Brute force de-shredding algorithm using the GPU

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Simon Naudé
Department of Computer Science
...
Supervisor:
Karen Bradshaw

Grahamstown, South Africa
08 May 2014

Chapter 1

Background

1 Introduction

Graphics processing units (GPUs) have increased substantially in performance over the years, allowing them to outperform central processing units (CPUs) in certain aspects [7]. Not only have GPUs started to outperform CPUs, but they have also become cheaper in terms of Flops[1]/dollar [11]. This has created an interest in general purpose GPUs (GPGPUs) in an attempt to access the growing processing power of GPU for non-graphical applications. A GPU however has the limitation that it must be run in lock-step (where a group of cores execute the same instruction). This adds limitations to what processes can benefit from being run on the GPU. This paper will cover both GPGPUs as well as de-shredding algorithms. The first half covers GPUs and what tools are available for creating GPGPU applications. The second half covers a broad outline of shredding and then looks at different de-shredding algorithms that have been implemented.

2 General processing on a graphics processing unit

There is no single architecture that is optimal for running all types of workloads, and most applications use a mix of workloads during their runtime. A control-intensive application, where there are many branch instructions, would run best on a CPU, while a data-intensive application, where the same instruction needs to be run on multiple pieces of

[1]The number of floating-point operations per second, giving an accurate measure of the performance of the processor.

data, would run far better on a GPU [12]. Owing to the fact that GPUs have to run in lock-step the performance obtained by running a control-intensive application on the GPU would be far lower than running it on the CPU. Because of this GPUs are mainly used for processes that work with the lock-step requirement.

2.1 Graphics Cards

The processing power of a GPU is greater than that of a normal processor on a given die. However, it contains a constraint where a group of cores must execute the same instruction (lock-step execution). This causes the GPU to be faster but less versatile then normal processors (such as the CPU) [7]. Recently attention has been drawn towards using GPUs to accelerate non-graphic performance; this recent interest is due to two factors of the GPU: its price / performance ratio, and its evolution of speed [11]. The evolution of speed is mainly due to the gaming industry demanding better graphics, which in turn requires faster graphics cards. This demand has made the GPUs speed double approximately twice a year [11]. Since there is such a high demand in the gaming industry, the price of the actual graphics cards has remained relatively cheap in comparison to a CPU [11]. Rendering a graphical scene with a GPU is made up of a sequence of processing stages. These stages run in parallel and in a fixed order, which is known as the graphics hardware pipeline [11]. From a simple perspective there are four main stages in this process:
1) In stage one (vertex processing) the GPU obtains a 3D polygraph mesh, which is then converted into a 2D screen position render. During this, color and texture coordinates associated with the vertex are evaluated [11].
2) Within the second stage the vertices are grouped into triangles, and then scan-converted which generates a set of fragments. These fragments stores the state information needed to update a pixel [11].
3) During the third stage (fragment processing) texture coordinates of the fragments are used to get colors from one or more textures. Mathematical operations are then performed to determine the ultimate color for the fragment [11].
4) The final stage is comprised of various tests (such as depth and alpha tests), which will determine whether the fragment should be used to update the pixel [11].
There are two major producers of GPUs, NVIDIA and AMD (Radeon), and each has its own programming structure: NVIDIA use CUDA, and AMD use OpenCL. Both CUDA and OpenCL are discussed in the next section of this paper.

2.2 Development environments

CUDA

Developed by NVIDIA in 2006, Compute Unified Device Architecture (CUDA) introduced new components that were designed specifically for the GPU and solved several limitations of GPGPU [13]. CUDA introduced unified shader pipelines, which allows for each arithmetic logic unit (ALU) to be used to perform general-purpose computations. In order to create a GPU that can be used for more general purpose computing, NVIDIA had to build the ALUs in compliance with the IEEE standards for single-precision floating-point arithmetic. Once CUDA was developed the main issue NVIDIA faced was that there was no way of programming for CUDA without using Open Graphics Library (OpenGL) or DirectX which forced the programmer to disguise the program as a graphical task. To ensure people could access the power of CUDA, NVIDIA created the language CUDA C which was built on top of a restricted version of the C language with a few additional keywords that allowed the user to access features of CUDA [13].

OpenCL

The Open Computing Language (OpenCL) was released in 2008 by the Khronos Group [1], a nonprofit technology consortium [12], and has been managed by them since. OpenCL is classified as a heterogeneous framework, and so it supports application development that can be run across different devices manufactured from multiple vendors. This allows for an application to be run on a system that consists of GPUs and CPUs, and allows the programmer to utilize both these components [12].

OpenCL is coded in OpenCL C which was built on top of a modified version of the C99 language with additions for running data-parallel code. This allows for code that is developed for one device to be run on any other hardware that is OpenCL compliant, creating device-independent applications [12]. The OpenCL architecture is split into four models: the platform model, the execution model, the memory model, and the programming model [15].

Platform model: The platform model is comprised of a host connected to OpenCL devices. The host is classified as any computer with a CPU running on a standard operating system while the OpenCL devices can be GPUs or a multi-core CPU [1]. A

OpenCL device is generally made up of multiple compute units, each of which contains multiple processing elements. These processing elements execute instructions either as single instruction, multiple data (SIMD) or as single program, multiple data (SPMD) [1]. SIMD instructions are generally run on GPUs while SPMD instructions are generally run on CPUs [1]. Figure 1 displays how the platform model is designed.

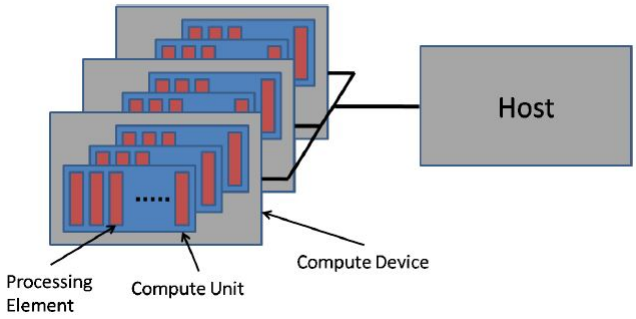

Figure 1 - Platform model (taken from [1])

Execution model: The execution model is separated into two sections: kernels and host programs. A kernel is a unit of executable code that is run on the OpenCL devices. When a kernel is queued, it is split into multiple work items, where each work item executes the same instruction on a different piece of data. This is stored in an n-dimensional index space, and from this each work item is given a global ID. So for an image processing kernel, it will be stored in a 2-dimensional index space and the work item for the pixel x = 80 and y = 93 would be given the global ID of (80,93) [1]. Work items can also be grouped together to form work groups which allows the work items to share local memory within the group, and gives the added benefit of being able to use synchronization on the work items within the group.
The host program is responsible for managing and executing kernels. This is done through the use of a context, which is a space where the OpenCL devices that are in the same context can share objects such as programs, kernels and data buffers [9].

Memory model: Memory space in the OpenCL framework is split into 4 sections: global memory, constant memory, local memory, and private memory. Global memory is the section of memory that all work items have full access to for both the OpenCL device and the host device. This section of memory can only be declared by the host during runtime. Constant memory is part of global memory, and only allows read access to work items. The host however has full access to the constant memory section. Local

memory is stored within a work group, and only work items within the work group have access to this section of memory. Private memory is a small section of memory that is only accessible to a single work item [1]. An extended version of this model includes the host memory, which contains the application data structures and the programs data, as well as the PCIe memory, which is part of the host memory but is accessible by the host and the OpenCL device. However, to modify this memory a synchronization state must be formed between the CPU and OpenCL device [2]. Figure 2 displays how the memory model is designed.

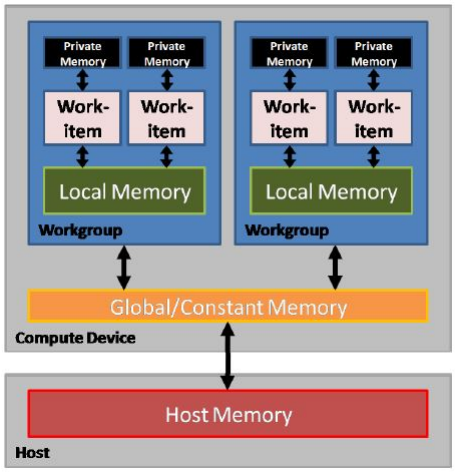

Figure 2 - Memory Model (taken from [1])

Programming model: The programming model simply maps the hardware threads to the GPU hardware [12].

3 De-shredding algorithms

Paper shredding is a popular way of destroying critical pieces of information. Even today, many important documents are not stored electronically but instead stored as hard copies [6]. Shredding can be seen as a weak method of cryptography, where it separates paper into blocks of data and then rearranges the block. This method will secure the data but because no data is altered during this process, anyone with the patience and means will be able to access the information after some time [6].

A document can be shredded in multiple ways depending on the type of shredder, each of which has a different level of security. The security of the shredder is determined by the

complexity of the shredding, where long strips of paper with a large width provide the lowest security and small strips of paper with a narrow width have a higher security level. According to Brassil [4] the US Department of Defense classifies a Class I secure shredder to be crosscut in nature with output pieces being no larger the 0.79375 mm wide and 12.7 mm long, This ensures that a single character 10 points in size will be vertically cut multiple times. He goes on to state that this does not ensure that the document cannot be reconstructed, but will at least deter attempts at reconstruction.

There are two popular methods for shredding a document, namely, straight line shredding and cross-cut shredding. Each provides a different level of security and satisfies a certain need of the user such as speed over security, as discussed in the next subsection of this paper.

The act of de-shredding is to take a document that has previously been shredded and attempt to put it back together to view the information. This is often used by government officials to re-assemble possible evidence. Reconstruction of a shredded document however is a challenge, and requires a large amount of resources [5]. This challenge led to the DARPA shredder challenge in 2011 created by the US Defense Advanced Research Projects Agency (DARPA) [3]. The DARPA challenge was a contest where teams had to reconstruct multiple sets of documents, with the difficulty in the reconstruction increasing with each document ranging from 224 pieces to over 6000 pieces to reassemble [3]. The teams then used the information obtained from the documents to answer specific questions and were awarded points for full or partial correct answers [3].

3.1 Straight line shredding

A straight line shredder is the most basic of shredders and so is also the easiest to reconstruct. The shredder only cuts along the papers length and commonly has widths ranging from 3.175 mm to 7.9375 mm. The mechanism makes use of two arrays of rotating ribbed metal bands, the width of which determines the width of the resulting strips of paper. This style of shredding is favored by those who wish to shred multiple documents quickly where the shredder is able to handle pages either stapled together or bound by paper clips [4].

3.2 Cross-shredding

A cross-cut shredder is a more secure method of shredding as it cuts both vertically and horizontally, which gives an added level of security. In most cross-cut shredders the horizontal cut will only occur every few inches on a page [4]. The dimensions of a cross cut shredded piece of paper range from 9.525 mm wide by 80.16875 mm long to 079375 mm wide by 4.7625 mm long [16]. The horizontal cuts with the cross cut shredder do not necessarily happen at the same time for each strip giving the added complexity of offset strips as shown in Figure 3 [5].

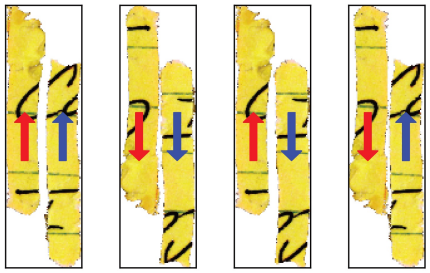

Figure 3 - Example of the offset of a cross-cut shredder (taken from [5])

An extension to the cross-cut shredder is the confetti shredder, which includes crumpling the resultant shredded pieces [4].

3.3 Implemented Solutions

Owing to the increased interest in this field, people from around the globe have attempted to create a solution to the deshredding problem. The DARPA challenge involved more then 9000 teams attempting to solve this problem [8].

DARPA shredder challenge

The DARPA challenge [3] forced teams to design different ways of reconstructing multiple shredded documents. The top team "All Your Shreds Are Belong To U.S" made use of a computer-vision algorithm, coded by the team, which found possible matches and asked the user to verify the match. The solution included a scoring system, and made use of a repeating set of yellow dots found on the pieces of paper to help with the matching

algorithm to return the best possible matches and to save time by having as little human input as possible [5].

The second placed team, "Shroddon", used a similar approach, but also included a text recognition aspect to the matching algorithm in an attempt to determine the next most probable character in the word using dictionaries.

The sixth placed team, "UCSD", decided on a crowd sourcing approach, which proved effective for the first few documents, but owing to fraudulence by the users they were forced to implement a security feature, which would determine how often a user can move a piece depending on their previous success [3].

Straight-line De-Shredder

The De-Shredder, developed by Chan, Gillespie, and Leong [6], was designed to handle straight line shredding of a single page. The method used is a pixel matching algorithm that matches the pixels on the leftmost side of one strip with the rightmost side of another strip [6].

After scanning the strips on a red background, each strip is fed through a MatLab script, which, using a color threshold technique, forces each pixel on the strip to either a pure red, pure white, or pure black pixel. A MatLab function[2] is used to find the centroid of the strips which in turn is used to rotate the strips so that they are aligned vertically [6]. These strips are finally processed by the De-Shredder program.

The De-Shredder program creates two arrays for each strip, one containing the leftmost pixels of a strip and the other containing the rightmost pixels of the strip. These pixels are defined by the rightmost/leftmost pixel that is not a red pixel. These arrays are used in the matching algorithm. Once the arrays have been set up, the program checks for matching pixels between the left edge array of one strip with the right edge array of another strip; if the colours between the two strips are a match the matching coefficient increases. The amount by which the matching coefficient increases depends on the colour of the matched pixel. If the color of the matched pixel is white the matching coefficient will increase less than if there are two matching black pixels [6].

After the initial matching algorithm has generated the matching coefficient, the De-Shredder shifts one of the pieces up or down a single pixel and generates another matching

[2]The MatLab function is called bwlabel: http://www.mathworks.com/help/images/ref/regionprops.html

coefficient for the two altered pieces. This is repeated for $\pm 0.25\%$ of the total number of pixels along the strips edge, with the final matching coefficient being the largest coefficient for the two pieces [6]. This helps avoid errors that may occur owing to small errors in the scanning process or previous image processing. A matching coefficient for every possible pair is generated.

Next, the program requires that the user specify which strips are classified as edge strips, for use by the reconstruction part of the process. Once the edge strip is specified, the program shows the user the strip with the highest matching coefficient, to which the user responds by confirming whether the strip is a match. This process continues as the document is slowly built up starting from one of the edge pieces. The user can view the reconstructed document to see if it is correct and easy to read at any point during this process [6].

Chan et al. [6] states that the De-Shredder program works well in re-constructing single page documents, but is far from a practical solution. The program is unable to handle multiple pages, or pages that have been shredded using the cross-cut method. The program also relies on human input to accept matches, which can take time, increasing the total time taken for the document to be reconstructed.

Cross-cut Deshredder

Deshredder is a program developed by Butler, Charkraborty, and Ramakrishan [5] designed to reconstruct shredded paper shred by a cross-cut shredder. Deshredder uses an array of methods to reconstruct the document such as: vertical edge time series, Luma time series, color targeting matching, and user interaction.

Deshredder takes as input all the scanned pieces on a single sheet. The program then separates all the pieces into single shreds, which are then straightened using a vertical edge time series. The vertical edge time series gives the distance from a perfectly straight vertical line to the leftmost or rightmost pixel for each row in the image. The average of these distances are then minimized to find the optimum θ (see Figure 4 for an example of this process). When implemented on the first test of the DARPA challenge, this method produced results with only a 2% error, which came about owing to the shred not being completely separated from other pieces. [5].

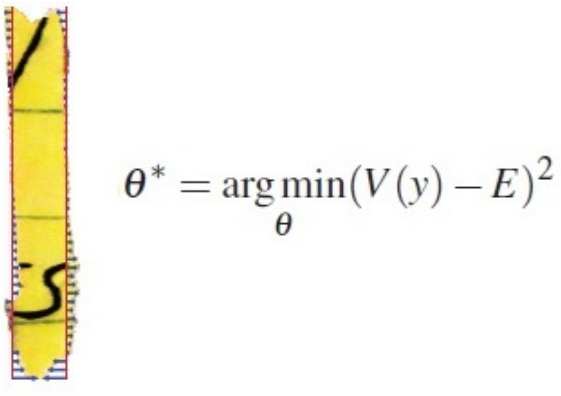

Figure 4 - Example of a vertical time series (taken from [5])

A Luma time series shows the leftmost and rightmost pixel in each row of the image. This is used to find the largest peak values along the edges, which are then used to create a Chamfer distance distribution. This allows the program to locate features of a shredded piece, which will produce a Chamfer similarity value. Once the similarity value is produced the program uses the value to attempt to match two strips that have similar features (For cross shredding it may only be a partial match as the pieces do not fit squarely with each other.) Figure 5 shows the feature matching in this process. This method is not perfect as shredded documents are generally sharp and distinct, and so an almost match would fail just as much as a complete mismatch would [5].

Figure 5 - Example of a Luma time series match (taken from [5])

Color targeting allows for human input during the matching process. The user selects a specific color he/she would like the matching algorithm to look at and the system highlights all the colors that are similar to the selected color. The user is now able to use a threshold to vary how much the system should consider certain colors. This can help the

system focus on features that the user deems more important [5]. The user then moves to the match selector section of the program. The user selects which piece he/she would like to attempt to match to which the program responds with the pieces that have the highest possibility of matching. Once the user has selected a match the program stores the match in the reconstruction palette, which allows the user so see all the matched pairs and allows the user to do multiple transforms on the strips to attempt to perfect the match [5].

Reconstruction of shredded document based on image feature matching

The system developed by Lin and Fan-Chiang [14] uses a two stage approach: the first stage uses image-based techniques to create a matching algorithm, while the second uses graph-based algorithms to restitch the document. The system was developed for a straight-line shredder, and was tested using a single page that was both digitally 'shredded' and physically shredded [14]. The shredded document is scanned on a blue background, and then put through object segmentation and length normalization. An image processing technique known as morphological erosion is used to remove the noise on the boundaries of the shredded piece that may have occurred during the shredding process and/or the scanning process. This is done by passing a structuring element (small shape or template) over the image. The structural element is a small 2-dimensional binary grid consisting of a pattern of "1"s and "0"s, where one of the points is set as the origin point. This point can be anywhere in the grid depending on what the programmer wants to do with the structural element. As the element passes over the image, the system does a logical check to see if the element fits at each point in the image. Equation 1 [10] shows the rule used for erosion where s is the structural element, and f is the image. The element is positioned so that the origin is placed above pixel (x,y) and then the rule is applied to the pixel [10].

$$g(x, y) = \begin{cases} 1 & \text{if } s \text{ fits } f, \\ 0 & \text{otherwise,} \end{cases}$$

Next the system uses a histogram obtained from the horizontal projection of the shred to align the text lines of each of the shredded pieces with every other piece, which is vital to the matching process [14].

The system identifies three special types of pieces found in a shredded document: blank pieces, leftmost piece, and rightmost piece. The blank pieces, which are defined by not

having any text visible on them, are removed as they are of no importance to the reconstruction process and would be difficult to near impossible to place correctly. The leftmost and rightmost pieces are identified as having no black pixels on the left/right section of the piece and having black pixel on the right/left section of the piece, respectively [14]. These pieces are then used as the starting and ending points in the reconstruction process.

Using the horizontal projection histogram created above, the system locates and creates text blocks within each shredded piece separated by blank blocks, as shown for the first six strips in Figure 6. The system creates a binary pattern for each strip. This is achieved by first creating a shred model (the final strip in Figure 6), which is a strip that contains every possible location for a text block using the locations of the text blocks in each shredded piece [14]. Each piece is compared to the shred model to create the binary pattern, where a '1' is assigned to those sections that include a text block, and a '0' is assigned to the sections that do not include a text block matching the shred model's text block. Based on the binary codes created, the system is able to group and sort the shreds using binary operations [14].

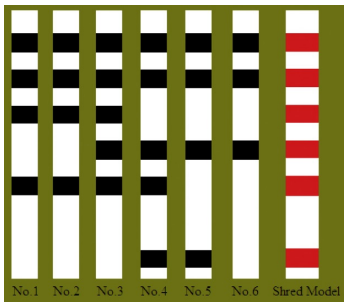

Figure 6 - Example of the text blocks and shred model created using the horizontal projection histogram (taken from [14])

Next the system gives the shredded pieces a similarity measure, where a higher score indicates a higher correlation between the two shreds. Lin et al. [14] used two methods, namely, shred coding discrepancy and average word length, for doing this.

Shred code discrepancy The shred coding discrepancy check uses the binary pattern created for each shred and checks these for differences. Each difference gives the pair an accumulative negative score, so two pieces with no differences will have a perfect score of zero. A smaller score between two shreds can also indicate spatial familiarity [14]. Since this gives a broad check for similarity, the information gained is not accurate enough to begin the reconstruction process.

Average word length The average word length check is the second method used to create the similarity score between two pieces. As a unit of measure pixel count does not work for finding the word length in a reconstructed section owing to the fact that the character size may differ both between the documents and also between different text areas in the same document. As an alternative, Lin et al. [14] suggested a method where each shred is divided vertically into four blocks. These blocks are either part of a word or part of a blank space between the words, where the blank space between words will always take up a full partition. The word length is taken as the rounded integer strip width [14]. Figure 7 shows an example of the vertical dividing of the strips and how the word length is measured.

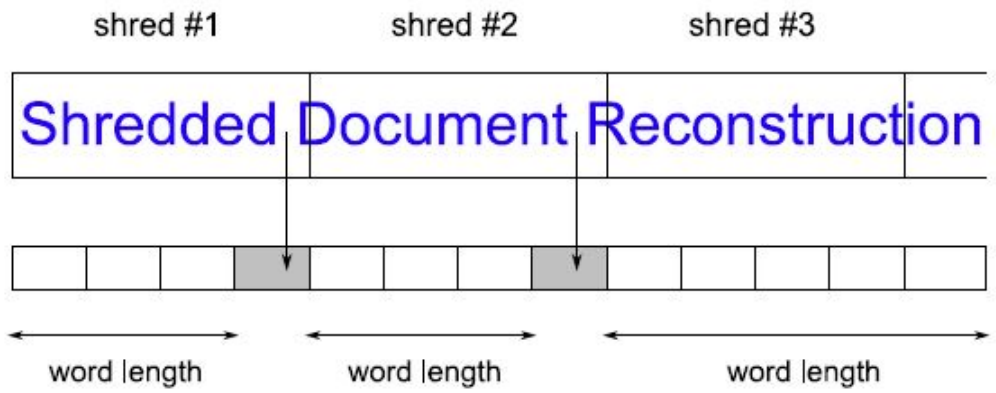

Figure 7 - Example of how the word length is calculated (taken from [14])

Once the system finds the word length, a negative score is accumulated based on the difference between the word length and the average word length. Since word length in a document varies, a high score simply implies that the shred permutation is reasonable [14].

The reconstruction section of the process is divided into two stages and uses a graph based scheme to sort them. In the first stage, a digraph is created based on the scores given by the binary pattern matching. The directed edges are weighted by the bit difference between the pair of binary codes [14]. The second stage then focuses on finding the shortest path of the digraph created in the first stage. Since the edge pieces are defined in the special selection stage at the beginning of the deshredding process, it is possible to find the shortest path between theses two points [14].

4 Summary

As a result of the creation of OpenCL and CUDA, it has become much easier to perform non-graphical processing on the GPU. This provides a means to solve tasks that would not have not been feasible to run on a CPU. By using a GPU the deshredding problem should be more possible to solve in a shorter amount of time.

5 Plan of Action

Date	Event
03 March 2014	Formal written proposal completed
04 March 2014	Project Presentation
30 May 2014	Literature review and plan of action
6 - 20 June 2014	First Semester Exams
22 June 2014	Start developing a deshredding algorithm for the CPU
06 July 2014	Test deshredding algorithm on the CPU with a digitally shredded document
08 July 2014	Start developing the algorithm to be run on the GPU
20 July 2014	Start pre-paring for Seminar
29 July 2014	Seminar Series 2 - Part 1
5 August 2014	Seminar Series 2 - Part 2
12 August 2014	Seminar Series 2 - Part 3
18th August 2014	Test the deshredder on the GPU
1 September 2014	Debugged the algorithm
2 September 2014	Test the deshredder algorithm with an actual shredded document
7 September 2014	Start short paper
15 September 2014	Short Paper Submitted
18 September 2014	Test performance between GPU and CPU
1 October 2014	Start official write-up
20 October 2014	First draft handed in
20 October 2014	Start preparing for Seminar
24 October 2014	Second draft created and handed in
27 - 29 October 2014	Seminar Series 3
31 October 2014	Project deadline
1 November 2014	Modify research website
7 November 2014	Research Website Complete
19 November 2014	Research oral exam

References

[1] Advanced Micro Devices. *Introduction to OpenCL Programming*, 1 ed. AMD, May 2010.

[2] Advanced Micro Devices. *AMD Accelerated Parallel ProcessingOpenCL Programming Guide*, 2.7 ed. AMD, 2013.

[3] Aron, J. Inside the race to crack the world's hardest puzzle. *New Scientist 212*, 2841 (2011), 26 – 27.

[4] Brassil, J. Tracing the source of a shredded document. Tech. rep., Hewlett-Packard, August 2002.

[5] Butler, P., Chakraborty, P., and Ramakrishan, N. The deshredder: A visual analytic approach to reconstructing shredded documents. In *IEEE Conference on Visual Analytics Science and Technology 2012, VAST 2012* (Seattle, WA, 2012), pp. 113–122.

[6] Chan, H. W., Gillespie, E., and Leong, D. Design and Implementation of a Paper De-shredder. Online, December 2010. Accessed: 30 May 2014. Available from: `https://courses.ece.ubc.ca/412/term_project/reports/2010/deshredder.pdf`.

[7] Chongstitvatana, P. Putting general purpose into a gpu-style softcore. In *Int. Conf. on Embedded Systems and Intelligent Technology* (2013).

[8] Deever, A., and Gallagher, A. Semi-automatic assembly of real cross-cut shredded documents. In *19th IEEE International Conference on Image Processing* (2012), pp. 233–236.

[9] Diop, T., Gurkfinkel, S., Anderson, J., and Enright Jerger, N. Distcl: A framework for distributed execution of opencl kernels. In *International Symposium*

on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) (2013).

[10] Efford, N. *Digital Image Processing: A Practical Introduction Using Java*. Addison-Wesley, 2000.

[11] Fan, Z., Q.-F. K. A. Y.-S. S. Gpu cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing* (2004), p. 47.

[12] Gaster, B. R., Howes, L., Kaeli, D. R., Mistry, P., and Schaa, D. *Heterogeneous Computing with OpenCL*, 1.2 ed. Morgan Kaufmann, 225 Wyman Street, Waltham, MA 02451, USA, 2013.

[13] Jason Sanders, E. K. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, 2011.

[14] Lin, H.-Y., and Fan-Chiang, W.-C. Reconstruction of shredded document based on image feature matching. *Expert Systems with Applications 39*, 3 (2012), 3324 – 3332.

[15] Munshi, A., Gaster, B. R., Mattson, T. G., Fung, J., and Ginsburg, D. *OpenCL Programming Guide*. Addison-Wesley, 2011.

[16] Office depot. What level of security do you need? Tech. rep., Office depot, May 2008.