OpenCL Acceleration of Digital Forensic Methods

Ethan Bayne, Robert Ian Ferguson, John Isaacs

School of Science, Engineering and Technology, Abertay University, Kydd Building, Bell Street, Dundee, DD1 1HG

Abstract

String searching plays a critical role in the digital forensic (DF) methods used to analyse digital corpora for evidence to assist an investigation. As the storage capacity on modern devices continues to grow, the current generation of DF tools struggle to efficiently perform analyses on the growing sizes of forensic data. This struggle presents a requirement for further research to be conducted on providing low-cost and effective techniques of improving file-carving performance.

One strategy to mitigate this is to employ general purpose computing on graphics processing units (GPGPU) to handle the processing of data in a parallel fashion. Currently, only some of the research on GPGPU processing has been transferred to the field of DF, mostly based around Compute Unified Device Architecture (CUDA)—a closed-source GPGPU framework with limited hardware compatibility. This project researches an alternative open-source GPGPU framework, Open Computing Language (OpenCL), to accelerate file carving with a wider compatibility across devices. The proposed OpenCL solution demonstrates processing improvements over traditional single-threaded central processing unit (CPU) processing models that are currently used to perform string searching in file carving and other DF techniques.

Keywords: , Digital forensics, String searching, File carving, GPGPU, GPU, IGP, Parallel programming

1. Introduction

The growth of storage capacities available on modern computers is seen as one of the largest concerns within the digital forensic (DF) community, as the current generation of tools are failing to efficiently process increasing sizes of digital corpora. This is primarily due to the amount of processing power required to inspect each segment of data within the forensic image. As a result, the time needed to investigate cases has increased significantly and, as an inevitable side effect, caused large backlogs of forensic investigations. This issue highlights the urgent need for more powerful methods of processing digital evidence.

String searching is a vital underlying task of many techniques used to ascertain evidence in DF investigations; however, performing string searching is computationally intensive. While suggestions have been made over the past few years to improve processing speeds through the use of distributed computing techniques (Ayers, 2009), this study presents an alternative processing solution utilising parallel processing and graphics processing units (GPUs) that may prove more efficient and cost-effective for DF professionals. Results ascertained from this study show significant performance gains from the proposed parallel processing solution when compared to a traditional central processing unit (CPU) processing model.

Modern GPUs, such as the Nvidia GeForce 670, contain 1,344 independent general purpose processing cores, which are able to compute large amounts of data in a short time due to an efficient parallel processing design. When considering the computational power of these devices, GPUs prove to be very affordable processing platforms when employed in scientific investigations. At the time of writing, hardware vendors AMD and Intel have both released desktop- and laptop-based CPUs with powerful integrated graphics processors (IGPs) (Intel Corporation, 2013; Advanced Micro Devices, 2014). These releases have opened up the possibility of efficient parallel processing without the need for a discrete GPU. Similarly, CPUs paired with powerful IGPs can be witnessed on devices other than desktop machines—modern tablets and phones also show similar technological developments. Consequently, in order to utilise the advantages of these technologies, a compatible general purpose computing on graphics processing unit (GPGPU) framework, such as Open Computing Language (OpenCL), must be explored.

Significant research has been conducted on the benefits of utilising GPGPU programming to assist in calculating highly demanding processing tasks; yet, only some advances made in this research have been transferred to the field of DF. Efforts have been primarily focused on utilising Complete Unified Device Architecture (CUDA)—a closed-source GPGPU programming framework offered by Nvidia exclusively for use on their line of GPUs (Marziale et al., 2007; Skrbina & Stojanovski, 2012; Collange & Dan-

Email addresses: `e.bayne@abertay.ac.uk` (Ethan Bayne), `i.ferguson@abertay.ac.uk` (Robert Ian Ferguson), `j.isaacs@abertay.ac.uk` (John Isaacs)

dass, 2009; Zha & Sahni, 2011a). The largest setback of the existing developments is the incompatibility of CUDA applications with the GPU technologies offered by other hardware vendors, including the new breed of IGPs found on CPUs offered by Intel and AMD.

This paper investigates whether OpenCL technologies could assist DF investigation. The aim of the experiments undertaken in this paper is to measure the effectiveness of offloading processor-intensive tasks to the GPU using OpenCL and CUDA processing frameworks, while comparing the resulting performance with single-threaded CPU processing. Second, this paper analyses the potential processing obstacles of the proposed solution through experimentation with different storage technologies and various mid- to high-end system specifications. The authors of this paper are unaware of any existing research that proposes the use of OpenCL GPGPU processing for string searching in the context of DF investigations, making this implementation unique in the field.

2. Background and Related Work

Existing research using CUDA GPGPU processing has demonstrated promising results, suggesting that an OpenCL implementation may produce a similar outcome. Research that focused on measuring the potential differences between CUDA and OpenCL (Fang et al., 2011; Karimi et al., 2010) demonstrates that CUDA shows a slight performance advantage over its open-source counterpart. This means that with OpenCL, a minor sacrifice in performance currently exists to obtain a wider compatibility across devices from various vendors. Findings from these studies also indicate that much of the existing research on CUDA may be relevant and transferable in facilitating future OpenCL research due to the similar approaches utilised by each technology to interface with the GPU.

Skrbina and Stojanovski (2012) present a theoretical insight into the planning and processes involved in implementing a GPGPU solution to file carving. The authors explore how CUDA could be utilised within the context of DF investigations, examining the differences between string matching and pattern matching algorithms for use in GPGPU applications. The paper concludes that the most appropriate algorithms for parallel applications are the Boyer-Moore (Boyer & Moore, 1977) and Aho-Corasick (Aho & Corasick, 1975) algorithms for handling single- and multi-string searches, respectively. The findings presented were nevertheless based on theoretical grounding from algorithm characteristics and they have not been proven as fact in an actual experiment.

One empirical study conducted utilising GPGPU processing in DF is presented by Marziale, Richard and Roussev (2007), who led an investigation of modifying Scalpel, an existing open-source file carving tool. The study compares the time taken to complete various searches through different sized forensic images using the unmodified and modified GPU versions of Scalpel. Results show significant improvement with the modified version, and Marziale, Richard and Roussev concluded that incorporating GPGPU processing is a viable option for significantly increasing processing performance in existing DF tools. At the time of the research, however, the CUDA framework was still in beta. The authors acknowledge that the beta release possessed a number of bugs and the beta compiler did not fully optimise the code; these factors limit the proposed solution's potential achievable performance.

When incorporating a fast multi-pattern matching algorithm, opposing research from Zha and Sahni (2011a) states that the performance gain achievable from file carving is limited by the time required to read data from the disk rather than the time required to search for headers and footers. Zha and Sahni conducted experiments through modifying Scalpel as well as integrating a series of Boyer-Moore and Aho-Corasick algorithms for handling string matching. The authors' experiments indicate that multi-threaded acceleration using a dual-core CPU did not improve the required processing time, concluding with an arguable assumption there are no advantages of using other accelerators such as GPUs, despite conducting no actual experiments with GPGPU processing. Later research from the same authors (Zha & Sahni, 2011b, 2013) shows that incorporating similar algorithmic techniques using GPGPU processing produced notable improvements over single-threaded CPU approaches, besting multithreaded CPU processing in some scenarios. Results of these later studies form the argument that the authors' previous research on processing techniques to improve file carving had not been thoroughly investigated.

In a thesis, Mohan (2010) adopted another fascinating method of utilising GPGPU processing in file identification, utilising an MD6 file-hashing method on a CUDA GPGPU framework to identify similar files, both individually and those contained within archives. His results demonstrate a significant performance increase over traditional CPU processing, which led us to conclude that the parallel nature of GPUs is well suited for the large-scale processing of MD6 file hashing. Despite the author's findings, this method of discovery requires a search list of known file hash signatures, limiting its usefulness when performing exploratory examinations on unknown incriminating files.

Collange et al. (2009) adopted a similar approach, proposing the use of GPUs in file carving to look for file identifiers. They use a CUDA GPGPU implementation to calculate and compare hashes of data to distinguish potential image file identifiers located on hard disk drives (HDD). The authors make a strong statement that, with the computational power of the hardware, GPUs make an ideal platform on which to perform parallel hash calculations, potentially delivering a powerful and usable file identification technique for DF investigation. Although Collange et al. eliminate the requirement of knowing complete file hashes, the proposed approach still requires and is dependent on the CPU to verify the matches found as valid image files; this potentially slows the overall performance when faced with forensic images containing significant amounts of data.

3. Overview

In this section, we introduce the themes used in this study, beginning with a brief introduction to OpenCL before discussing the core differences—in both programming methods and architecture—between CPU and GPU.

3.1. OpenCL and GPU Architecture

OpenCL is a heterogeneous programming framework managed by the non-profit technology consortium, Khronos Group (2013). The framework is widely compatible across a variety of devices offered by various vendors, including leading companies such as Intel, AMD, ATI and Nvidia. OpenCL allows developers to create applications that can perform diverse levels of parallelism across single or multiple processing devices, such as the typical CPUs and GPUs found on computers and mobile platforms.

CPU processing is a traditional programming concept that usually operates by executing commands in a serial or limited parallel fashion on a system's CPU. CPUs perform parallel operations through a task parallelism model, which acts by allocating multiple tasks to numerous threads with each task containing an array of different complex instructions. Each thread executed on a CPU is required to be individually programmed and managed explicitly by the program.

In contrast, GPGPU processing employs the collaboration of a system's GPU to perform calculations on a massively parallel basis; this is accomplished through utilising a Single Instruction Multiple Data (SIMD) model to achieve high levels of efficiency. The model differs from the task parallelism model by allocating thousands of simplistic threads to numerous multiprocessors, with all threads managed at the hardware level. To accomplish this, additional code in the form of a kernel is required. The purpose of the kernel is to provide instructions to direct the GPU in processing the data. The instructions that form kernels tend to be far more simplistic in functionality, generally offering limited logical and mathematical functions. When compared to the complex functions that can be handled by the CPU, GPGPU kernel instructions can be somewhat limited. This can be forgiven in scientific applications due to the GPGPU processing model being capable of efficiently executing repetitive instructions on large amounts of data.

Similarly, when comparing the hardware architecture of both CPUs and GPUs in Fig. 1, it can be seen that the architectures have been designed around optimising their processing models. The current conventional CPUs are, in all regards, designed to efficiently handle the diversity of tasks encountered in everyday computing. Modern CPUs typically range between two arithmetic logic unit (ALU) cores and eight ALU cores from higher-end offerings. These cores possess a range of arithmetic instruction functions, which can process a variety of tasks with ease. CPUs are paired with a large cache for caching data from the main system memory. Ultimately, CPU architecture is optimised for the perpendicular processing of complex tasks with minimal latency; however, the architecture is not well equipped to handle large sets of superficial calculations.

GPUs have, in comparison, thousands of ALU cores. Groups of these cores form multiprocessor units capable of performing thousands of highly intensive calculations simultaneously. GPUs are superior to CPUs in handling massively parallel computing tasks due to the sheer volume of ALU cores; yet, unlike CPUs, GPUs currently have limited algorithmic instruction functions, which hamper their ability to handle diverse tasks. GPUs typically have smaller caches, which hinder the ability to handle complex datasets. GPUs have a large dedicated memory used to hold data for processing, and this memory is typically optimised to reduce latency from reading and writing to the processing units. This architecture allows GPUs to be more suited to handle large quantities of simple yet intensive calculative workloads, such as its primary function of producing graphics and assisting in scientific calculations of processor-intensive datasets.

3.2. File Carving

File carving is employed as an essential method of analysing forensic images for content when a file system's directory has been corrupted or destroyed. Recognised as "a science and an art unto itself" (Altheide & Carvey, 2011), the technique is used to analyse forensic images by attempting to recognise file content from otherwise unstructured streams of data. Typically, this is achieved by comparing the stream of data with a database of known file headers and footers (also referred to as magic numbers). File carving can often reconstruct a copy of discovered files by identifying and assembling data between headers and footers; however, dangers, such as file fragmentation, can render the recovered file unusable or incomplete. Unusable and incomplete files are often

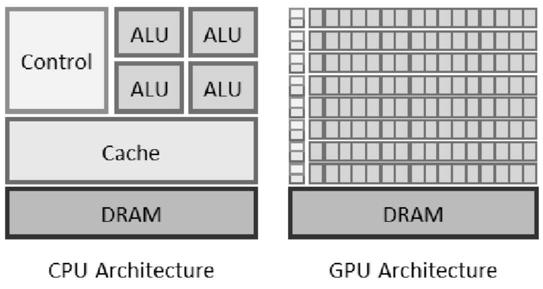

Figure 1: CPU and GPU architecture

referred to as false positives in file carving results, as investigators cannot interpret them.

Good file carving practices employ various techniques to improve performance. Reading from storage devices tends to be the most time-consuming task and, as a forensic image could be terabytes in size, file carvers must minimise the number of reads of the original image in order to optimise performance. One method used to mitigate this is an optimised algorithm to aid in the search for and identification of file headers and footers in the file carving process. File carvers can also use techniques to validate files carved using what is known about the structure of a file, which improves the accuracy of results and minimises the number of false positive results presented to the investigator.

Fragmentation issues within file carving have been a prominent area of research in recent years. The Digital Forensics Research Workshop (DFRWS) File Carving Challenge (Carrier et al., 2006) asked researchers to produce algorithms to detect fragmented files with low false positive rates. In response, Garfinkel (2007) demonstrates that after conducting extensive fragmentation research on more than 300 used hard drives, file fragmentation itself is relatively rare. Interestingly, Garfinkel also noted that fragmentation rates typically decrease as storage sizes available on devices increase. On this basis, it could be argued whether developing fragmentation detection algorithms would benefit investigations at the expense of adding additional processing burdens.

4. Experimental Study

This study investigates OpenCL GPGPU technologies, benchmarking the technology's performance in processing forensic images by using two common DF techniques on a range of test platforms. Results from OpenCL were then compared to the performance gained with comparable solutions utilising CUDA and CPU processing to measure and conclude whether an OpenCL GPGPU framework could provide a reliable foundation to analyse digital evidence and decrease the time required to process forensic images without affecting accuracy.

4.1. Our Approach

To analyse the performance of OpenCL file carving, a program was developed using Microsoft Visual C# alongside the Cudafy .NET (Hybrid DSP, 2013) framework to manage GPU operations. Cudafy .NET provides a comprehensive set of libraries and methods to allow C# applications to interface with GPUs. The framework also hosts emulation features, which simulate the processing of GPU kernels on the CPU, providing excellent simulation and debugging functionality. The most significant advantage of using Cudafy .NET in this experimental study is kernel creation, as it comes with the ability to translate a kernel written in C# to both CUDA and OpenCL languages without the need to write separate kernels for the solution. The authors of this study reviewed the kernels generated by Cudafy .NET for efficiency as well as to ensure that both the OpenCL and CUDA kernels followed the same underlying code when carrying out instructions.

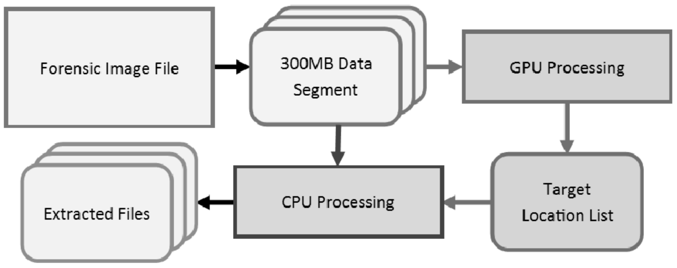

Figure 2: CPU and GPU processing methods

The solution's processing approach to the problem is outlined in Fig. 2. Both CPU and GPU approaches follow a similar set of motions, requiring only one single read of the forensic image.

The CPU process starts by reading the data from the forensic image in 300-MB segments; the CPU then systematically processes the data by searching for a predefined list of targets using a string search algorithm. If file carving is employed, the matches found are validated against what is known about the file structure before the file is reconstructed from the data segment. Once all the data segments are processed, the results are presented back to the user.

The GPU implementations differ by synchronously processing the 300-MB data segments on the GPU. In this scenario, the GPU analyses data segments and produces a list of target locations that are sent back to the CPU. The CPU is then able to perform any additional file carving tasks with the locations while the GPU processes the next data segment. When all segments are processed, the results are presented back to the user.

Two string search algorithms were used to process the raw data obtained from the forensic images. The GPGPU algorithm was simply a linear search of bytes in a data stream, looking for the first byte of each specified target in parallel; this method relies heavily on the GPU architecture to accelerate the search for multiple targets within the data stream. Upon discovering the first byte of a target, the GPU would then verify the sequential bytes of the stream against the target to validate a match.

For CPU processing, a modified multiple-string Boyer-Moore search algorithm was chosen specifically for the experimental study to mimic Foremost, a widely used file carver available for Linux. As the proposed solution was built for Windows, a direct comparison between the proposed solution and Foremost would be unfair due to operating system differences. Therefore, the algorithm used for comparison was founded through Foremost's source code, allowing the study to compare the performance achieved by the proposed GPU implementation in the CPU algorithm, as employed by Foremost. The Boyer-Moore algorithm operates by searching a stream of data for the last byte of the target. When the algorithm discovers the last byte, the rest of the target is validated. Searching through the data stream is accelerated by the creation of a skip table. The skip table acts as a reference on the range to find the next possible match, depending on the value read; this significantly reduces the time required to search through the data stream for target matches.

4.2. Experimental Evaluation

Testing took place within two distinct case studies: case study A recorded the time required to perform string analysis to identify a series of seven search targets; whilst case study B measured the algorithm's performance in the context of a realistic file carving problem by analysing the time required to carve JPEG files with a set of three defined headers and footers. File carving in case study B differed from case study A in performing validations and reconstructions of JPEG files from the data when search targets were identified.

Each target specified in the search parameters adds an additional calculative workload to the processor. Case study B was designed to search for fewer targets as well as to analyse how both CPU and GPU processing times would be affected by a reduction in specified targets.

Two forensic images were used in this experimental study: a 20-GB storage device containing large amounts of different media files and a 150-GB Windows 7 image, which contained a copious amount of different file types. The latter 150-GB image test was not conducted on the solid state drive (SSD) devices due to storage constraints.

Each case study analysed the performance gain achievable by examining the forensic images on SSD and HDD storage devices, respectively. Each test analysing the forensic image was performed twenty times, and analysis was conducted on the average mean derived from the twenty times produced. A reboot was performed between each run to account for any caching effects from skewed results.

As DF is a scientific field, results are required to be accurate and reproducible. To facilitate the compliance of this goal, Foremost was used to identify the content of the forensic images before testing was carried out. As Foremost is a recognised and established tool within DF, the results derived from the tool are assumed accurate, serving as a benchmark for the accuracy of the proposed solution.

To measure performance gain; a range of mid- to high-end computers of varying performance levels were used to benchmark the algorithms of each case study. The test platforms were all equipped with SSD and HDD storage devices to measure algorithm performance with different storage technologies. SSD storage devices are known to achieve faster data throughput, as they can read and write data with lower access time and less latency than HDD drives. However, it was assumed that there would be a correlation between the type of storage device used and performance gain achieved from GPU acceleration, as, in both case studies, the forensic image is read once in a single pass to minimise the impact of storage device activity.

Table 1 shows system specifications of the computers that served as test platforms along with their allocated platform identifiers. Where possible, the hardware varied to identify potential setbacks in processing times. Test platforms A and B are desktop computers. Test platforms C and D are the same laptop, tested first with the Intel Haswell processor's IGP and second with the laptop's discrete Nvidia GPU. The Haswell processor's IGP shares video memory with the main system memory. While it is also slower than the high-speed memory found on the dedicated GPUs of test platforms A and B, a slight benefit was assumed, as there would be no physical transfer of data from system memory to discrete GPU memory.

4.3. Case Study A - String Search

Results of the low-level string matching stage using SSD drives are shown in Table 2. From these results, significant performance gains can be observed across all tests with GPGPU acceleration from both OpenCL and CUDA frameworks. OpenCL processing shows average performance gains of 89.06% when compared to the results achieved from the same test done by CPU processing, which is only marginally slower than the 90.10% average performance gain achieved by CUDA processing.

Similarly, the second phase, which tested algorithm performance on HDD storage drives with the same search parameters, as shown in Table 3, yielded significant performance gains, with averages of 77.67% for OpenCL and 79.45% for CUDA. These figures show a conclusive drop in performance gain between CPU and GPU technologies when compared to the results achieved from SSD drives,

Test Platform	**A**	**B**	**C**	**D**
Type	Desktop	Desktop	Laptop	Laptop
Operating System	Windows 8.1 Pro	Windows 8.1 Pro	Windows 8.1 Pro	Windows 8.1 Pro
Processor	Intel Core i5-2500k	Intel Core i3-2120	Intel Core i7-4700MQ	Intel Core i7-4700MQ
Processor Specification	Quad Core @ 4.2GHz	Dual Core @ 3.3GHz	Quad Core @ 2.4GHz	Quad Core @ 2.4GHz
Memory	16GB DDR3 1600MHz	8GB DDR3 1600MHz	8GB DDR3 1600MHz	8GB DDR3 1600MHz
GPU	Nvidia 670 GTX (2GB GDDR5)	Nvidia 580 GTX (1.5GB GDDR5)	Nvidia 740m (2GB DDR3)	Intel HD 4600 (Haswell)
GPU Specification	1344 Core @ 915MHz	512 Core @ 612MHz	384 Core @ 810MHz	20 Core @ 1200MHz
SSD Storage	120GB SATA3 SSD	120GB SATA3 SSD	120GB mSATA SSD	120GB mSATA SSD
HDD Storage	2TB SATA3 7200RPM HDD	1TB SATA2 5400RPM HDD	750GB SATA2 5400RPM HDD	750GB SATA2 5400RPM HDD

Table 1: Test bed specifications

which was unexpected due to both CPU and GPU approaches requiring a single read of the storage device data. Therefore, it is affected by the same time required to read the forensic image. However, it was noticeable that test platform A performed significantly better in this test than the other platforms.

4.4. Case Study B - File Carving

The results for carving JPEG images from forensic images obtained from SSD tests 1 and 2 are presented in Table 4. The results across the tests are promising, with an average performance gain of 63.94% in OpenCL tests and with CUDA processing surpassing the OpenCL results by only a few seconds; in some cases, CUDA achieved an average performance gain of 68.01%.

Table 5 shows the test results achieved by using the test platform's HDD storage to read the forensic image. The OpenCL figures of test 2 show the test platforms produce a slower average performance gain of 47.51%, around 16% slower than comparative results derived from SSD tests. Likewise, this drop in performance is observed in test platforms utilising CUDA, which show average gains of 51.42%, around 17% slower than the performance achieved in SSD tests.

Comparing both string matching and file carving case studies, the performance gap between SSD and HDD tests is far larger in file carving than in string searching. Analysing further, it can be seen that CPU tests require significantly more time to complete string analysis than file carving; though it can be observed that GPU processing results have less time divergence between string analysis and file carving. Furthermore, OpenCL and CUDA processing tests demonstrate a drop in performance between 11% and 17% when conducting tests on HDD storage devices.

4.5. Discussion

It is evident from the experiments that the performance of both string matching and file carving with parallelised GPU processing on sequential data is far superior to that achieved with a single-threaded Boyer-Moore algorithm approach on CPUs.

With the increase in specified targets in case study A, we can observe that both GPGPU technologies produced higher performance gains than what was observed in case study B. With a greater number of search targets, it was observed that CPU performance decreased significantly, mainly due to the single-threaded processing limitation and reduced efficiency of the Boyer-Moore algorithm when handling a multitude of search targets. Consequentially, from this comparison we can also measure how effectively GPGPU processing handles the increased number of search targets, showing that GPGPU processing results have less time divergence between string analysis and file carving; this shows superior scalability to that offered by CPU implementation.

	Test 1: 20GB Storage Device		
Test Platform	**CPU**	**OpenCL**	**CUDA**
A	722.23	102.30	92.30
B	846.81	71.18	62.00
C	824.71	84.44	79.20
D	814.65	89.08	

Table 2: Case study A - SSD platform time results (secs)

	Test 1: 20GB Storage Device		
Test Platform	**CPU**	**OpenCL**	**CUDA**
A	752.39	144.24	135.04
B	1031.99	256.85	247.21
C	1060.51	289.59	279.51
D	1075.78	294.22	
	Test 2: 150GB Win7 Image		
Test Platform	**CPU**	**OpenCL**	**CUDA**
A	7385.17	1149.42	1076.13
B	10430.77	1993.14	1917.48
C	9910.25	2275.85	2187.31
D	10249.38	2280.76	

Table 3: Case study A - HDD platform time results (secs)

	Test 1: 20GB Storage Device		
Test Platform	**CPU**	**OpenCL**	**CUDA**
A	286.48	122.63	109.00
B	310.15	109.19	90.88
C	321.89	111.31	92.08
D	322.33	102.03	

Table 4: Case study B - SSD platform time results (secs)

	Test 1: 20GB Storage Device		
Test Platform	**CPU**	**OpenCL**	**CUDA**
A	330.01	166.31	152.85
B	497.74	283.24	267.04
C	520.59	306.64	288.82
D	520.87	300.20	
	Test 2: 150GB Win7 Image		
Test Platform	**CPU**	**OpenCL**	**CUDA**
A	3007.08	1314.86	1208.48
B	4367.35	2201.64	2060.89
C	4628.52	2381.65	2252.84
D	4624.98	2334.58	

Table 5: Case study B - HDD platform time results (secs)

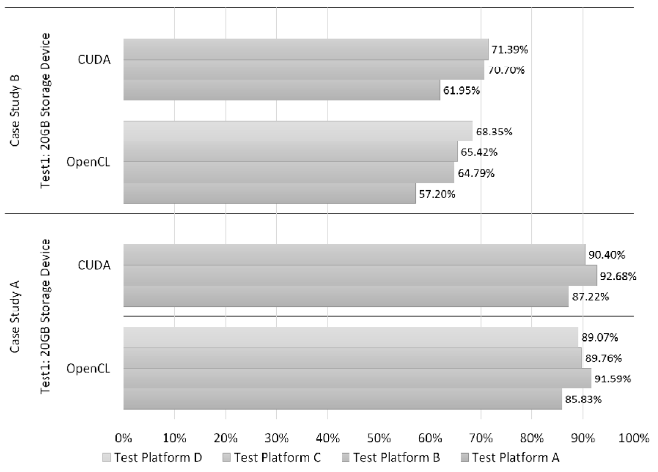

Figure 3: Percentile performance gains in SSD tests

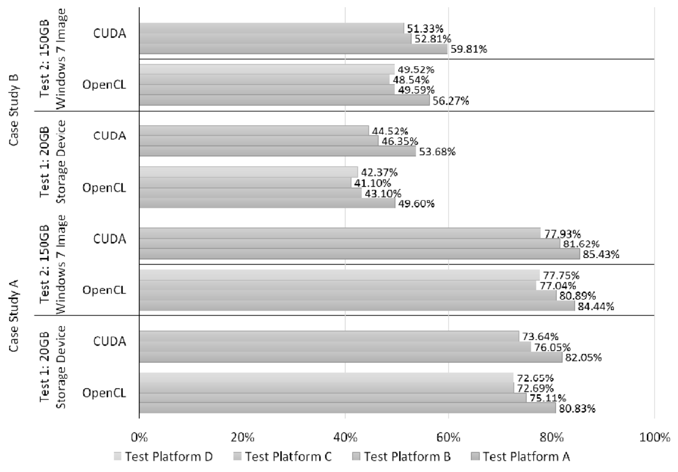

Figure 4: Percentile performance gains in HDD tests

Storage devices had an overall impact on performance in both case studies. This can be mostly deduced by two significant performance-affecting factors in our experiments: the type of storage technology—SSD or HDD—and the interface to which the drive is connected. Experiment results show a notable improvement in processing times achieved with test platform A's HDD on a SATA3 interface compared to other test platform HDDs using SATA2, as data transfer speeds on a SATA3 interface are two times faster than its predecessor's speeds. While the SATA3 interface does substantially improve the performance of HDDs, it still lags behind SSD storage drive technology in our tests.

When inspecting the performance gains delivered by OpenCL and CUDA in tests (Figs. 3 and 4), no significant variations exist; although, it is acknowledged that CUDA processing does, overall, perform marginally faster in tests, which supports the findings of the GPGPU performance research from Fang, Varbanescu and Sips (2011), and Karimi, Dickson and Hamze (2010). This study demonstrates that both OpenCL and CUDA technologies could significantly aid in processing the demands of DF investigations; however, tests utilising OpenCL on an IGP illustrate similar performance gains to that of discrete GPUs, which signifies it could be possible to have powerful DF investigative tools on mobile platforms with IGPs.

5. Conclusion

OpenCL has demonstrated to be a fast and reliable framework for intensive processing tasks within DF. The recorded GPGPU platform results show that the proposed OpenCL solution managed to process forensic images with a negligible loss in performance when directly compared to the widely researched CUDA alternative. With technology trends indicating continued developments of more powerful IGPs on CPUs as well as the deployment of ARM-based processors in mobile and embedded systems; the authors of this study believe that a more compatible and open-sourced model of the OpenCL platform favours the scientific requirements and development needs of DF tools and analyses.

This study also demonstrates how GPGPU technologies can speed up analyses on both desktops and laptops, respectively. The results obtained from test platforms have shown significant performance improvements throughout; however, it is recognised that storage drive performance can limit possible performance gains that can be achieved through GPU acceleration. This finding is deemed significant, as it counters the critique that forensic analysis is solely limited by storage device transfer speeds and not processing power. The results from this study illustrate that the limitation is a combination of processing power and storage device transfer speeds.

6. Future Work

The research presented in this paper has identified a number of additional research possibilities, including further trials of the Boyer-Moore and Aho-Corasick algorithms for both CPU and GPU processing. It is believed that further algorithm research is vital to achieve faster multistring file carving with the proposed GPU framework. Assessment of the benefits of utilising the proposed OpenCL framework on a multi-GPU system has also been proposed.

An application demonstrating the advantages of OpenCL in DF investigations is being prepared for public release and will be available at http://www.openforensics.com.

References

Advanced Micro Devices (2014). AMDs most advanced APU ever. URL: `http://www.amd.com/us/products/desktop/processors/a-series/Pages/nextgenapu.aspx`.

Aho, A. V., & Corasick, M. J. (1975). Efficient String Matching : An Aid to Bibliographic Search. *Communications of the ACM*, *18*, 333–340.

Altheide, C., & Carvey, H. (2011). *Digital Forensics with Open Source Tools*. (1st ed.). Waltham: Syngress.

Ayers, D. (2009). A second generation computer forensic analysis system. *Digital Investigation*, *6*, S34–S42. URL: http://linkinghub.elsevier.com/retrieve/pii/S1742287609000371. doi:10.1016/j.diin.2009.06.013.

Boyer, R. S., & Moore, J. S. (1977). A Fast String Searching Algorithm. *Communications of the ACM*, *20*, 762–772.

Carrier, B., Casey, E., & Venema, W. (2006). DFRWS 2006 File Carving Challenge. URL: http://www.dfrws.org/2006/challenge/.

Collange, S., & Dandass, Y. (2009). Using graphics processors for parallelizing hash-based data carving. *System Sciences*, (pp. 1–10). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4755790.

Fang, J., Varbanescu, A. L., & Sips, H. (2011). A Comprehensive Performance Comparison of CUDA and OpenCL. *International Conference on Parallel Processing*, (pp. 216–225). URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6047190. doi:10.1109/ICPP.2011.45.

Garfinkel, S. L. (2007). Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, *4*, 2–12. URL: http://linkinghub.elsevier.com/retrieve/pii/S1742287607000369. doi:10.1016/j.diin.2007.06.017.

Hybrid DSP (2013). CUDAfy .NET. URL: http://www.hybriddsp.com/Products/CUDAfyNET.aspx.

Intel Corporation (2013). 4th Generation Intel Core Processors. URL: http://www.intel.com/content/www/us/en/processors/core/4th-gen-core-processor-family.html.

Karimi, K., Dickson, N., & Hamze, F. (2010). A performance comparison of CUDA and OpenCL. *ArXiv e-prints*, *1005.2581*. URL: http://arxiv.org/abs/1005.2581. arXiv:1005.2581.

Khronos Group (2013). OpenCL. URL: http://www.khronos.org/opencl/.

Marziale, L., Richard, G. G., & Roussev, V. (2007). Massive threading: Using GPUs to increase the performance of digital forensics tools. *Digital Investigation*, *4*, 73--81. URL: http://linkinghub.elsevier.com/retrieve/pii/S1742287607000436. doi:10.1016/j.diin.2007.06.014.

Mohan, D. (2010). *Faster file matching using GPGPUs*. Ph.D. thesis University of Delaware. URL: http://dspace.udel.edu:8080/dspace/handle/19716/5905.

Nvidia (2013). CUDA Developer Zone. URL: https://developer.nvidia.com/category/zone/cuda-zone.

Skrbina, N., & Stojanovski, T. (2012). Using parallel processing for file carving. *ArXiv e-prints*, *1205.0103*. URL: http://adsabs.harvard.edu/abs/2012arXiv1205.0103S. arXiv:1205.0103.

Zha, X., & Sahni, S. (2011a). Fast in-Place File Carving for Digital Forensics. *Forensics in Telecommunications, Information, and Multimedia*, (pp. 141--158).

Zha, X., & Sahni, S. (2011b). Multipattern String Matching On A GPU. *IEEE Symposium on Computers and Communications*, (pp. 277--282).

Zha, X., & Sahni, S. (2013). GPU-to-GPU and Host-to-Host Multipattern String Matching on a GPU. *IEEE Transactions on Computers*, *62*, 1156--1169.