Digital Investigation 24 (2018) S29–S37

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2018 Europe — Proceedings of the Fifth Annual DFRWS Europe

OpenForensics: A digital forensics GPU pattern matching approach for the 21st century

E. Bayne*, R.I. Ferguson, A.T. Sampson

School of Design and Informatics, Abertay University, Dundee, Scotland, United Kingdom

Keywords:
Digital Forensics
Processing model
Pattern matching
Asynchronous processing
GPU
GPGPU

A B S T R A C T

Pattern matching is a crucial component employed in many digital forensic (DF) analysis techniques, such as file-carving. The capacity of storage available on modern consumer devices has increased substantially in the past century, making pattern matching approaches of current generation DF tools increasingly ineffective in performing timely analyses on data seized in a DF investigation. As pattern matching is a trivally parallelisable problem, general purpose programming on graphic processing units (GPGPU) is a natural fit for this problem. This paper presents a pattern matching framework – *OpenForensics* – that demonstrates substantial performance improvements from the use of modern parallelisable algorithms and graphic processing units (GPUs) to search for patterns within forensic images and local storage devices.

Introduction

DF investigation has struggled to remain effective in recent times, as the capacity of storage devices seized as part of a DF investigation has grown substantially in the past 10 years. The continuing increase in capacity of consumer storage devices requires similar improvements to the performance of pattern matching techniques employed by DF tools used to analyse forensic data. Recently, in 2015, a report by Her Majesty's Inspectorate of Constabulary (HMIC) found that delays of up to 12 months or more were "not uncommon" after the inspection of 124 cases from six UK police forces (Her Majestys Inspectorate of Police, 2015).

It could be argued that processing approaches within DF have been a largely neglected research area since the conception of the first generation of commercial DF analytical tools. Whilst pattern matching algorithms and use of massively-parallel hardware have provided significant breakthroughs in other data analysis areas (Bellekens et al., 2014; Hung et al., 2014), the tools that drive DF investigation have largely failed to incorporate recent innovative advances in pattern matching techniques and processing technology. The most current systematic reviews of the field have acknowledged that processing of DF corpora remains one of the greatest challenges of the field at the time of writing (Raghavan, 2013; Garfinkel, 2010; Beebe, 2009), yet the problem has attracted little research or commercial interest. Pattern matching is a technique that underpins many DF analysis techniques, such as *file-carving*—where disk images are analysed in the rawest form to find and reproduce files when a partition table or other metadata is corrupted or missing.

This paper presents a processing approach for conducting pattern matching within a DF context to perform file-carving. The processing approach incorporates asynchronous parallelisation of available central processing unit (CPU) and GPU devices and conducts string searching with the aid of the *Parallel Failureless Aho-Corasick (PFAC) algorithm* (Lin et al., 2013) to significantly reduce the time required to perform pattern matching on raw data.

The proposed methods discussed in this paper are intended to provide effective techniques for performing complete analysis of DF corpora for the next 10 years, due to their scalable approach with the processors available on the analyst's machine. By making use of all usable compute resources, processing speed is evidenced to be notably improved—reducing the time required to analyse large amounts of forensic evidence.

Background

The file-carving process is conducted in several phases, each of which can impact directly on performance. Processing can be analysed in three areas: the *detection and processing method*, the

* Corresponding author.
E-mail addresses: e.bayne@abertay.ac.uk (E. Bayne), ian.ferguson@abertay.ac.uk (R.I. Ferguson), a.sampson@abertay.ac.uk (A.T. Sampson).

https://doi.org/10.1016/j.diin.2018.01.005

data reading method, and the *file reproduction method*. Current research in the field has proposed achieving enhancements to the detection and processing task through novel ideas, such as employing low-cost clusters of commodity PCs to process data (Ayers, 2009), or through applying Bloom filter hashing algorithms to detect known file hashes in raw data (Penrose et al., 2015). This paper will present potential improvements to the physical investigation of a storage device, assuming the requirement that all forensic data requires analysis with 100% false-negative pattern matching accuracy.

File detection and processing method

Performing pattern matching on data is the most time-consuming processing task in any file-carving operation once data has been loaded to memory. The two important factors for performing pattern matching are the *processor* and *algorithm* used. Most free and commercial file-carvers rely on CPUs to conduct pattern matching. However, as pattern matching is a trivially-parallelisable problem, GPGPU approaches are a natural fit. Currently, only some of the research in employing GPGPU has been transferred to the field of DF (Marziale et al., 2007; Zha and Sahni, 2011), of which, a closed-source GPGPU framework was used—Complete Unified Device Architecture (CUDA) (Nvidia). Findings from these earlier studies have found that local storage devices from which data is read present an insurmountable performance bottleneck. This research aims to reinstate GPGPU processing techniques as a well-suited approach to performing pattern-matching in a modern DF context.

The algorithm employed to conduct pattern matching can have a significant effect on the time taken to perform file-carving on forensic data. Pattern matching algorithms search for one – or more – patterns within a corpora of data. Today, popular open-source file-carvers – *Scalpel* (Richard and Roussev, 2005) and *Foremost* (Kendall et al.) – still employ a modified *Boyer-Moore (BM) algorithm* (Boyer and Moore, 1977) to conduct pattern matching for multiple patterns.

The BM algorithm searches for a pattern with the aid of a skip table to accelerate searching. This skip table operates by calculating when the next potential match could occur in a serial read of data. Research that measures the effectiveness of pattern matching algorithms (Skrbina and Stojanovski, 2012) recognises the BM algorithm as an effective method for single-pattern searching, but less efficient for multi-pattern searching. In the same study, it was proposed that the *Aho-Corasick (AC) algorithm* (Aho and Corasick, 1975) was the most suitable approach when searching for multiple patterns. The PFAC algorithm is an extension of AC that adapts the algorithm for massively-parallel execution (Lin et al., 2013).

Data reading method

The method used to read data from storage device is an aspect of file-carving that is rarely discussed. However, we propose that this is an important aspect that should be considered when discussing file-carving performance—in particular, the method that file-carvers use to read data from the analysed storage device to memory. Furthermore, many existing studies often fail to present performance metrics of the storage devices used, which arguably weakens their argument when concluding that file-carving performance is limited by the storage device's I/O transfer speeds. We do not argue that this is not the case; rather, we propose that any research that discuss file-carving performance should measure the potential data throughput of the storage devices tested. Through the presentation of these metrics, the reader will have more finite detail on the effectiveness of the proposed processing solution.

Reading data from storage devices is an accepted component of testing performance in other DF file-carving studies (Richard and Roussev, 2005; Marziale et al., 2007; Zha and Sahni, 2011). When reading data from a storage device, *threads* and *queue-depth* are equally important factors here as they are in processing data—if not, more so, as storage devices are recognised as the most probable area to present a processing bottleneck in file-carving. Threads – akin to how they function elsewhere – relate to how many processing threads are allocated to read data from the storage device. Queue-depth specifies how many read instruction tasks are queued at a single point for the storage device to process. Typically, in order to achieve a storage device's maximum I/O data throughput, a file-carver must exploit the use of threading and queue-depth.

File reproduction method

File reproduction is considered to be another important factor when discussing file-carving performance. Specifically, when in the file-carving process that files are reproduced from the data analysed. One of the most basic methods of reproducing files is to simply save any data found between two points – a *file header* and *file footer* – to a new file.

File-carvers may adopt further processing tasks to enhance accuracy and ensure integrity of reproduced files. For certain file types, there are further possible checks that could be completed to ensure that files are reproduced using the correct start and end point in data. It is also possible to include optional checks to check the file integrity of certain file types, such as looking for defining features of a file's structure. There has been research that incorporates intuitive methods of accounting for fragmentation (Garfinkel, 2007); however, it is important to acknowledge a potential trade-off of processing speed to make more advanced checks.

Many existing file-carvers employ a two-pass approach to file-carving—the first pass to identify potential files and locations in data, and the second pass to reconstruct files from data. A two-pass approach to file-carving can be computationally expensive, as data has to be read from the storage device twice. However, if the file-carver performs additional processing – such as file verification processing or fragmentation checks – the two-pass approach may have performance advantages, as pattern matching is less likely to be halted by overlapping I/O and computation tasks.

Methodology

In this section, we outline and discuss the approach taken with our proposed solution—*OpenForensics*. OpenForensics is a file-carving application that was produced in C# to demonstrate the advantages of the pattern matching approach outlined in this paper. For validation of these techniques, the OpenForensics binaries and source code are freely available from GitHub (Bayne).

File carving approach

This research focuses on investigating and improving the processing speed of pattern matching in a file-carving context. In order to achieve this, fundamental changes were made to the three aforementioned processing stages of file-carving. The following subsections will outline and discuss the changes presented.

File detection and processing method

In our solution, we propose a scalable asynchronous approach for performing pattern matching on data. We define this approach as employing processing threads that act independently from each other to queue, read, and process data. The advantage of this processing approach is hypothesised to scale well with the level of

processing power available on analysts' computers when tasked to perform file-carving on data. Through employing all available processing resources, processors would be less likely to be bottlenecked when performing analysis on high data-throughput storage devices.

Our solution employs two approaches to file detection and processing—GPU and CPU processing. The processing approach undertaken can be seen in Fig. 1.

When performing GPU processing, forensic data is processed on graphics processors using *Open Computing Language (OpenCL)* (Khronos Group). The GPU operation is able to scale with the amount of available graphics processors present on the system, which enables the searching to employ all of the available processing power on the system. During this operation, the GPU devices available on the system perform pattern matching processing, whilst the CPU is tasked with validation and recording of files found.

In GPU operation, the available CPU threads are divided equally between the usable GPUs—e.g. on a system with 2 GPUs and an 8 logical core processor, each GPGPU device is allocated 4 independent CPU processing threads. The level of threading employed, however, is limited by the amount of system memory available, as each processing thread employed requires around 150 MiB of RAM and VRAM to run.

Performing processing on the CPU – intended for systems that do not have any form of graphics processor – employs concurrency on multi-threaded CPUs to accelerate searching. In CPU operation, a thread is employed per each logical core available on the CPU. Each thread performs both pattern matching and file matching processing sequentially for each allocated data segment.

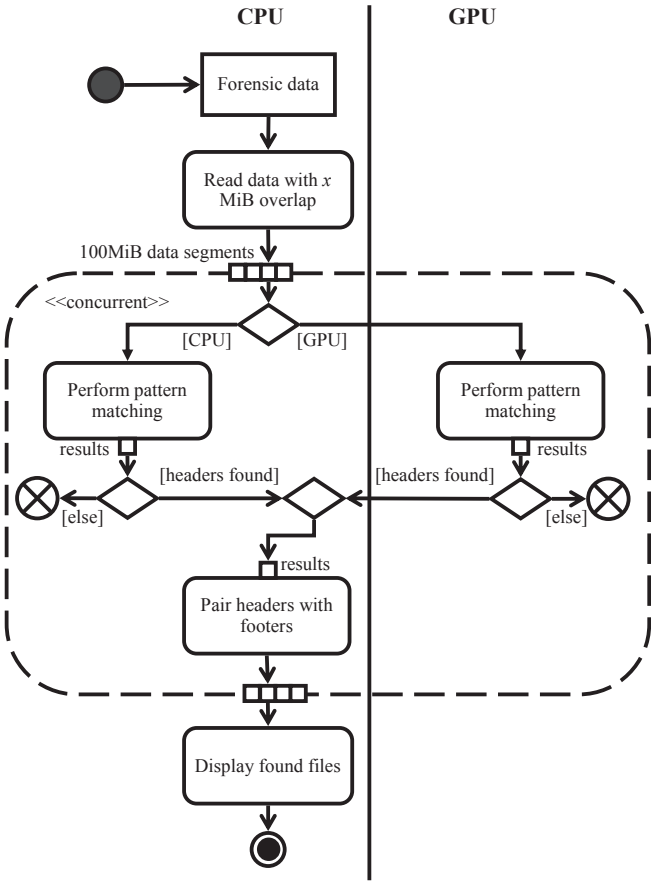


Fig. 1. The OpenForensics pattern matching process.

OpenForensics employs the PFAC algorithm to accelerate GPU and CPU pattern matching operations. The implementation of the PFAC algorithm entails two entities; the pre-processing of searched-for file headers and footers (*patterns*) to formulate a state transition table (STT), and the processing steps that both CPU and GPU devices followed.

The PFAC STT generation – as shown in Algorithm 1 – generates a state machine that is processor agnostic, in that both CPU and GPU implementations can follow the STT to look up their next instruction.

Algorithm 1. PFAC STT generation.

Algorithm 1 PFAC STT generation

```
1: patterns is a 2D array of byte
2: STT is a variable-length array of integer
3: row is an array[256] of integer
4: sort patterns array by byte length, a < b
5: add fail state row to STT (STT[0])
6: for i = 0 to num of patterns do
7:    add row to STT (pattern found state)
8: end for
9: add initial state row to STT
10: state ← num of patterns + 1
11: for i = 0 to num of patterns do
12:    walkindex ← num of patterns + 1
13:    if STT[walkindex][target[i][0]] = 0 then
14:       STT[walkindex][target[i][0]] ← state
15:       state += 1
16:    end if
17:    walkindex ← STT[walkindex][target[i][0]]
18:    for j = 0 to patterns[i] length do
19:       if STT length <= walkindex then
20:          add row to STT (transition state)
21:       end if
22:       if STT[i][patterns[i][j]] = 0 then
23:          if j != patterns[i] length − 1 then
24:             STT[i][patterns[i][j]] ← state
25:             state += 1
26:          else
27:             STT[i][patterns[i][j]] ← i + 1
28:          end if
29:       end if
30:       walkindex ← STT[i][patterns[i][j]]
31:    end for
32: end for
33: return STT
```

The generation of the PFAC STT is completed by the CPU before searching begins. It requires an input of the patterns searched in the format of a byte array and returns an output of the STT. The STT used by OpenForensics is a 2D array of integers (Fig. 2). Within this 2D array, rows are referred to as *states* and columns represent *input value*. The STT consists of four key state types. The *fail state* (row 0) signals that no pattern matches have been found. After the fail state, a *pattern state* is created in the 2D array to provide a unique identifier for each pattern searched for. Following this, an *initial state* is created and used as an initial search state for each byte of data. Lastly, *transition states* are used by the processor to decide the next state based on the data value read. The number of transition states generated by the STT varies depending on the number and length of patterns searched. Each row of the STT has a column for

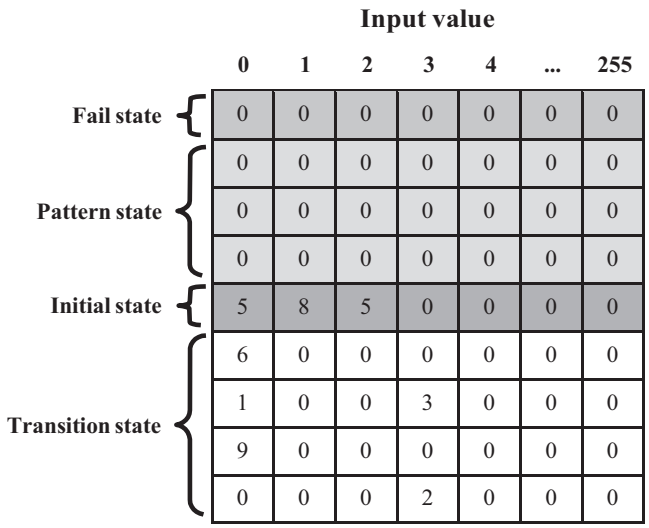


Fig. 2. OpenForensics state transition table (STT).

the number of possible inputs that the processor can read when searching—in this application, 256 columns are created that represent each possible value for a byte read in data.

The processing steps shown in Algorithm 2 present the logic used by the CPU and GPU to accomplish searching for patterns using the generated STT. The algorithm begins on the initial state row of the STT for each individual byte of data analysed. The STT is used by the processor to transition state depending on the value read from each following byte. This process continues until the byte value read has a state value of 0 – which indicates a fail state – or until the state value is less than the initial state with no further transitions available, indicating a pattern match. If a pattern is matched by the STT, the starting byte is flagged in memory as being a match for the pattern.

The 100 MiB segments of data processed by OpenForensics are allocated to a unique asynchronous GPU or CPU processing thread. Whilst the CPU and GPU methods of OpenForensics utilise the same PFAC approach to processing, the GPU operation is able to analyse significantly more bytes simultaneously, achieving greater processing efficiency than CPU methods.

Algorithm 2. PFAC processing steps.

Algorithm 2 PFAC processing steps

```
1: n ← segment length
2: initial state ← num of patterns + 1
3: for i = 0 to n do
4:    state ← initial state
5:    pos ← i
6:    while pos < n do
7:       state = STT[state, segment[pos]]
8:       if state = 0 then
9:          break
10:      end if
11:      if state < initialstate then
12:       record position i as match for pattern state
13:      end if
14:   end while
15: end for
16: return found patterns
```

It is envisioned that the PFAC algorithm employed by OpenForensics would benefit CPU and GPU processing when tasked with searching for multiple patterns within data, as all defined patterns are searched with a single read of the target source. The algorithm is expected to significantly reduce the time taken to complete searches for large amounts of patterns when compared to alternative algorithms, such as the modified BM algorithm that Foremost and Scalpel employs.

The file detection method adopted by OpenForensics is intentionally basic, as the focus of this research aims to measure and improve the processing rate of file-carving. A two-stage approach is adopted for searching. The first pass marks files found within data, and the second pass optionally performs file-carving of found files. On the first pass, searching is conducted for patterns and locations are stored in an array. The locations found from the search are then passed to the CPU, which scans the result array for a file header, then pairs it with a matching file footer. This is repeated until all 100 MiB segments of the target file or storage device have been scanned.

To account for partial patterns that may occur at the end of segments, the search utilises a windowed technique, where each segment is overlapped by a frame that is the size of the maximum defined file-size (10 MiB by default). The windowed technique is performed by OpenForensics by storing the frame of bytes from the end of each segment in memory and attaching to the start of the next segment.

Data reading method

To take full advantage of storage devices, an investigation was carried out to discover the optimal settings to perform data reading in a DF context. As part of this investigation, many factors were considered and benchmarked. The data reading method was greatly improved through analysing the behaviour of storage device benchmarking tools. Threads and queue depth were principally the largest influencing factors to performance. However, other variables that may impact the performance of reading the storage device were also considered as part of this investigation, such as the size of the stream buffer and the amount of data read by each read instruction queued.

It was decided to allocate a single thread to read the data with a queue depth of 32 read instructions, akin to the default settings of CrystalDiskMark (CrystalMark)—a popular storage device performance benchmarking software. Only a single thread was used so that it would not interfere with the performance of other asynchronous threads employed for processing. For multi-threaded CPU operation, where all logical cores of the CPU were used to process data, employing more threads to read data may have disadvantaged active pattern matching threads when employed with smaller-levels of parallelism. However, allowing a queue depth of 32 read instructions provided the single read thread enough queued read instructions to make efficient use of the possible transfer-rate of the storage devices analysed. The thread and queue-depth settings were trialled with a range of storage devices—traditional hard disk drives (HDDs), solid state drives (SSDs), and NVMe SSDs. The settings achieved data transfer rates similar to the sequential read performance stated by the storage device manufacturer.

The stream buffer is a setting which specifies the size of buffer to use to read data from the physical storage device to memory. As the data from the storage device is transferred into memory by the CPU using the .NET library *FileStream* method, the size of the stream buffer defined is largely limited by the size of the internal cache of the CPU. Both CPUs tested in this study possessed a fairly large internal cache, so it was assumed that increasing the stream buffer size may have had a positive effect on data transfer speed. However, experiments that expanded the stream buffer to 8, 16 and 32 KiB

showed that larger stream buffer sizes did not produce faster transfer rates of data on storage device to memory. In conclusion, it was found that maintaining a small stream buffer size of 4 KiB remains effective, whilst maintaining a wider compatibility with processors equipped with smaller memory caches.

Further benchmarking was performed to find the optimal queued read instruction length to read forensic data from storage devices. Data read by each queued task was tested in sizes of 32, 64, 128, 256, 512, 1024, and 2048 KiB segments respectively. Results from testing the various sizes of data read by each queued task revealed that sizes smaller than 256 KiB performed significantly slower than larger sizes when reading the forensic file from the storage device; additionally, larger segment sizes of 1024 and 2048 KiB were found to produce the most consistent results when reading the storage device multiple times. From the observations of the trial, a segment size of 1024 KiB was found to be optimal for each queued read task.

Fig. 3 outlines the data reading method of how OpenForensics reads data from storage devices. Illustrated are how the components, as previously discussed in this section, all work together. The storage device is read by the CPU with a stream buffer size of 4 KiB to fill the 1 MiB queued read task; which builds up the 100 MiB data segment requested by the method invoked to fetch the next segment of forensic data.

File reproduction method

OpenForensics performs file reproduction by performing a second pass on the data analysed. This pass specifically extracts data between two points on the storage device that has been flagged to contain a file. This pass behaves differently from the first pass as it does not read all data from the storage device to memory to perform file-carving. Instead, only the data between matched file headers and footers are read. All data found between paired headers and footers are saved to new files within the directory set by the investigator. Once all found files have been reproduced, the file-carving process is complete.

During the second pass, the files are read using similar methods to the first pass. This phase employs an asynchronous thread for each logical core on the CPU. For each file, an available thread is tasked to read the data between file header and footer locations. The thread queues 1 MiB read instructions to read the file from the storage device.

It is important to note that, as the focus of this research is specifically aimed at accelerating pattern matching in DF, OpenForensics does not conduct any additional file processing or verification tasks other than extracting data between found headers and footers. It is, therefore, reasonable to assume that OpenForensics would have a processing advantage over file-carvers that may incorporate more complex file reproduction methods.

Testing methodology

To compare the performance of the OpenForensics searching methodology, this research has presented performance results of performing pattern matching on connected hard disk drives and a forensic image file. The performance achieved by OpenForensics v.1.5.1b has then be compared with other currently available open-source and commercial file-carving software – Foremost v.1.5.7–6 and Recover My Files v.6.1.2 (2375) – in performing the same tests.

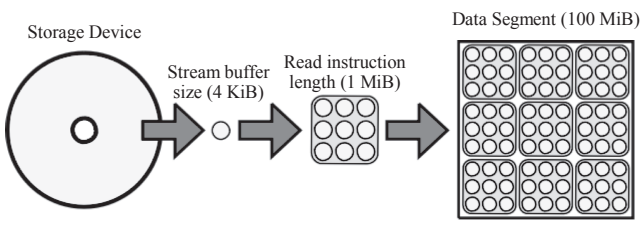


Fig. 3. Design of data reading method.

Testing was conducted with a high-end workstation PC (test platform A) and a high-end laptop (test platform B), specifications of both test platforms are given in Table 1. Both test platforms are configured to dual-boot Windows 10 Pro and Ubuntu GNOME 17.04, located on storage device 2. Both OpenForensics and Recover My Files are Windows binaries and were tested with Windows 10 Pro. Foremost, on the contrary, was tested under Linux using Ubuntu GNOME 17.04.

Measurements were taken on the time required to search for predetermined patterns outlined in Table 2. For Foremost and OpenForensics, two groups of patterns were searched for each test. The first test involves performing pattern matching for all 45 unique patterns. The second group involves searching for a smaller group of 10 unique patterns – highlighted on the table – to show relationships between defined search patterns and time taken. Recover My Files cannot explicitly be instructed to search for specific patterns. To achieve a reasonable comparison, Recover My Files was instructed to search for a set of 19 file types in a *many pattern test* and 4 file types in a *lesser pattern test*. Tests were repeated 5 times each and a mean average time taken to perform each test was used for analysis.

Two types of test were performed. The first test type is a raw forensic file analysis, where a *dd* image of an external storage device is analysed from the slowest storage device on the test beds—storage device 1. On test platform A, storage device 1 is a software RAID0 array consisting of 2 SSD storage devices operating over a SATA3 bus. On test platform B, storage device 1 is a single SSD storage device operating over SATA3. We hypothesise with this test that storage device transfer speeds would limit the possible rate of analysis.

The methodology behind creating the forensic image for the first test was not to simulate a realistic scenario, but rather to know the ground truth of how many files of each format were contained in the image used. The files loaded on the 20 GB drive also exhausted the space available, leaving little unused space on the drive. Whilst the data on the storage device is not deemed to be a realistic case, the tests performed within this research was interested in the comparative performance and accuracy between the proposed and existing processing methods. It is assumed that the observed performance differences when performing string searching or file-carving operations on the simulated forensic data would not vary significantly when tasked with different data.

The second test performs pattern matching on storage device 2 of each platform, where the data is physically accessed from the storage device to memory. With both test platforms, NVMe storage devices are used, which communicate over a PCIe bus. The PCIe bus allows storage devices to benefit from significantly faster read and write speeds than storage devices operating over the more prevalent SATA3 bus. We hypothesise that this test would highlight limitations of current string searching approaches adopted in DF file-carving tools.

Contrary to the first test, the physical storage devices used in the second test mimic a realistic test scenario, where both storage devices have been used for over a year with a Windows 10 operating system. No ground truth is known about the files contained on either storage device; however, both storage devices report that greater than 80% of the overall capacity is used within the Windows disk management utility.

Table 1
Test platform specifications.

Test Platform	A	B
Type	Desktop	Laptop
Operating System	Windows 10 Pro & Ubuntu GNOME 17.04	Windows 10 Pro & Ubuntu GNOME 17.04
Processor	Intel Core i7-5820K	Intel Core i7-7700HQ
Processor Specification	6 Core/12 Thread @ 3.8 GHz	4 Core/8 Thread @ 3.8 GHz
Memory	16 GiB DDR4 2400 MHz	32 GiB DDR4 2400 MHz
GPU	Nvidia Titan XP (12 GiB GDDR5X)	Nvidia 1070 (8 GiB GDDR5)
GPU Specification	3584 Cuda cores @ 1417 MHz	2048 Cuda cores @ 1442 MHz
Storage Device 1	2x Samsung Evo 940 250 GB SATA3 SSD (RAID0)	SanDisk Ultra II 960 GB SATA3 SSD
Storage Device 2	Samsung Evo Pro 256 GB NVMe PCIe SSD	Toshiba THNSN5256GPUK NVMe 256 GB SSD

Table 2
Defined search patterns.

File Type	File Header (File Footer)
jpg	0×FFD8FFE00010 (0×FFD9)
jpg	0×FFD8FFE135FE (0×FFD9)
gif	0×474946383961 (0×003B)
gif	0×474946383761 (0×003B)
png	0×89504E470D0A1A0A (0×49454E44AE426082)
mpg	0×000001BA (0×000001B7)
mpg	0×000001B3 (0×000001B7)
docx	0×504B030414000600 (0×504B0506)
pdf	0×25504446 (0×0A2525454F46)
tiff	0×49492A00
tiff	0×4D4D002A
wim	0×4D5357494D
mp4	0×00000014667479706973F6D
mp4	0×00000018667479703367 7035
mp4	0×0000001C667479704D534E56012900464D534E566D703432
mov	0×00000014667479707 1742020
m4v	0×00000018667479706D703432
wmv	0×3026B2758E66CF11A6D900AA0062CE6C
mkv	0×1A45DFA3934282886D6174726F736B61
wma	0×3026B275
m4a	0×00000020667479704D344120
doc	0×D0CF11E0A1B1
zip	0×504B0304
zip	0×504B0506
zip	0×504B0708
zip	0×504B03041400010063000000000
rar	0×526172211A0700
rar	0×526172211A070100
xar	0×78617221
xz	0×FD377A585A00
jar	0×4A4152435300
jar	0×5F27A889
iso	0×4344303031
cso	0×4349534F
img	0×504943540008
img	0×514649FB
img	0×53434D49
cas	0×5F434153455F
rpm	0×EDABEEDB
mof	0×FFFE23006C0069006E00650020003100

Results and discussion

This section presents the results from conducting pattern searching with the test platforms.

Analysis of a raw forensic dd file

The first test analyses a 20 GiB forensic image file on storage device 1 of each test platform. On both test platforms, the solid-state storage devices used as storage device 1 perform relatively faster than traditional mechanical storage devices, but also considerably slower than the NVMe storage device present on each test platform.

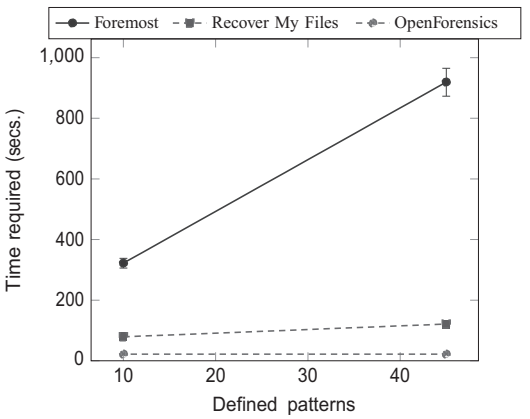
Fig. 4 presents the mean time taken to analyse the forensic image on test platform A with the respective file-carvers. OpenForensics finished analysis in 22 s for both 10 and 45 pattern tests. Furthermore, OpenForensics completed these tests with no variation in results between repetitions. Recover My Files managed to complete the analysis in 79 s for the lesser pattern test, and 121 s for the many pattern test. Lastly, Foremost completed analysis in 322 and 919 s for the 10 and 45 pattern tests respectively, significantly slower than the two aforementioned file-carvers. Foremost and Recover My Files shown little variation in search time between test repetitions (<5%).

The same forensic image was then analysed using the same file-carvers on the laptop—test platform B. Fig. 5 illustrates a similar order of speed to perform pattern matching. OpenForensics, processing with both test platform B's discrete GPU and integrated IGP, took 45 s to perform analysis for both 10 and 45 pattern searching. Recover My Files completed the lesser pattern analysis in 161 s and completed the many pattern analysis in 306 s. Foremost was seen to be slowest in this test, finishing the 10 pattern analysis in 198 s and taking 826 s to complete the 45 pattern analysis.

The OpenForensics time to complete analysis on test platform B was around double that of test platform A, which collates to the theoretical read speed limits that the storage devices can read data. This indicates that the laptop may have been able to analyse much faster storage devices with the employed pattern matching approach.

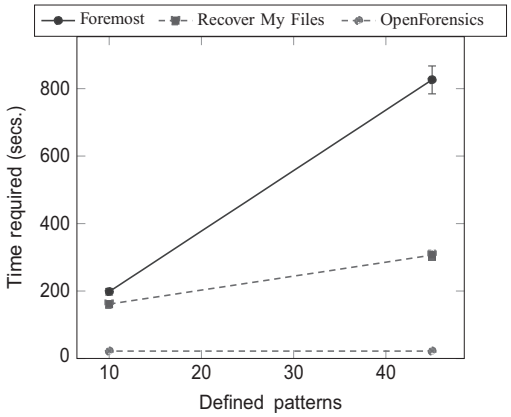
Analysis of a physical storage device

The second trial conducted involved searching an entire physical drive. Although both Foremost and OpenForensics were both able



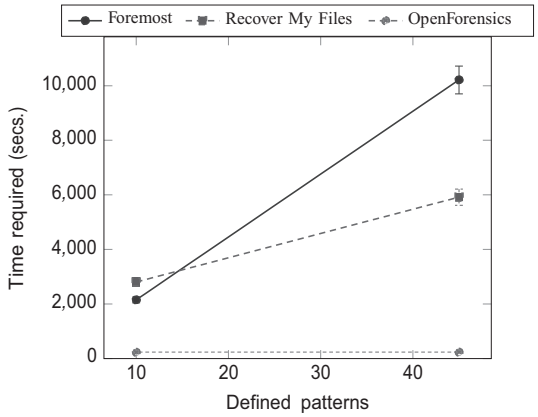
Defined patterns	Foremost	Recover My Files	OpenForensics
10	322	79	22
45	919	121	22

Fig. 4. Test platform A: Mean time taken to conduct pattern matching on raw forensic dd file (secs.) with 95% confidence intervals.



Defined patterns	Foremost	Recover My Files	OpenForensics
10	198	161	45
45	826	306	45

Fig. 5. Test platform B: Mean time taken to conduct pattern matching on raw forensic dd file (secs.) with 95% confidence intervals.



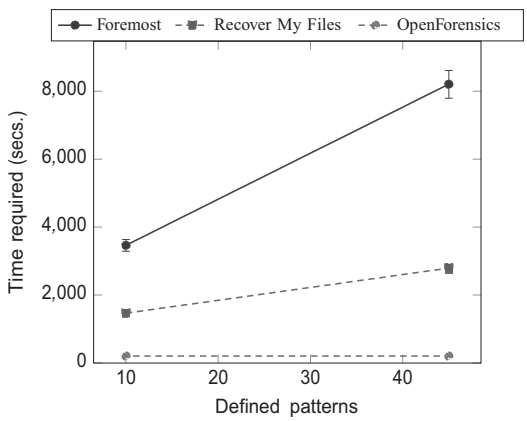
Defined patterns	Foremost	Recover My Files	OpenForensics
10	2148	2805	226
45	10211	5916	231

Fig. 7. Test platform B: Mean time taken to conduct pattern matching on physical storage device (secs.) with 95% confidence intervals.

to target the full physical drive, Recover My Files was only able to target visible partitions. On both test platforms, this limitation signifies that Recover My Files is only analysing the size of each test platform's largest partition—194.3 GiB on test platform A and 193.8 GiB on test platform B. OpenForensics and Foremost, on the contrary, analyse the full physical drive size—256 GiB on both test platform A and B. The following charts may favour Recover My Files due to this limitation, as no adjustments were made to the times to account for the size difference between partition and drive size.

Fig. 6 presents the time required to perform pattern analysis on storage device 2 of test platform A. Similarly to the previous forensic image tests, OpenForensics manages to perform analysis the quickest and shows no variation in time when searching for 10 or 45 patterns. Interestingly, however, Recover My Files shows greater deterioration from its lesser and many pattern test than the previous forensic image analysis. Foremost analyses data the slowest in this test, however, maintains a similar performance pattern between its 10 and 45 pattern searches.

Analysing storage device 2 on test platform B (Fig. 7) shows similar results, with the exception of Recover My Files, which performed slower in its few pattern test on the large partition than Foremost did in a 10 pattern test on the full drive. Unfortunately, as Recover My Files is a closed-source commercial application, further analysis of this outcome was difficult. OpenForensics managed times of 226 and 231 s to search for 10 and 45 patterns on test platform B.



Defined patterns	Foremost	Recover My Files	OpenForensics
10	3460	1464	203
45	8203	2789	203

Fig. 6. Test platform A: Mean time taken to conduct pattern matching on physical storage device (secs.) with 95% confidence intervals.

Processing speed analysis

In the proceeding section, pattern matching was completed against a 20 GiB forensic image and a physical drive. From the results, it was clear that OpenForensics was fastest to conduct pattern matching. This section aims to analyse the speed that pattern matching was done to establish the rate of analysis, and whether the fastest pattern matching method — OpenForensics — achieves the maximum possible throughput from the storage devices tested.

To begin, Table 3 shows the measured maximum sequential read rate of the storage devices tested in *Mebibytes per second (MiB/s)*. These measurements were taken from measuring sequential read performance of each storage device with the Microsoft Diskspd utility (Microsoft) using a single thread and a queue depth of 32. We assume that the maximum recorded sequential read speed of the storage device is the maximum rate that analysis can be done.

Figs. 8 and 9 shows the rate of analysis for all three file-carvers tested. The rate of analysis was calculated from the size of data analysed divided by the time taken to complete analysis. The graph provides a 10% positive deviation across tests to reflect the windowed searching technique configured by OpenForensics and Foremost to conduct pattern matching. The windowed technique employed by the file-carvers in these tests overlaps 10 MiB in every 100 MiB of data analysed to check for partial matches at the end of segments.

Due to the limitations of specifying patterns with Recover My Files, the results cannot be fairly compared to the specific pattern conditions of Foremost or OpenForensics. However, test results can serve as an indicate the possible performance achievable with

Table 3
Test platform storage device sequential read performance measured by Microsoft DiskSpd utility.

Test Platform	A	B
Storage Device 1	943 MiB/s	521 MiB/s
Storage Device 2	1378 MiB/s	1562 MiB/s

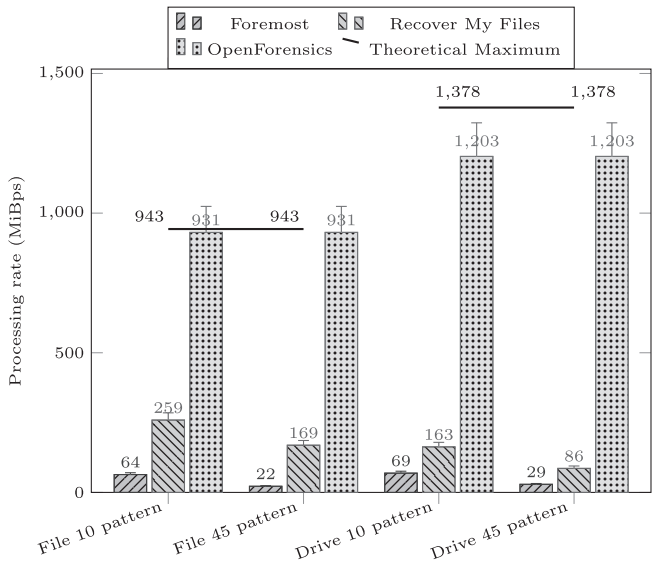


Fig. 8. Test platform A file carver rate of analysis with 90% confidence intervals.

Recover My Files when searching for fewer and greater patterns. OpenForensics and Foremost have been tested with identical search parameters, therefore, direct comparisons can be made between the performance of both file-carvers.

In all test cases, OpenForensics is able to achieve the greatest rate of analysis, achieving a processing rate close to the sequential read performance of most storage devices tested. OpenForensics used 98.7% of the measured storage device sequential read performance at best, and 85.4% on average across all tests from the two test platforms. The commercial tool trialled – Recover My Files – managed to utilise 27.4% of the storage device performance at best, and 13.6% when averaged across all tests. In this analysis, Foremost appears to take least advantage of the available storage device performance. On average, Foremost uses 6.8% of the potential read performance of storage devices. Foremost's best result – on test platform B conducting pattern matching for 10 patterns – utilises 19.8% of the sequential read performance.

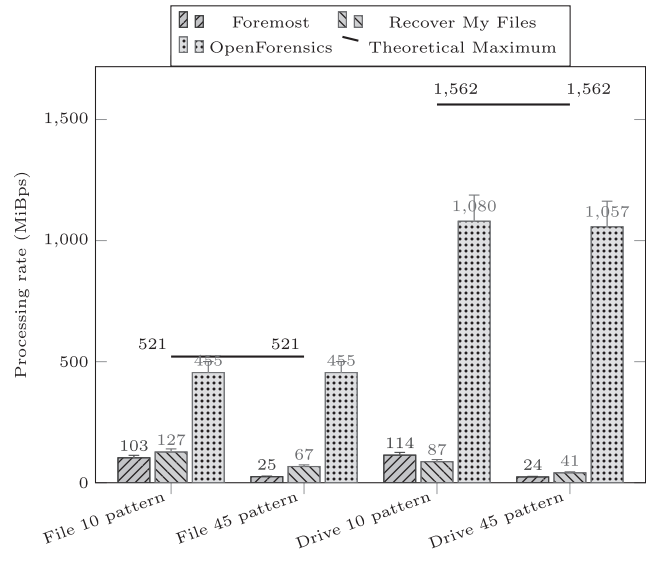


Fig. 9. Test platform B file carver rate of analysis with 90% confidence intervals.

The measured performance of OpenForensics demonstrate the advantages of employing modern pattern matching techniques and recent technological developments to the area of DF analysis methods.

OpenForensics tests on platform A, which utilises a single Nvidia Titan XP graphics card to accelerate processing, were seen to provide the highest throughput on the storage device tests. OpenForensics results from test platform A proved to better utilise the measured sequential read speed than test platform B, which approached processing using a combination of the laptop's discrete GPU and IGP. However, whilst test platform A only adopted a single GPU to conduct analysis, the platform was able to adopt more levels of concurrency than what was applied on test platform B (12 vs. 8 concurrent threads). Digressing, the speed achieved by test platform B presents compelling evidence that physical analysis could be performed quickly on a mobile platform utilising the proposed processing technique.

We hypothesised that storage device transfer speeds would limit the possible rate of analysis when analysing the forensic image from storage device 1 of each test platform. This hypothesis evidenced to be true. OpenForensics was seen to be limited by the maximum recorded sequential read speed of the storage device. It was also noted that performance derived from Foremost and Recover My Files were not limited by the storage devices.

We also hypothesised that analysing the storage device 2 of each test platform would highlight limitations of current string searching approaches adopted in DF. This hypothesis was also evidenced to be true. Analysing the performance achieved by Foremost and Recover My Files, it was evident that performance remained relatively similar to the forensic image tests. Contrary to this finding, OpenForensics was demonstrated to perform faster on the NVMe storage devices, utilising a significant portion of the storage device's maximum sequential read speed.

Conclusion

This paper has presented OpenForensics—an open-source implementation of an asynchronous GPU PFAC solution for pattern matching of data in a DF context. Results show that OpenForensics is able to perform pattern analysis close to the maximum theoretical sequential read speed performance of storage devices in most tests, providing substantial performance advantages over the searching techniques employed by Foremost and Recover My Files.

The paper further highlights the inadequacies of current DF tools in being able to perform analysis on modern consumer hardware. It is hoped that the methods and technique shared by this research serve as a framework to improve upon, or create, the next generation of DF tools used by professionals internationally.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research. The authors would also like to thank Abertay University's R-LINCS initiative for the funding of a compute server that was used in the development of this processing model.

References

Aho, A.V., Corasick, M.J., 1975. Efficient string matching : an aid to bibliographic search. Commun. ACM 18 (6), 333–340.

Ayers, D., 2009. A second generation computer forensic analysis system. Digit. Invest. 6, S34–S42. https://doi.org/10.1016/j.diin.2009.06.013.

E. Bayne, OpenForensics. URL https://github.com/ethanbayne/OpenForensics.

Beebe, N., 2009. Digital forensic research: the good, the bad and the unaddressed. Adv. Digit. Forensics V 17–36. https://doi.org/10.1007/978-3-642-04155-6_2.

Bellekens, X.J.A., Tachtatzis, C., Atkinson, R.C., Renfrew, C., Kirkham, T., 2014. GLoP: enabling massively parallel incident response through GPU log processing. In: Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14, pp. 295–301. https://doi.org/10.1145/2659651.2659700.

Boyer, R.S., Moore, J.S., 1977. A fast string searching algorithm. Commun. ACM 20 (10), 762–772.

CrystalMark, CrystalDiskMark. URL http://crystalmark.info/software/CrystalDiskMark/index-e.html.

Garfinkel, S.L., 2007. Carving contiguous and fragmented files with fast object validation. Digit. Invest. 4 (Suppl. l), 2–12. https://doi.org/10.1016/j.diin.2007.06.017. http://linkinghub.elsevier.com/retrieve/pii/S1742287607000369.

Garfinkel, S.L., 2010. Digital forensics research: the next 10 years. Digit. Invest. 7, S64–S73. https://doi.org/10.1016/j.diin.2010.05.009.

Her Majestys Inspectorate of Police, July 2015. Online and on the Edge: Real Risks in a Virtual World. Technical Report. https://www.justiceinspectorates.gov.uk/hmicfrs/wp-content/uploads/online-and-on-the-edge.pdf.

Hung, C.-L., Lin, C.-Y., Wang, H.-H., 2014. An efficient parallel-network packet pattern-matching approach using GPUs. J. Syst. Architect. 60, 431–439. https://doi.org/10.1016/j.sysarc.2014.01.007.

K. Kendall, J. Kornblum, N. Mikus, Foremost. URL http://foremost.sourceforge.net/.

Khronos Group, OpenCL. URL http://www.khronos.org/opencl/.

Lin, C.-H., Liu, C.-H., Chien, L.-S., Chang, S.-C., 2013. Accelerating pattern matching using a novel parallel algorithm on GPUs. IEEE Trans. Comput. 62 (10), 1906–1916. https://doi.org/10.1109/TC.2012.254.

Marziale, L., Richard, G.G., Roussev, V., 2007. Massive threading: using GPUs to increase the performance of digital forensics tools. Digit. Invest. 4, 73–81. https://doi.org/10.1016/j.diin.2007.06.014.

Microsoft, TechNet Diskspd Utility: A Robust Storage Testing Tool (superseding SQLIO). URL https://gallery.technet.microsoft.com/DiskSpd-a-robust-storage-6cd2f223.

Nvidia, CUDA. URL http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html.

Penrose, P., Buchanan, W.J., Macfarlane, R., 2015. Fast contraband detection in large capacity disk drives. Digit. Invest. 12 (S1), S22–S29. https://doi.org/10.1016/j.diin.2015.01.007.

Raghavan, S., 2013. Digital forensic research: current state of the art. CSI Trans. ICT 1 (1), 91–114. https://doi.org/10.1007/s40012-012-0008-7.

Richard III, G.G., Roussev, V., 2005. Scalpel: a frugal, high performance file carver. In: Proceedings of the 2005 Digital Forensics Research Workshop (DFRWS '05), pp. 1–10.

Skrbina, N., Stojanovski, T., 2012. Using parallel processing for file carving. In: Proceedings of the Nineth Conference on Informatics and Information Technology, 19-22 April, Web proceedings, pp. 175–179. ISBN 978-608-4699-01-9. http://ciit.finki.ukim.mk/data/papers/9CiiT/9CiiT-36.pdf. arXiv:1205.0103.

Zha, X., Sahni, S., 2011. Fast in-place file carving for digital forensics, lecture notes of the institute for computer sciences. Soc. Inform. Telecommun. Eng. 56, 141–158. https://doi.org/10.1007/978-3-642-23602-0_13.