interface [19]. Due to its open-source nature, ClamAV has been used by many types of research to improve the performance of malware detection. The following reviewed work presents some research using ClamAV.

Miretskiy et al. [20] modified the original ClamAV, creating a new version with two new modules: Avfs and Oyster. ClamAV engine is used in the works of Cha et al. [7] and Oberheide et al. [21] to provide antivirus as in-cloud network service. Vasiliadis and Ioannidis [6] modify ClamAV engine to execute string match using GPU.

The paper of Huang and Tsai [22] analyzed the 0.95.2 version performance of ClamAV during the inspection of a 600 MB file. They observed that, despite 80% of the malware signatures being hash codes, ClamAV uses 1% of the entire execution time during the match of this type of signature. Nevertheless, the search for signatures based on strings consumes at least 90% of that AV's processing time.

Huang and Tsai state that ClamAV matches signatures based on strings using two algorithms: Aho–Corasick (AC) and a variation of Boyer–Moore (BM). The first one was presented in the paper of Aho and Corasick [23] and used for fixed size string search. The second is used for regular expressions match [24,25].

Through source code analysis of the 0.98.1 version of ClamAV, this work identified 14 modules inside the project of the engine. The two most important among these 14 modules are clamscan and libclamav. The first one (clamscan) is the entry point of the application, responsible for command line interfaces with the AV user, while the second (libclamav) is the main module that implements all functions responsible for a signature match.

The malware signature database of ClamAV contains two main types of signatures: hash number and strings. The string type is subdivided into two groups: fixed size strings and regular expressions. ClamAV has other types of signatures for spam and malware packer detection, but they are not relevant compared to the hash numbers and strings [22].

3. Related Work

This section reviews previous research on frameworks for multi-core programming and approaches to speed up ClamAV.

3.1. Multi-Core CPU Programming

The work of Shen et al. [15] compares the performance of the two most utilized parallel programming libraries for shared memory devices: OpenMP and OpenCL. They use benchmark applications created by Che et al. [26]: Computational Fluid Dynamics (CFD), K-means and Pathfinder. They tested the processing speed of the same applications when executed in the same hardware but compiled with each one of the two parallel programming libraries. They conclude that the performance of applications compiled with OpenCL can be slightly superior when compared to the same application compiled with OpenMP on hardware with multi-core CPU. Nevertheless, OpenCL requires more programming effort and code optimization than OpenMP [15,17]. Therefore, OpenMP is more appropriate for parallel programming systems on multi-core CPU [17].

Schmidl et al. [27] access the performance of OpenMP programs on the new Intel Xeon Phi coprocessor and the Intel Sandy Bridge (SNB). The first one, offering more than 60 cores, can act as an accelerator like in a GPU model or as a standalone SMP. The work [27] studies the performance of Xeon Phi and SNB using kernels, benchmark codes and four real-world applications, achieving a speedup superior to 100 when compared to one core use on Xeon Phi. It shows that OpenMP programs can be portable for the state-of-the-art CPU with almost no modification when compared to GPU based approaches [27]. Despite the described outstanding performance of Xeon Phi, this processor still is not widely in systems for most users of ClamAV.

The works [28] and [16] evaluate the performance of OpenMP benchmarks and applications on a simultaneous multithreaded processor (SMT) and a single-chip multiprocessor (CMP). They conclude that OpenMP performs better on CMP than SMT because threads do not share core resources. They [16,28] say that the performance tends to stop increasing when the number of threads extrapolates the number of cores. The works [28] and [16] do not evaluate experiments where the number of threads is superior to eight. Otherwise, the work [29] says that oversubscription (two, four and eight tasks per core) can improve overall system performance with OpenMP tunning.

3.2. ClamAV Speedup

Mostly, antivirus applications use a database of virus signatures provided by the manufacturer or community responsible for the program. Using hash calculation, string match algorithms or heuristics, anti-viruses search for malware signatures inside computer files [4]. Some approaches to accelerate antivirus processing with x86 architecture for personal computers (PC), Field Programmable Gate Arrays (FPGAs) and dedicated circuits are described in the works of Erdogan and Cao [30], Ho and Lemieux [31] and Lin et al. [32], respectively.

Lee and Yang [33] proposes a flexible head-body matching (FHBM) algorithm for signature-based Network Intrusion Detection Systems (NIDSs). The work [31] uses two types of signature sets from ClamAV, types 1 and 3, achieving a throughput 55% higher than the original deterministic finite

automaton (DFA) algorithm from ClamAV. Lee and Yang [33] does not cover all signature types of ClamAV and does not comment about false positive occurrence.

Miretskiy et al. [20] propose an on-access AV named Avfs. This AV intercepts the calls to the OS and executes a real-time scan before reading or writing to the file system. The scanner module of Avfs is denominated Oyster and uses ClamAV. Oyster modifies the string search algorithms of ClamAV using bloom filters.

Cha et al. [7] present a distributed malware detector where the main scan engine uses ClamAV. Their work denominates this new AV as SplitScreen, and it is based on a cache-optimized structure called feed-forward Bloom filter (FFBF), an evolution of the work presented in [30]. This last one uses a hash function and Bloom filter arrays to match the signature prefix, accelerating antivirus processing. In the same way, the FFBF described in [7] is responsible for a fast string search using signature prefix match.

Vasiliadis and Ionnidis [6] propose a massively parallel antivirus engine named GrAVity. Their work shows a modification of the ClamAV's string search mechanism using DFA. The DFA represents prefixes of the string signatures, an approach similar to [7]. Vasiliadis and Ionnidis [6] evaluate their proposal on a GPU device. Because of the prefix match, there are possible occurrences of false positives with the method proposed in [7,30] and [6].

The SplitScreen achieves a performance twice faster than ClamAV and requires half the memory space. In the worst case, Refs. [7,30] and Ref. [6] may have a performance similar to the original version of ClamAV because they depend on this AV to avoid false positives. Therefore, approaches to accelerate ClamAV are always welcome.

None of the works discussed in this section explores the use of OpenMP library to improve the performance of a stable version of ClamAV. Some of the works cited in this section used GPUs [6] or other dedicated hardware [31,32] to speed up AV processing. They achieved a better performance than this work at the expense of portability, implementation cost and high incidence of false positives.

4. Proposal of ClamAV Parallel Architecture

Figure 1 is the result of analyzing the source code of the version 0.98.1 of ClamAV and understanding its execution flow.

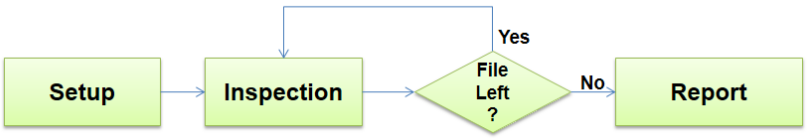

Figure 1. ClamAV's diagram processing.

Figure 1 shows the three major routines of ClamAV: set up, inspection and report. The setup routine is responsible for unpacking the malware signature files and mount the state trees used by the string search algorithms of the AV engine. It takes the same execution time independently of the quantity and size of the inspected files. Nevertheless, it depends on the size of signature database and configuration parameters of the AV. Therefore, the setup routine is not a good candidate for a parallelization attempt.

The execution time of the inspection routine depends on the size and type of the inspected files. This step of ClamAV processing is a loop over the set of files and each iteration search for malware signatures in only one file. The report routine is simple when compared to the first two because it is responsible only for the output of the system results. It includes the indication of the infected files and summarization of the scan results. Because of its low cost of computation when compared to the other two steps of the ClamAV, the report routine is also not a good candidate for parallelization.

During the execution of ClamAV against the executable file vlc-2.0.5-win32.exe, a Nullsoft scriptable install system (NSIS), the AV engine was profiled to collect internal function calls and computational effort. This work calculated the number of calls and the execution time of each function during the profile. Table 1 presents a summary of the results.

Table 1. Execution time of ClamAV string search functions.

Function	Quantity Call	Execution Time	Execution Time (%)
Initialization routine	1	10.43 s	42.83%
cli_bm_scanbuff	2066	4.29 s	17.62%
cli_ac_scanbuff	2149	8.58 s	35.24%
ClamAV	1	24.35 s	100.00%

Table 1 shows that the functions cli_bm_scanbuff and cli_ac_scanbuff are responsible for the most of the computational cost of the ClamAV. These routines are the implementation of the string search algorithms BM and AC, respectively. Table 1 shows a huge number of calls to BM and AC because ClamAV divides the data into blocks of fixed size and both functions search for signatures inside each block at least once.

Designing a Parallel Version of ClamAV with OpenMP

After profiling ClamAV, it is possible to describe the use of the program against **n** distinct files as the sequential execution of the same function scan( ) **n** times. Figure 2 illustrates this behaviour.

Figure 1 shows the same function scan( ) being executed over different data files, characterizing a single program multiple data (SPMD) model [34]. As ClamAV does not use any parallel processing, every call to the function scan( ) of the inspection step is sequentially executed. Figure 2 also shows the existence of a single thread, named master thread, which is responsible for executing the entire program.

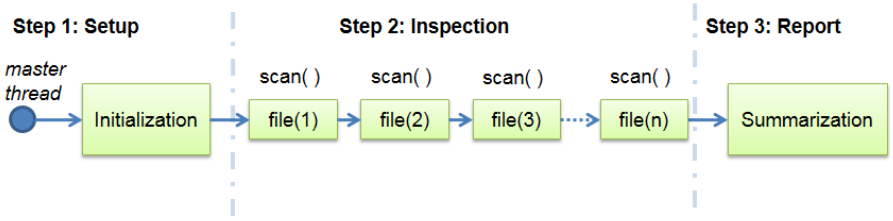

Figure 2. ClamAV's diagram showing the sequential processing.

Even if there is more than one file to be inspected by the AV engine, they are analyzed sequentially by the system. This fact makes the inspection step of the ClamAV a strong candidate for the process of parallelization. Based on such assumptions, this work proposes a parallelization scheme for ClamAV, illustrated by Figures 3 and 4.

Figure 3 shows several threads running in parallel. As described in the previous section, there is no dependence between the calls to the function scan( ). Therefore, ClamAV starts with a single thread named master, which is responsible for running the program step 1. After this step, the fork–join model as defined by the OpenMP library [10] generates new threads.

Figure 4 describes a scenario of unbalanced workload where each new thread and the master thread are responsible for analyzing one or more files during the step 2. After the end of each thread in step 2, the master one can continue and finalize the step 3, summarizing and reporting the malware detection results.

Due to advances in semiconductor manufacturing, nowadays, most of the personal devices have CPUs with two, four or eight cores [35]. Sometimes, the number of scanned files by the AV is greater

than the number of available cores. In this situation, the system can launch more threads than the number of available cores, causing oversubscription of threads [29].

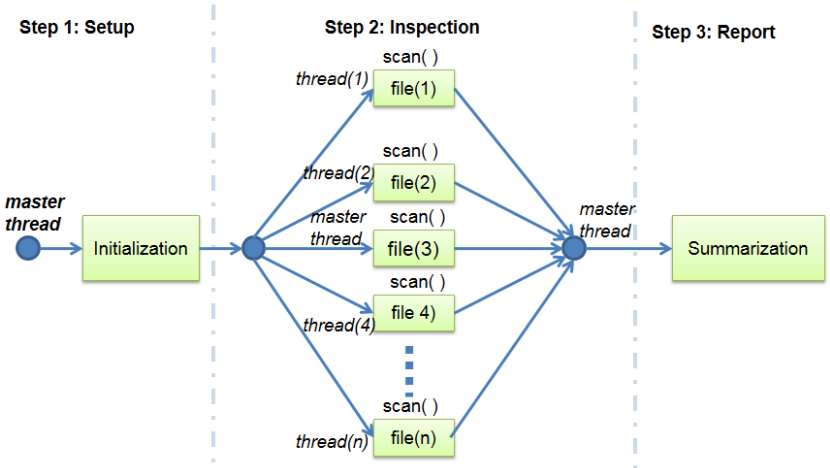

Figure 3. ClamAV's parallel diagram processing (single file per thread).

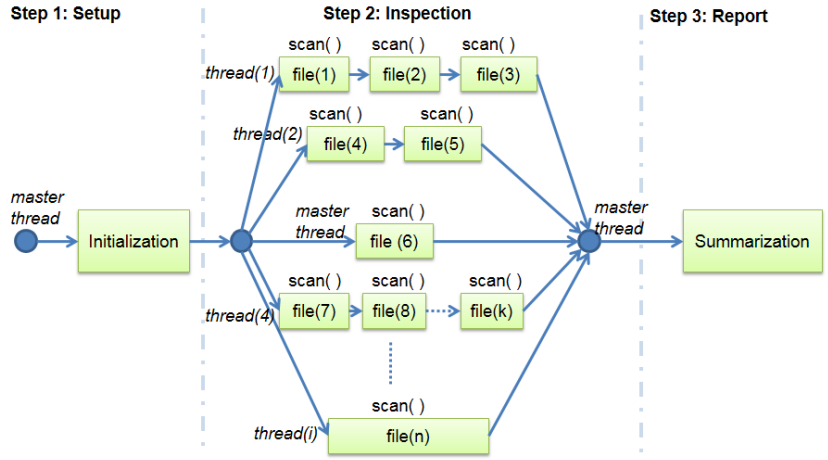

Figure 4. ClamAV's parallel diagram processing (multiple file per thread).

This work uses four approaches to parallelize the inspection loop using OpenMP library: static, round-robin, dynamic and guided. Each one corresponds to the scheduling types defined in the OpenMP library [16].

An OpenMP directive defines the maximum number of threads that can run in parallel in the static approach. The number of target files will be equally divided between the threads. The system maps those groups of files to the threads in a round-robin fashion. The number o parallel threads depend on the number of available CPU cores.

An OpenMP directive defines the maximum number of parallel threads and the number of target files per block in the second approach. The system also maps each block of files to the threads in a round-robin scheduling. The number of parallel threads is also limited by the number of available CPU cores during the execution time.

The difference between the round-robin and the dynamic approaches is the mapping process. The third approach dynamically maps the blocks of files to the running threads to optimize the CPU usage. This run-time scheduling process causes an overhead time when compared to the static and round-robin approaches.

The guided approach looks like dynamic one. The difference between the two approaches is that the guided one decreases the number of files per block exponentially at run-time.

5. Performance Results

This work executes five experiments to validate the proposal. The first uses the version 0.98.1 of the ClamAV against a dataset containing hundreds of files, some of them malware infected. The first experiment creates a baseline for the parallel version of ClamAV with OpenMP. The second, third, fourth and fifth experiments evaluate the four parallelization approaches proposed in Section 4: static, round-robin, dynamic and guided. This work uses the same computing environments and dataset in the five tests. The next three subsections describe in more detail the experimental process and analyze the results.

5.1. System Platform and Experimental Process

For the evaluation of the proposed parallel version of ClamAV, this work uses two PCs. The first one with an Intel Core I5-3210m CPU with two hyper-threading (HT) processor cores, 2.5 GHz clock speed and 4 GB of DDR3 RAM. The second one with an Intel Core2 Duo T6660 CPU with two processor cores, 2.2 GHz clock speed and 4GB of DDR3 RAM. Both computers use Microsoft Windows version 2007 for the x84 64 bit architecture. All programs were implemented in C using OpenMP library 2.0 and compiled using Microsoft Visual Studio 2010. Figure 5 presents a global view of the experimental process.

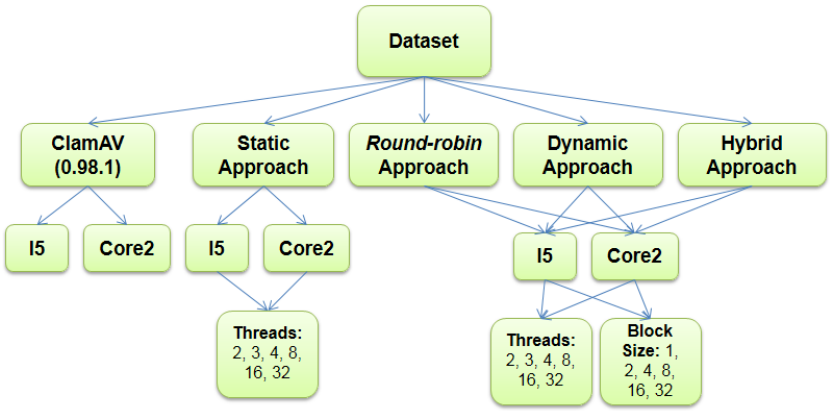

Figure 5. Test diagram.

This work uses a dataset of 2766 different files with sizes ranging from 1 KB to 30 MB. The total size of the dataset was 1.33 GB, and there were 12 files infected with malwares. This small number of infected files represents the reality because the majority of the computer files are not malicious [6].

This work uses the execution time in seconds and speedup of step 2 (Figures 2–4) as a performance metric. The true- and false-positive rates are also calculated to show any reduction in malware detection rate introduced by the parallel approaches. As a way to decrease the random variation, the execution time is measured as an average of 10 runs of each test scenario presented in the Figure 5.

Figure 5 shows the parameters used to test the four approaches of parallelization. The number of threads is a parameter assuming the following values: 2, 3, 4, 8, 16, and 32. The values below or equal to four test scenarios where the number of threads equals the number of CPU cores. The values greater than 4 test an oversubscription scenario. The round-robin, dynamic and guided parallelization approaches also use the block size as a parameter. The combination of these two parameters defines a grid of 36 scenarios that are fully covered by the test.

5.2. Through-Time Analysis

This subsection analyses the performance results in the time domain. It analyses each one of the four parallelization approaches of ClamAV separately and compares the performance of the four approaches in the two computational environments. Note that the *x*-axis is logarithmic (log base 2) for better data visualization.

5.2.1. Static Approach

Figure 6 shows the execution time of the inspection step as the number of threads increases in the PC with I5 and Core2 CPUs during the static approach described previously.

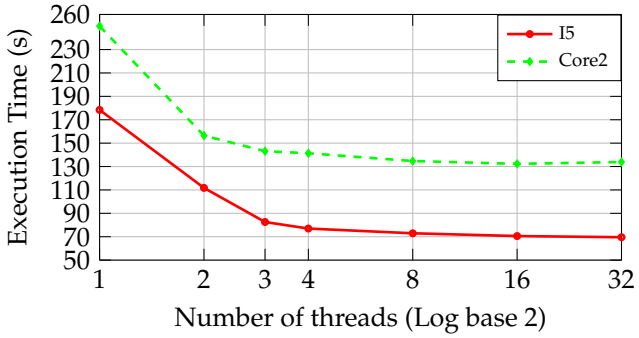

Figure 6. Execution time in the static approach.

As can be observed in Figure 6, when two cores are used in the inspection step, the execution time is reduced from 178.41 s to 111.75 s. As the third core is used for processing, a similar percentage reduction in the execution time is obtained. The performance gain is smaller than with the second and third cores when the fourth core is employed. The execution time presents an asymptotic behavior as the number of threads is increased, getting closer to the minimum time of 68 s.

Figure 6 shows analogous results in the PC with Core2 CPU. The execution time is reduced from 250.03 s to 156.43 s employing two parallel threads. Adding more than two threads does not cause a new reduction in the execution time superior to 15%, achieving a minimal value of 132.32 s. Moreover, there is a worsening of the processing time when the number of threads is greater than 16.

5.2.2. Round-Robin Approach

Figures 7 and 8 show the execution time of the inspection step as the number of threads and block size are increased in the PCs with I5 and Core2 CPUs during the round-robin approach.

According to Figure 7, the behavior of the round-robin is similar to the static approach. As the second CPU core is used, the execution time is reduced from 178.41 s to 91.81 s, achieving a performance better than obtained by the static approach. Nevertheless, the round-robin approach does not reach better performance when using the third and fourth CPU cores. Figure 7 shows that the blocks of 04 and 08 files minimize the execution time. The minimum one obtained in the PC with I5 CPU is 68.46 s, almost the same time obtained by the static approach.

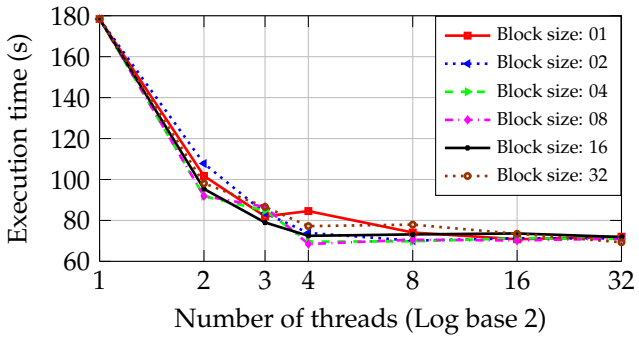

Figure 7. Execution time in the round-robin approach in the personal computer (PC) with I5 central processor unit (CPU).

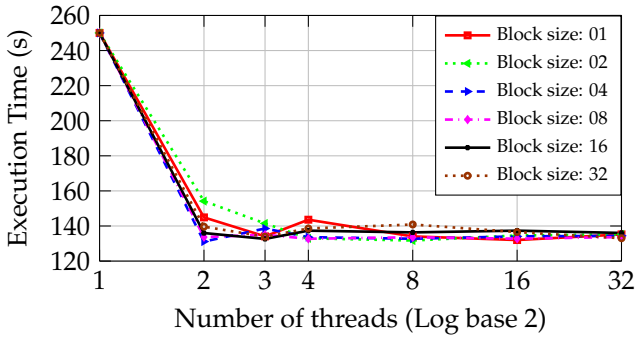

Figure 8. Execution time in the round-robin approach in the PC with Core2 CPU.

Figure 8 shows a reduction of the execution time in the PC with Core2 CPU from 250.03 s to 131.13 s when two parallel threads and a block size of 4 files are used. This last time is the minimum one obtained in the PC with Core2 CPU. Adding more threads does not improve the system performance. Moreover, the round-robin approach presents a similar behavior to the static one as the number of threads becomes superior to 16. In this situation, a worsening in the execution time occurs.

5.2.3. Dynamic Approach

Figures 9 and 10 show the execution time of the inspection step as the number of threads and block size are increased in the PCs with I5 and Core2 CPUs during the dynamic approach.

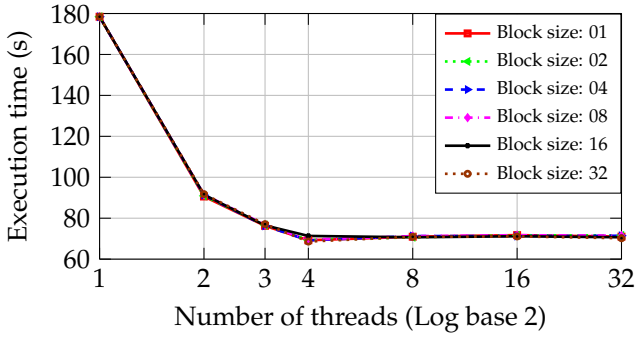

Figure 9. Execution time in the dynamic approach in the PC with I5 CPU.

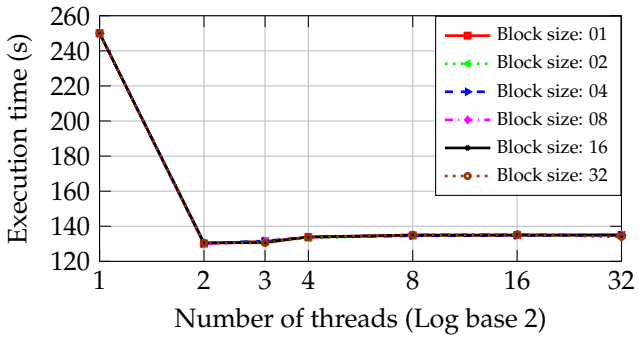

Figure 10. Execution time in the dynamic approach in the PC with Core2 CPU.

Figure 9 shows that the block size has little influence on the execution time. Therefore, the six graphics that correspond to each block of files are very close to each other. As the second CPU core is used, the execution time is reduced from 178.41 s to 90.60 s, achieving a performance slightly better than obtained by the static and round-robin approaches. As the third and fourth cores are used, the approach reaches a reduction of 16% and 10% of the execution time, respectively. It is much inferior to the 50% achieved when using the second core.

Figure 10 presents the same behavior of Figure 9, showing that the block size has almost no influence on execution time. Figure 10 shows a reduction of the execution time in the PC with Core2 CPU from 250.03 s to 129.90 s when two parallel threads are used. This last time is the minimum one obtained in the PC with Core2 CPU. Adding more threads does not improve the system performance, causing a worsening in the execution time.

5.2.4. Guided Approach

Figures 11 and 12 show the execution time of the inspection step as the number of threads and block size are increased in the PCs with I5 and Core2 CPUs during the guided approach.

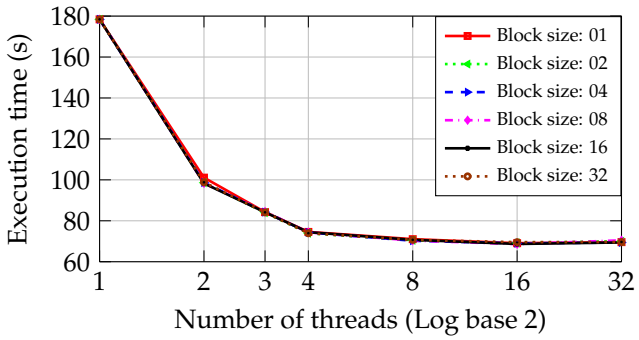

Figure 11. Execution time in the guided approach in the PC with I5 CPU.

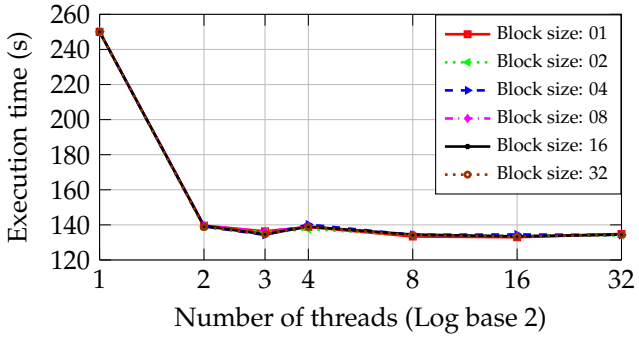

Figure 12. Execution time in the guided approach in the PC with Core2 CPU.

According to Figure 11, the behavior of the guided is very similar to the dynamic approach. As the second CPU core is used, the execution time is reduced from 178.41 s to 98.27 s, achieving a performance worse than the dynamic approach. The block size also presents little influence on the execution time during the guided approach. As the third and fourth cores are used, reductions of 8% and 3% in the execution time are obtained, respectively. This reduction is lower than the 44% achieved when using the second core.

Figure 12 presents the same behavior of Figure 11, showing that the block size has almost no influence on the execution time. Figure 12 shows a reduction of the execution time in the PC with Core2 CPU from 250.03 s to 138.81 s when two parallel threads are used. Adding one more thread (three threads) improves the system performance, causing a reduction in the execution time. This behavior was not expected in a CPU with two physical cores.

5.2.5. Comparing Through-Time Results

Figures 13 and 14 compare the two approaches in PCs with I5 and Core2 CPUs, respectively, using the optimal block size (08) to plot the round-robin, dynamic and guided approaches' graphics. As can be seen in Figure 13, the dynamic approach achieves a performance better than the other three approaches in the I5 CPU. In addition, the minimum execution time converges to close values as the number of threads becomes greater than eight.

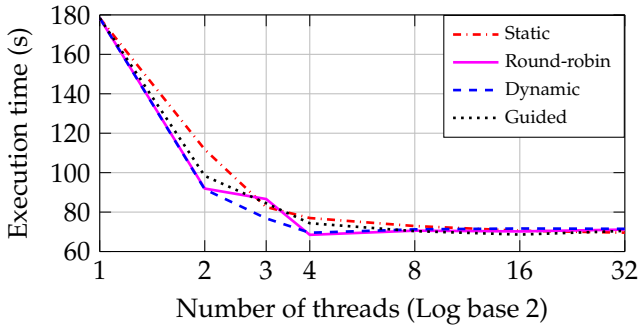

Figure 13. Comparing the four approaches in the PC with I5 CPU.

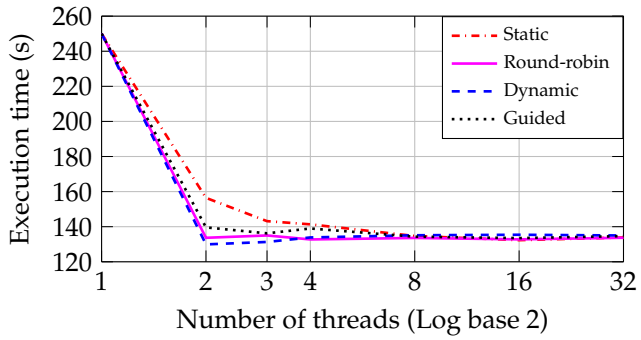

Figure 14. Comparing the four approaches in the PC with Core2 CPU.

The round-robin shows a behavior very similar to the dynamic approach when the number of threads becomes equal to or greater than four in Figure 13. This is explained by the fact that the dataset includes several files of different sizes, causing an unbalanced workload during the scanning process. Therefore, the dynamic approach presents a better performance in the processing of unbalanced tasks.

Figure 14 shows that the dynamic approach presents a better performance than the other three in the Core2 CPU. Moreover, as occurred in the I5 CPU, the minimum execution time converges to the same value as the number of threads becomes greater than eight.

It also can be seen from the Figure 14 that the round-robin approach presents a performance very similar to the dynamic one. This behavior is due to the large range of file sizes in the dataset, making the dynamic approach more adequate to process this unbalanced workload.

Table 2 presents a comparison between the minimal execution interval times obtained in the PCs with I5 and Core2 CPUs by the original AV version and each parallelization approach.

Table 2. Execution time by approach and CPU.

	Original	**This Work Modified ClamAV Version**			
	ClamAV 0.98.1v	**Static**	**Round-Robin**	**Dynamic**	**Guided**
I5	178.41	69.57	68.47	68.59	68.60
Core2	250.03	132.32	131.13	129.91	132.95

Table 2 shows that the minimal executions times are very close to each other in both CPUs. At first glance, the four parallelization approaches present almost equal performances. Nevertheless, a round-robin one requires a tuning step to discover the optimal block size of files. Three approaches, except the dynamic one, require a crescent number of threads (greater than eight) to achieve a minimum execution time. During the dynamic approach, it was only necessary to set the number of threads equal to the number of CPU cores to achieve the best performance. Moreover, the dynamic approach presents the advantage of being almost independent of the block size of files mapped to each thread.

5.3. Speedup Analysis

This subsection assesses the performance using speedup bar graphs. Firstly, it shows the results achieved with the approaches that depend on data chunk size, i.e., round-robin, dynamic, and guided. Lastly, this section compares the four methods in the test environments. The speedup is normalized to the original sequential execution.

5.3.1. Round-Robin Approach

Figures 15 and 16 show the speedup behavior in the round-robin approach when the tests increase the size of data chunks and the number of threads.

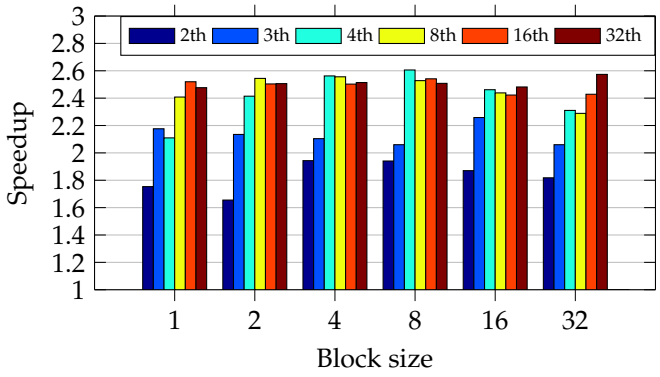

Figure 15. Speedup with the round-robin approach in the PC with I5 CPU.

As Figure 15 shows, blocks of eight files with four threads achieve the greatest speedup. In addition, the oversubscription has not caused speedup degradation in all test scenarios. The oversubscription improved the speedup during the round-robin approach with sets of 32 files, achieving the speedup of 2.6.

The round-robin schedule has not shown an expected behavior when the data chunk size and number of threads have varied. For example, Figure 15 shows an abnormal degradation with data

chunk size equal to one and number of threads equal to the number of CPU cores. It indicates a very unbalanced workload.

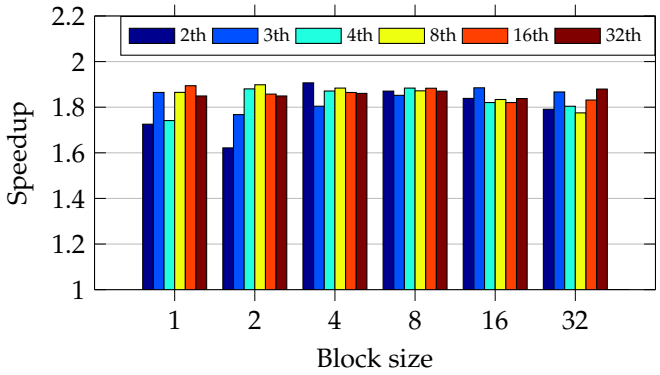

Figure 16. Speedup with the round-robin approach in the PC with Core2 CPU.

As Figure 16 shows, blocks of four files using two threads achieve the greatest speedup. The oversubscription has not caused a gain of performance. During the test scenario with data chunk size equal to 2, the oversubscription caused a linear gain of speedup. Otherwise, when the number of threads becomes greater than 8, it reduces the speedup. Figure 16 shows that data chunk size equals one, and three threads presented an unexpected speedup degradation as occurred in I5 CPU. It confirms that the workload is very unbalanced.

5.3.2. Dynamic Approach

Figures 17 and 18 compare the speedup using the dynamic schedule.

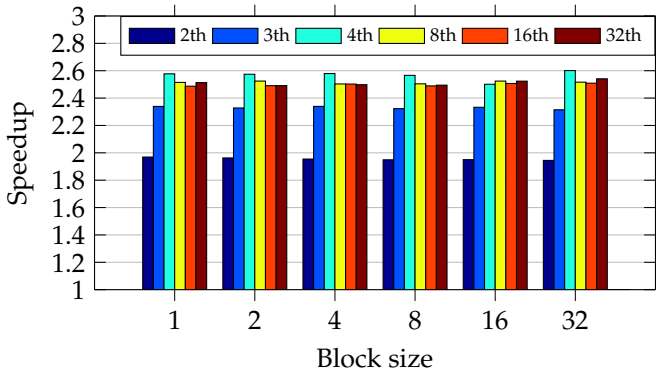

Figure 17. Speedup with the dynamic approach in the PC with I5 CPU.

Figure 17 shows that four threads achieve the greatest speedup in almost all test scenarios. Four is the number of available cores in the I5 CPU. Moreover, the oversubscription has degraded the performance in all test scenarios. The dynamical allocation of data chunks to the threads explain the performance degradation. The allocation process is done on the fly, not in the round-robin fashion. Thus, there is a growing overhead when the number of threads becomes greater than the available CPU cores.

Figure 18 shows the same behavior in the Core2 CPU as occurred in I5 CPU. The only difference is because of the dual-core CPU.

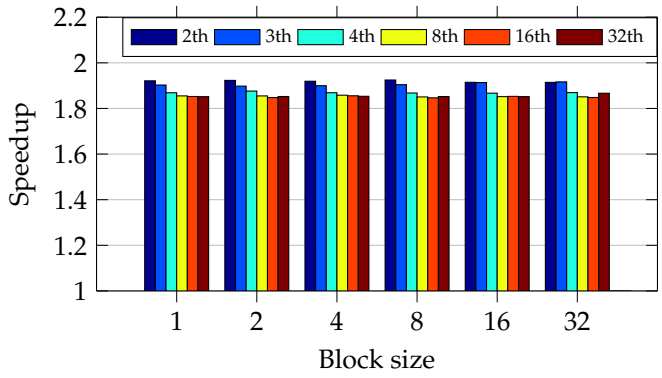

Figure 18. Speedup with the dynamic approach in the PC with Core2 CPU.

Figures 17 and 18 indicate that a dynamic schedule is a good approach for unbalanced workload because it does not depend on data chunk size. It was also possible to determine an optimal number of threads equal to the number of CPU cores.

5.3.3. Guided Approach

Figures 19 and 20 show the speedup using the guided approach.

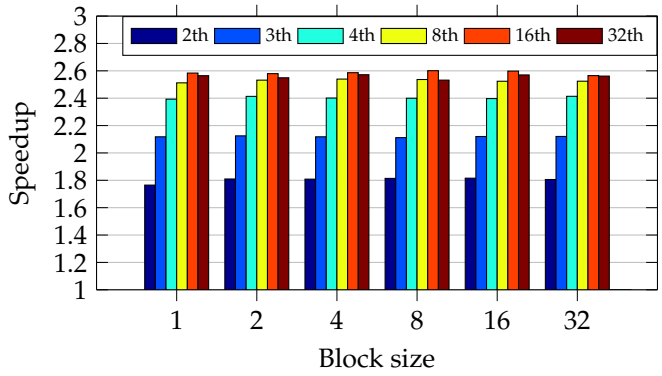

Figure 19. Speedup with the guided approach in the PC with I5 CPU.

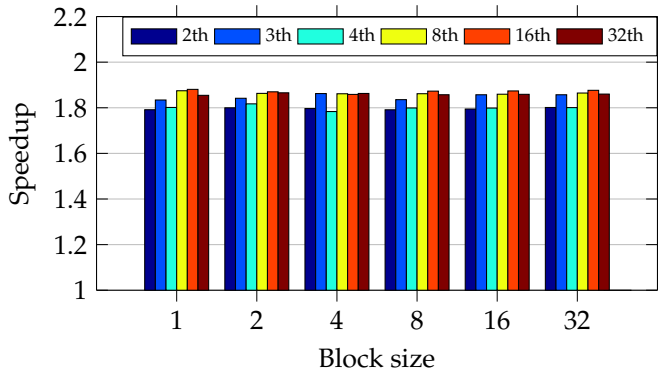

Figure 20. Speedup with the guided approach in the PC with Core2 CPU.

Figure 19 presents that 16 threads is the optimal number. Therefore, the oversubscription increases the speedup when the number of threads is smaller than or equal to 16. Above this value, there is a performance degradation. The guided schedule has presented a behavior that mixes the round-robin

and dynamic approaches. First, it shows almost no difference when the data chunk size varies, as occurred in the dynamic schedule. However, the oversubscription has improved the performance in a way similar to the round-robin schedule.

Once again, Figure 20 shows that the Core2 CPU presents a similar behavior to the I5 CPU. Oversubscription slightly improves the speedup and the performance is almost independent of the data chunk size.

Figure 20 also shows optimal speedup when the number of threads is equal to 16. The oversubscription, when limited to 16 threads, has caused a performance gain.

Figures 19 and 20 indicate that the guided schedule is a good approach for unbalanced workload because it is independent of the data chunk size and it was possible to determine an optimal number of threads equal to 16.

5.3.4. Comparing Speedup Results

Figures 21 and 22 compare the speedup of the four approaches. These two figures show that the round-robin and dynamic approaches present an overall performance better than the other two. The oversubscription improves the speedup in the static and guided approaches.

The speedup figures show that the new parallel version of ClamAV achieves a speedup of 2.6 in the I5 CPU when compared to the sequential ClamAV (version 0.98.1). The same parallel version achieves a speedup of 1.9 in the Core2 CPU.

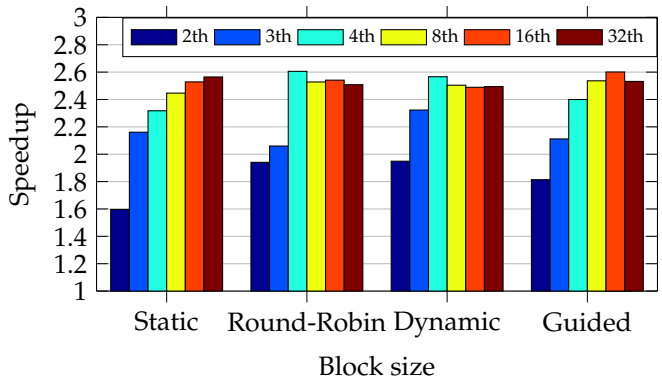

Figure 21. Speedup comparing the four approaches in the PC with I5 CPU.

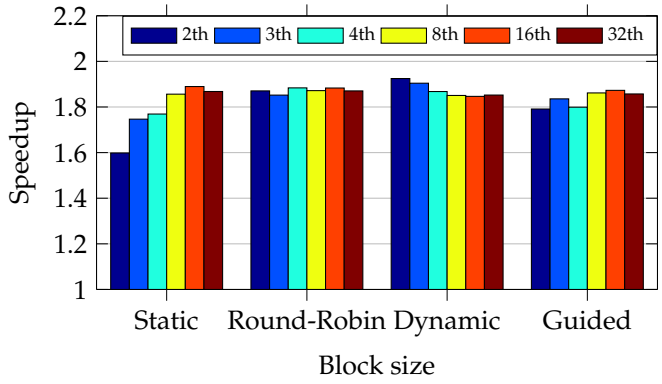

Figure 22. Speedup comparing the four approaches in the PC with Core2 CPU.

The maximum theoretical speedup is 4 and 2 with the I5 and Core2 CPUs, respectively. This is because the first CPU has four cores and the second one has two cores. Nevertheless, the I5 CPU is an SMT where each core shares several resources, reducing the overall performance. The Core2 is a CMP

with each core sharing much fewer resources when compared to the I5 architecture. It explains the speedup closer to the theoretical value in the Core2 CPU. Nevertheless, as the number of threads and cores increase, thread allocation and context switching can cause significant overhead.

6. Conclusions

This research proposed a parallel version of the sequential program ClamAV (version 0.98.1) to run on Microsoft Windows 7 OS using OpenMP library on multi-core CPU device. In addition, this work profiled the AV and concluded that the string search algorithms AC and BM were responsible for more than 90% of the computational cost of ClamAV (version 0.98.1), which recursively executes both algorithms to inspect file blocks.

Based on the aspects reviewed, this work modified ClamAV's source code and improved its performance through parallel processing on multiple cores using OpenMP library.

By studying the source code, it was noted that ClamAV did not have any parallel processing, even during the inspection of more than one file. Thus, this work implemented four OpenMP approaches to parallelize ClamAV: static, round-robin, dynamic and guided. Each one was tested on two notebooks equipped with I5-3210m e Intel Core2 Duo T6660 multi-core CPUs, respectively.

Five proofs of concept were executed on both computers over the same dataset to observe the gain of performance achieved by the parallelization schemes proposed. The first proof of concept executed the original version of ClamAV to generate a baseline for comparison. The other four tested the parallelization approaches proposed and observed the influence of the following parameters: number of threads, task scheduling and the number of CPU cores.

During the execution tests, the new parallel version of ClamAV did not present false-positives or false-negatives when compared to the original version (0.98.1). The proofs of concept performed on the I5 and Core2 CPUs reached an execution time of 62% and 48% lower than the original version of ClamAV, achieving a speedup of 2.6 and 1.9, respectively. The dynamic and guided approaches presented a better performance on both CPUs when compared to the other two methods due to the very unbalanced workload. The dynamic and guided approaches also showed almost no dependence on the data chunk size. In addition, this work concluded that a number of threads equal to 16 achieve a top speedup in the guided approach in all test scenarios.

A better performance in the I5 environment was expected, but there are time delays related to threads' scheduling because I5 has more cores than Core2. In addition, the four cores of the I5 CPU are not completely independent, sharing some CPU resources, such as arithmetic logic unit (ALU) using the Intel HT technology. Therefore, it is proposed as a future work the execution of the same test scenarios in CPUs with more than eight physical independent cores, like the Intel Xeon Phi.

Furthermore, this work leaves as a suggestion for future work a new approach of parallelization of ClamAV on multi-core CPUs based on the parallel versions of the string search algorithms AC and BM, such as prefix and substring search with DFA, using OpenMP library. This new suggested approach would permit parallel processing of ClamAV during the scanning of only one file. We also consider, as a future work, the port of ClamAV to mobile devices to take advantage of multiple core processors in such platforms.

Acknowledgments: This research work has the support of the Brazilian research and innovation Agencies CAPES – Coordination for the Improvement of Higher Education Personnel (Grant 23038.007604/2014-69 FORTE – Tempestive Forensics Project) and CNPq – National Council for Scientific and Technological Development (Grant 465741/2014-2 INCT – Science and Technology National Institute on Cyber Security), as well as the DPGU – Brazilian Union Public Defender (Grant 066/2016).

Author Contributions: Igor Forain, Robson de Oliveira Albuquerque, Ana Lucila Sandoval Orozco, Luis Javier García Villalba are the authors who mainly contributed to this research, performing experiments, analysis of the data and writing the manuscript. Tai-Hoon Kim analyzed the data and reviewed the results. All authors read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Kolbitsch, C.; Comparetti, P.M.; Kruegel, C.; Kirda, E.; Zhou, X.Y.; Wang, X. Effective and Efficient Malware Detection at the End Host. In Proceedings of the USENIX Security Symposium, Montreal, QC, Canada, 10–14 August 2009; pp. 351–366.
2. Yen, T.F.; Heorhiadi, V.; Oprea, A.; Reiter, M.K.; Juels, A. An Epidemiological Study of Malware Encounters in a Large Enterprise. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 1117–1130.
3. Vinod, P.; Jaipur, R.; Laxmi, V.; Gaur, M. Survey on Malware Detection Methods. In Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK'09), Department of Computer Science and Engineering, Prabhu Goel Research Centre for Computer and Internet Security, IIT, Kanpur, 17–19 March 2009; pp. 74–79.
4. Idika, N.; Mathur, A.P. A Survey of Malware Detection Techniques. Available online: http://profsandhu.com/cs5323_s17/im_2007.pdf (accessed on 2 February 2017)
5. Vasumathi, D.; Krishna, T.M. Network Based Anti-virus technology for Real-time scanning. *Int. J. Comput. Sci. Issues* **2012**, *9*, 304–310.
6. Vasiliadis, G.; Ioannidis, S. Gravity: A Massively Parallel Antivirus Engine. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Ottawa, ON, Canada, 15–17 September 2010; Springer: Berlin, Germany, 2010; pp. 79–96.
7. Cha, S.K.; Moraru, I.; Jang, J.; Truelove, J.; Brumley, D.; In additionersen, D.G. SplitScreen: Enabling Efficient, Distributed Malware Detection. *J. Commun. Netw.* **2011**, *13*, 187–200.
8. Bal, H.E.; Haines, M. Approaches for Integrating Task and Data Parallelism. *IEEE Concurr.* **1998**, *6*, 74–84.
9. Ali, A.; Dastgeer, U.; Kessler, C. OpenCL for Programming Shared Memory Multicore CPUs. In Proceedings of the 5th Workshop on MULTIPROG, Paris, France, 23–26 January 2012.
10. Chapman, B.; Jost, G.; Van Der Pas, R. *Using OpenMP: Portable Shared Memory Parallel Programming*; MIT Press: Cambridge, MA, USA, 2008; Volume 10.
11. Jin, H.; Jespersen, D.; Mehrotra, P.; Biswas, R.; Huang, L.; Chapman, B. High Performance Computing using MPI and OpenMP on Multi-core Parallel Systems. *Parallel Comput.* **2011**, *37*, 562–575.
12. Terboven, C.; Schmidl, D.; Jin, H.; Reichstein, T.; Mey, D. Data and Thread Affinity in Openmp Programs. In Proceedings of the Workshop on Memory Access on Future Processors: A Solved Problem? Ischia, Italy, 5–7 May 2008; pp. 377–384.
13. Komatsu, K.; Sato, K.; Arai, Y.; Koyama, K.; Takizawa, H.; Kobayashi, H. Evaluating Performance and Portability of OpenCL Programs. In Proceedings of the Fifth International Workshop on Automatic Performance Tuning, Berkeley, CA, USA, 22–25 June 2010; Volume 66, p. 1.
14. Xu, R.; Chandrasekaran, S.; Chapman, B. Exploring programming multi-GPUs using OpenMP and OpenACC-based hybrid model. In Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), Cambridge, MA, USA, 20–24 May 2013; pp. 1169–1176.
15. Shen, J.; Fang, J.; Sips, H.; Varbanescu, A.L. Performance Gaps between OpenMP and OpenCL for Multi-Core CPUs. In Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW), Pittsburgh, PA, USA, 10–13 September 2012; pp. 116–125.
16. Liao, C.; Liu, Z.; Huang, L.; Chapman, B. Evaluating OpenMP on Chip Multithreading Platforms. In *OpenMP Shared Memory Parallel Programming*; Springer: Berlin, Germany, 2008; pp. 178–190.
17. Mittal, S.; Vetter, J.S. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.* **2015**, *47*, 69.
18. Sabne, A.; Sakdhnagool, P.; Lee, S.; Vetter, J.S. Evaluating performance portability of OpenACC. In Proceedings of the International Workshop on Languages and Compilers for Parallel Computing, Raleigh, NC, USA, 15–17 September 2014; Springer: Berlin, Germany, 2014; pp. 51–66.
19. ClamAV Anti-Virus. Available online: http://www.clamav.net (accessed on 3 May 2016).
20. Miretskiy, Y.; Das, A.; Wright, C.P.; Zadok, E. Avfs: An On-Access Anti-Virus File System. In Proceedings of the USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004; pp. 73–88.
21. Oberheide, J.; Cooke, E.; Jahanian, F. CloudAV: N-Version Antivirus in the Network Cloud. In Proceedings of the USENIX Security Symposium, San José, CA, USA, 28 July–1 August 2008; pp. 91–106.

22. Huang, N.F.; Tsai, W.Y. SHOCK: A Worst-Case Ensured Sub-Linear Time Pattern Matching Algorithm for Inline Anti-Virus Scanning. In Proceedings of the IEEE International Conference on Communications (ICC), Cape Town, South Africa, 23–27 May 2010; pp. 1–5.
23. Aho, A.V.; Corasick, M.J. Efficient string Mmtching: An aid to bibliographic search. *Commun. ACM* **1975**, *18*, 333–340.
24. Wu, S.; Manber, U. *A fast algorithm for multi-pattern searching*; Technical Report TR-94-17; Department of Computer Science, University of Arizona, Tucson, AZ, USA, 1994.
25. Boyer, R.S.; Moore, J.S. A Fast String Searching Algorithm. *Commun. ACM* **1977**, *20*, 762–772.
26. Che, S.; Boyer, M.; Meng, J.; Tarjan, D.; Sheaffer, J.W.; Lee, S.H.; Skadron, K. Rodinia: A benchmark Suite for Heterogeneous Computing. In Proceedings of the IEEE International Symposium on Workload Characterization, Austin, TX, USA, 4–6 October 2009; pp. 44–54.
27. Schmidl, D.; Cramer, T.; Wienke, S.; Terboven, C.; Müller, M.S. Assessing the performance of openmp programs on the intel xeon phi. In Proceedings of the European Conference on Parallel Processing, Aachen, Germany, 26–30 August 2013; Springer: Berlin, Germany, 2013; pp. 547–558.
28. Curtis-Maury, M.; Ding, X.; Antonopoulos, C.D.; Nikolopoulos, D.S. An evaluation of OpenMP on current and emerging multithreaded/multicore processors. In *OpenMP Shared Memory Parallel Programming*; Springer: Berlin, Germany, 2008; pp. 133–144.
29. Iancu, C.; Hofmeyr, S.; Blagojević, F.; Zheng, Y. Oversubscription on multicore processors. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), Atlanta, GA, USA, 19–23 April 2010; pp. 1–11.
30. Erdogan, O.; Cao, P. Hash-AV: Fast virus signature scanning by cache-resident filters. *Int. J. Secur. Netw.* **2007**, *2*, 50–59.
31. Ho, J.T.L.; Lemieux, G.G. PERG: A Scalable FPGA-Based Pattern-Matching Engine with Consolidated Bloomier Filters. In Proceedings of the International Conference on ICECE Technology, Dhaka, Bangladesh, 20–22 December 2008; pp. 73–80.
32. Lin, Y.D.; Lin, P.C.; Lai, Y.C.; Liu, T.Y. Hardware-software codesign for High-speed signature-based virus scanning. *IEEE Micro* **2009**, *29*, doi:10.1109/MM.2009.81.
33. Lee, C.L.; Yang, T.H. A flexible pattern-matching algorithm for network intrusion detection systems using multi-core processors. *Algorithms* **2017**, *10*, 58.
34. Kamil, A.; Yelick, K. Hierarchical computation in the SPMD programming model. In Proceedings of the International Workshop on Languages and Compilers for Parallel Computing, San José, CA, USA, 25–27 September 2013; pp. 3–19.
35. Schauer, B. Multicore Processors—A Necessity. In *ProQuest Discovery Guides*; ProQuest: Ann Arbor, MI, USA, 2008; pp. 1–14.