Drift corrected frame averaging and image alignment using a graphics processing unit for image processing

Pluk, A.H.M.

Award date:
2008

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Drift Corrected Frame Averaging and Image Alignment

Using a Graphics Processing Unit
for image processing

By

A.H.M. Pluk

Supervisors:

Robert van Liere (TU/e)
Gert-Jan de Vos (FEI)

Eindhoven, November 2007

I. Preface

This document describes the results of the graduation project of Arno Pluk and is the final report concluding the Master program Embedded Systems of the Eindhoven University of Technology (TU/e). The graduation project has been carried out at FEI Electron Optics B.V. (FEI) in Eindhoven, Netherlands.

FEI develops, produces and sells Transmission (TEM) and Scanning (SEM) Electron Microscopes, Focused Ion Beam and Dual Beam Systems. These electron microscopes are tools that support a submicron level of observation, analysis and manipulation. The microscopes of FEI are complex systems focused on the professional market. Software is an important aspect of these products (over 100 man years), which ranges from intelligent hardware boards with real-time firmware (C and Assembly code) to client applications (C#, C++, VB) for automatic or manual control of the microscope.
The research has been performed at the Imaging software group within FEI, which develops and supports components for processing, display, and storage of images taken from electron microscopes.

Acknowledgements

A lot of FEI employees showed their interest in my project results, which encouraged me to improve the work that I performed for the project. I would like to thank all my colleagues from FEI for their inspiring words and comments in the past months. There are some people that I would like to mention explicitly. First of all there are my supervisors Gert-Jan de Vos from FEI and Robert van Liere from TU/e who helped me with all my questions about the project. Next to that I would like to thank Sander Stoks, Remco Schoenmakers and Auke van Balen from FEI for their input in the project, the meetings and the reviews of all the documents.

Arno Pluk

Eindhoven, November 2007

II. Summary

Using the graphics adapter for computation in image processing is a new trend. Strongly parallelizable algorithms can be executed many times faster on a graphics adapter than on the CPU. FEI would like to equip their image processing library (RIPL) with routines which are optimized for execution on a GPU. GPUs have only limited processing capabilities, but the large number of processors on modern graphics adapters provides programmers with large amounts of parallel processing power.

Drift Corrected Frame Averaging (DCFA) is a computationally intensive algorithm applicable to electron microscopy software. The parallelizable nature of parts of the algorithm makes it a good candidate for implementation on the GPU. The first target of this project is to use the GPU to perform DCFA in order to make live display of drift corrected images possible.

A fast computation of the alignment of tilted images for tomography reconstruction provides the user with the ability to reconfigure alignment parameters in real time. A GPU implementation of image alignment can make this possible. The second target of this project is to adapt the GPU implementation of DCFA to reduce processing time of an image alignment to less than 1 second.

To reach these targets the most suitable GPU hardware and software environment for FEI is selected and tested. An implementation of DCFA is constructed using this GPU environment and this implementation is extended to perform image alignment. Benchmarking the implementations showed that live display of drift corrected images was achieved for 1024 x 1024 pixel images and that alignment of a set of 512 x 512 pixel images was achieved within 1 second.

Table of Contents

Table of Figures

1 Introduction

1.1 Background

1.1.1 Image processing at FEI

At FEI a Raw Image Processing Library (RIPL) has been developed, which is a modular image processing library containing all image processing functionality. In the industry it is common practice to use dedicated image processing hardware to reach good performance for such functionality. From a cost and maintenance point of view, FEI wants to limit the use of dedicated hardware as much as possible. A new trend in reaching good performance for these functions is the use of a graphics adapter (GPU). Certain algorithms can be executed many times faster on a graphics adapter than on the CPU. FEI would like to participate in this trend by equipping their image processing library with routines which are optimized for execution on a GPU.

1.1.2 GPGPU

General Purpose computing on a Graphics Processing Unit (GPGPU) is a trend using the GPU to perform computation tasks. GPUs are designed to process graphics and therefore have only limited processing capabilities. However, modern graphics adapters are equipped with a large number of processors which can produce a large amount of processing power when combined. GPGPU tries to make full use of this processing power by rewriting algorithms to utilize those parallel capabilities.

Programmability of GPUs has been available through programmable vertex and pixel shaders, which were programmed using shader models. These shader models made GPUs complicated to program and were not very flexible. Since shader programs became more general, vertex shaders and pixel shaders were combined into one type of shaders which could perform general computations. The largest GPU manufacturers ATI™ and NVIDIA® each provide their own way to facilitate programming of the shaders without the use of shader models.

1.2 Problem definition

1.2.1 Drift Corrected Frame Averaging

Modern electron microscopes provide a submicron level of observation. A drawback of the high levels of magnification is that the higher the level of magnification is, the more electrons will hit the sample at that point. This could result in a charge building up in the sample, causing a part of the sample to drift away, or even possible damage to biological samples. To avoid this, a quick scan of the sample is performed to assure that only few electrons will hit the sample. This results in low intensity images with a high contribution of noise, which is undesirable as resulting image. In order to produce images that lack this drawback but still provide a very high level of magnification, the successive images of a quick scan can be averaged. Since the noise within the successive images is random, averaging multiple images will reduce the contribution of the noise and will improve the intensity of the actual data within the resulting image.

In practice, the solution of averaging successive images from the microscope is not as ideal as it may seem. Because of the high level of magnification, the sample we are analyzing is not fully motionless. There is always some kind of vibration due to the mechanics or even due to acoustic noise. In addition the sample under analysis can build up a charge from the electron beam which results in a drift of the sample. Both these effects have as a consequence that the average of these images will be blurred making it unacceptable as resulting image.

A solution to get rid of these blurring effects is to do drift corrected frame averaging (DCFA). In essence this means that first the drift (translation) between two successive frames is computed, after which one of the images is shifted to correct for the drift. Averaging with this shifted image will not result in blurring in the image, but it still reduces the contribution of the noise in the image and improves the intensity of the actual data in the result.

Drift Corrected Frame Averaging is not a new concept in image processing and it is also not completely new to FEI. The main reason that it is not incorporated in their products is because of the computational intensity of the solution. The current implementation of DCFA can correct four images of 512 x 512 pixels per second, consuming all available processor power. This is clearly not enough to support live display of the corrected images (+/- 50 images per second).

The first target of this project is to use the GPU to perform DCFA in order to make live display of drift corrected images possible.

1.2.2 Image Alignment

For tomography, a series of images with different viewing angles (called ‘tilt angles’) is taken from a sample in order to reconstruct 3-dimensional information on the sample. The successive images within this series need to be aligned to prevent similar blurring effects as mentioned above. The accuracy of this alignment is an important factor determining the quality of the final reconstruction. The alignment process for the reconstruction is a computationally intensive application in which the user constantly needs to configure and tune several parameters to get to an optimal alignment. With each change of parameters the alignment has to be recalculated and the user has to wait for this to finish before the result can be checked and possibly parameters can be readjusted. Since calculations of a typical set of 140 images takes around 2 minutes to complete the user has no intention to repeat this process a dozen times. Therefore the user has to deal with all aspects of the calculations off the top of his head in order to make the right changes in the parameters. A fast alignment (< 1 second) can ease this job for the user, providing the ability to reconfigure the parameters and immediately see the effect of that configuration to the alignment. This would increase the usability of the software significantly.

The second target of this project is to adapt the GPU implementation of DCFA to reduce processing time of an image alignment for tomography to less than 1 second.

1.3 Research

In order to know whether we can use a GPU for these image processing tasks, a number of questions and goals have been formulated.

1.3.1 Goals

What hardware and software environment is appropriate for the implementation of image processing on a GPU?

The environment should accommodate the following parameters: Fast computation, orders of speedup compared to CPU implementation; Easy to implement general purpose algorithms on a GPU; Cost-efficient solution, large amount of processing power per euro; Availability, now and in the future.

Can a DCFA implementation be constructed that achieves live display of images?

The implementation of DCFA must achieve a performance of 50 images of 512 x 512 pixels per second in the selected environment to accommodate live display of drift corrected images.

Can the DCFA implementation be adapted to achieve image alignment for tomography in less than 1 second?

The implementation of the image alignment must achieve a performance of less than 1 second for a set of 140 images in the selected environment using 1024 x 1024 pixel images which are typical for tomography.

1.4 Structure of the document

To select a GPU platform for image processing, research is performed into the available GPU technologies both in hardware and in software. In chapter 2, one of those platforms is selected and extensively studied. Benchmark tests are performed to indicate the speedup with respect to a CPU.

In chapter 3 research is performed into the actual computation of DCFA. This will be combined with the current implementation of DCFA to produce a DCFA implementation on the GPU platform. That implementation will be thoroughly tested and the timing results in chapter 5.1 will provide us with an answer as to whether a GPU can be used to reach live computation and display of DCFA images. The conclusion of these tests will be in chapter 7.

In chapter 4 the GPU implementation of DCFA is adapted to support image alignment for tomography. This implementation will also be tested and the results in chapter 5.2 will answer the question if image alignment for tomography can be achieved in less than 1 second. The conclusions from these tests will be in chapter 7.

In chapter 6 the integration in the image processing library of FEI (RIPL) is discussed, together with the possibility of adapting the implementation for other algorithms and possible integration of these algorithms.

1.4.1 Intermediate documentation

During the internship a number of intermediate documents were prepared for FEI, reporting on the research performed for the project. These documents cover: a complete description and overview of the benchmarks that were performed in order to compare the CUFFT library with the FFTW library (1); a derivation of cross correlation with Fourier transforms is given in (2); (3) contains a short introduction in the CUDA architecture and a guideline in writing (efficient) CUDA kernels; in (4) a description of all steps performed in the development of shift measurement with CUDA are supplied; Finally (5) describes how to use the final product created for FEI.

(1) DSR 6565, A. Pluk, *Benchmarking CUFFT and FFTW*, June 2007

(2) DSR 6567, A. Pluk, *Fourier transforms and cross correlation*, August 2007

(3) DSR 6568, A. Pluk, *Writing CUDA kernels*, October 2007

(4) DSR 6569, A. Pluk, *Performing a shift measurement on the GPU*, October 2007

(5) DSR 6570, A. Pluk, *Using the DCFA library created with CUDA*, November 2007

2 NVIDIA® CUDA™

To select a GPU platform for image processing, research is performed into the available GPU technologies both in hardware and in software. Appendix A provides a classification of available GPGPU techniques on the numerous platforms. It recommends the use of NVIDIA® CUDA™ as software architecture and the GeForce® 8800GTX as GPU. We will follow this recommendation for the following reasons.

2.1.1 Speed

The whole CUDA architecture is especially designed to facilitate general purpose computing on multiprocessor GPUs, opposed to ATI's Close To Metal (CTM) solution, which provides developers with a low level assembly to support GPGPU. This is reflected in for instance the memory model on the GPU which provides support for gather and scatter operations. The design for general purpose computing makes CUDA more efficient on those tasks than the traditional workarounds using a graphics API or ATI's CTM.
Next to the advantage of the CUDA architecture, NVIDIA currently is the market leader in GPU multiprocessors; the GeForce 8800GTX is the fastest high end graphics card available in April 2007. It consists of 16 multiprocessors each containing 8 processing elements, providing a total of 128 processing elements which can work in parallel. This provides the GeForce 8800GTX a peak performance of 518 GFLOPs[*].

2.1.2 Ease of use

The CUDA programming interface offers a small set of C-extensions which allows a programmer to target part of a program for execution on a GPU. This makes programming a GPU a simple path for programmers familiar with the C programming language and far simpler than original GPGPU programming techniques using shader languages. Programming will also be easier than ATI's low level assembly language.

2.1.3 Availability

NVIDIA graphics cards running CUDA can be bought in most computer stores. NVIDIA has stated that all future generations of graphics cards will support CUDA so availability in the future is assured.

2.1.4 The downside

The biggest downside of CUDA is that it is only supported by modern NVIDIA GPUs from G8x on, making us dependent on the availability claim of NVIDIA. A smaller downside is that it uses its own compiler, making it more difficult to integrate it into an existing project.

[*] This is theoretical peak processing power. The GeForce 8800GTX has 128 processing elements (PEs) at 1.35GHz, each being able to run 1 multiply-add (2 FLOPs) and 1 multiply (1 FLOP) instruction per clock cycle: 128 PEs × 1.35 GHz × 3 FLOPs = 518.4 GFLOPs

2.2 FFT Benchmarks: CUDA vs. CPU

In order to test the speedup achievable by using CUDA, benchmarks were performed on CUDA implementations and on CPU implementations of the same algorithm. For this benchmark, the Fast Fourier Transform (FFT) algorithm was chosen, because this algorithm is used frequently in image processing and will be necessary for DCFA and image alignment.

2.2.1 CUDA benchmark

For CUDA there exists an implementation of FFT in the CUFFT library. The benchmark of the CUFFT library measures the time for copying data to the GPU, planning both FFT and its inverse transformation, executing both transformations a number of times, and finally copying the data back from the GPU. The memory copies to and from the GPU are deliberately included in the measurements, because they contribute the most to the overhead when a computation is performed on the GPU. This will have a large influence on the overall performance of the library.

2.2.2 CPU benchmark

As CPU implementation FFTW (http://www.fftw.org) is chosen since this is the most efficient library available. For FFTW some settings needed to be configured before the benchmark could be started. Since the CUFFT implementation performs single precision FFT computation this was also chosen as parameter for FFTW. An extensive planning of FFT's, which could be configured for FFTW, did not produce efficient speedups compared to the planning time, so this option will not be used in the benchmark. The option to utilize multiple CPU cores provided a significant speedup compared to the single CPU version, therefore it was selected. The FFTW benchmark measured the same steps, except for the copying of the data back and forth.

2.2.3 Result

The benchmarks of both libraries are performed on various image sizes. Computing the relative speedup factor of one library to the other provides us with a good graphical comparison between both libraries [Figure 1].

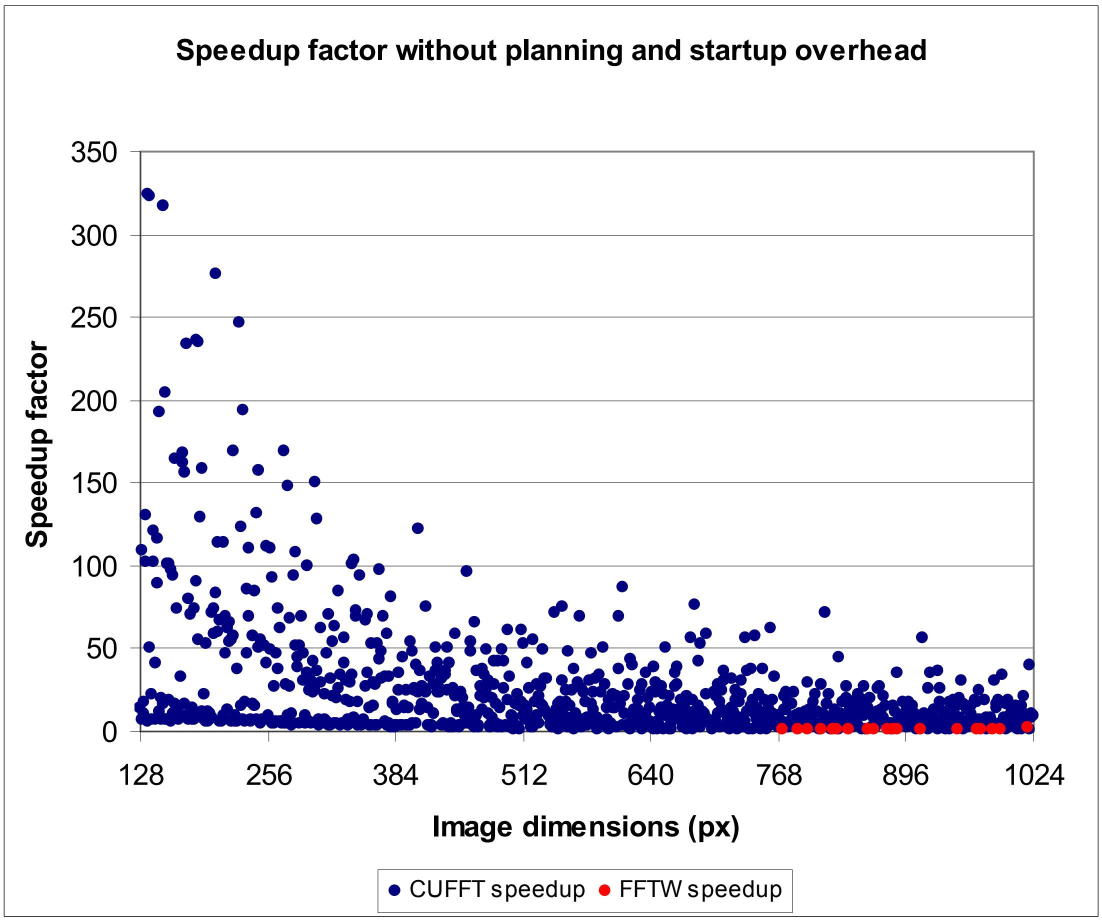


Figure 1: CUFFT and FFTW speedup factors

2.2.4 Conclusion

From Figure 1 we see that CUFFT performs better on most image dimensions. On average CUFFT performs 25 times faster than FFTW within the tested image dimension range (128 x 128 pixels to 1024 x 1024 pixels). For the dimensions that are of special interest to us (512 x 512 pixels and 1024 x 1024 pixels) CUFFT performs approximately 1 order of magnitude faster than FFTW. Since the CUFFT timings included some overhead of data transport, the performance advantage of CUFFT will only increase when more computations are performed on the GPU, distributing the overhead of data transport.

3 Drift Corrected Frame Averaging

3.1 General idea

DCFA is used to average successive images which are translated (drifted) with respect to each other. In essence it works by first computing the drift between two successive frames by means of cross correlation (3.2, 3.3.1), after which one of the images is shifted to correct for the drift and averaged (3.3.3).

3.2 Mathematics: Cross correlation

From the definition of cross correlation [6] we can derive its main application. Cross correlation ($\otimes$) is defined as:

$$(f \otimes g)(x) = \int_{-\infty}^{\infty} f^{*}(p) g(x+p) dp$$

Where $f^{*}(p)$ denotes the complex conjugate of $f(p)$.

The correlation effectively computes for every shift (x) how much the functions $f^{*}(p)$ and $g(x+p)$ align. This is done (in the discrete case) by looping over all function values (p) from $-\infty$ to ∞ and multiplying every function value of $f^{*}(p)$ and $g(x+p)$. When the positive areas in the functions align, the result of multiplication has a high value, and when negative areas align the result will also be high. Taking the complex conjugate of $f(p)$ ensures that aligned areas with imaginary components will also contribute positively to the integral. For the x value where the areas align best, the result will be maximal. When you find this maximum you know how much the functions are shifted with respect to each other.

Cross correlation can efficiently be computed with the use of Fourier transforms [1]. Applying the Fourier transform to both functions $f(p)$ and $g(p)$ and multiplying the complex conjugate of one resulting function with the other provides us with the Fourier transform of the cross correlation scaled by a factor $\sqrt{2\pi}$. Application of the inverse Fourier transform delivers the scaled cross correlation of $f(p)$ and $g(p)$. Since we are only interested in the shift vector between the functions, we only care about the location of the maximum and not in its value, so a scaled cross correlation is as effective for us as a normal cross correlation.

Fast Fourier Transform libraries are commonly available, so we are provided with a very computationally efficient way to perform cross correlation and shift measurement.

3.3 *Implementation*

3.3.1 Shift measurement

From the previous chapter we learned that the shift between two images can be measured by finding the maximum of a cross correlation, which can be performed in Fourier space. This is done by first applying a Fourier transform (3) to the image, secondly it is correlated with the Fourier spectrum of the previous image (5), an inverse Fourier transform is applied to the result (6) and finally the maximum in this result is computed, denoting the shift between the images (8). To perform this complete computation, a pipeline is constructed [Figure 2].

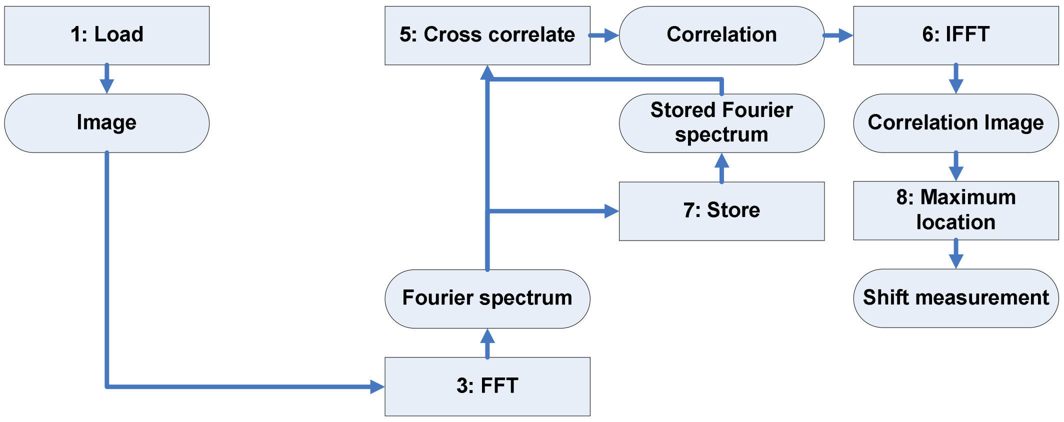


Figure 2: Shift measurement pipeline

3.3.2 Window and filter

To improve the shift measurement, a window and a filter function are introduced. The window function reduces artifacts that occur due to the borders of the image by fading out the intensity of the image near the borders. The window function (2) is applied on the original image data, so it is inserted into the pipeline before the FFT computation. Three window functions are implemented: a Hann window, a Hamming window, and a Gaussian window.

In order to reduce noise in the images, band pass filter functions let all frequencies pass within a given bound. An ideal band pass filter completely blocks all other frequencies. The other two band pass filters which are implemented, Butterworth and Gaussian, fade out the frequencies outside the band, resulting in fewer artifacts introduced by the filter. Because the filters (4) work in the frequency domain (Fourier space), they are introduced in the pipeline right after the Fourier transform and before the storage of the Fourier spectrum so the filter has to be applied only once per correlation.

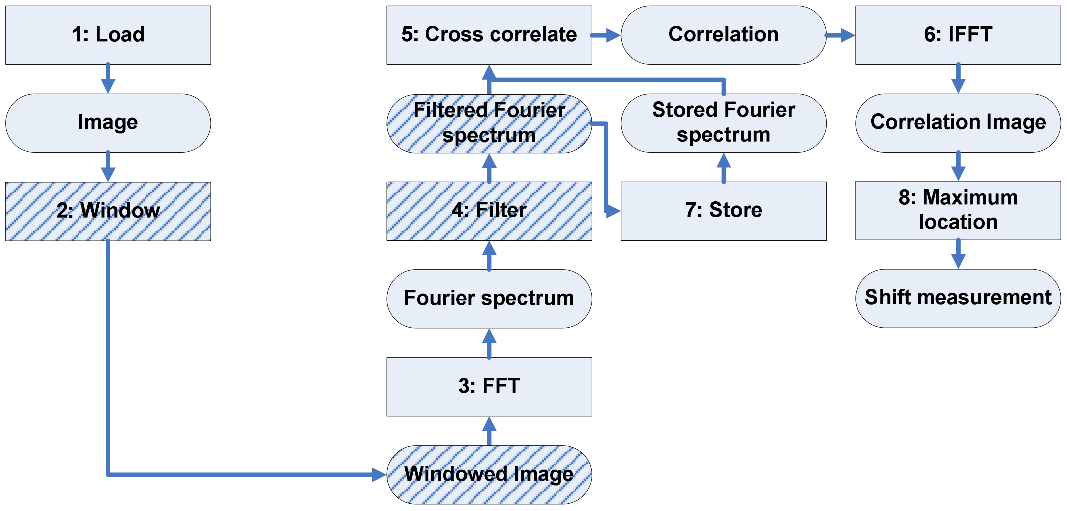


Figure 3: Shift-measurement with window and filter

3.3.3 Averaging

Once the shift between two successive images is known, the images can be translated back to compensate for the shift and an average can be computed based on factors set by the user. For this average (9) the original image is taken as starting point. The ‘Averaged image’ from the previous frames is translated according to the shift that has been calculated and is added to the original image. The result is stored in ‘Averaged image’ for use in the next frame.

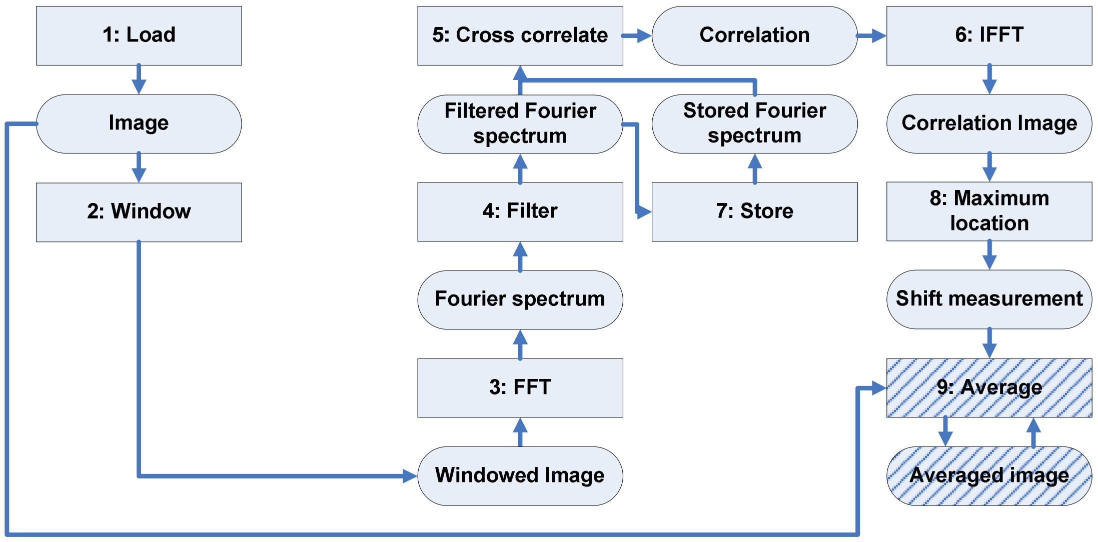


Figure 4: Drift corrected averaging

3.4 Extension: Affine registration

DCFA only computes and corrects for translations between images. It can be extended to make complete affine registration possible. Therefore, other techniques have to be used next to the cross correlation described above. These techniques were investigated for FEI, but not implemented in the project.
The registration of rotations between images can be performed by first transforming the images to a polar coordinate system and then performing a shift measurement. A shift in polar coordinates corresponds to a rotation in Cartesian coordinates. To get to complete affine registration we also need a computation of the scaling and shear between two images. Registration of scaling can be accomplished by a logarithmic transform of the input image. When a shift in logarithmic coordinates is calculated using the cross correlation, the scaling of the images can be computed from these coordinates. Proof or derivation of shear registration is not easily found in literature, only claims of complete affine registration are found for example in [2][3].

All these methods work in isolation. A computation of all transformations at once is more complicated. Using a parameter estimation method [4] translation, rotation and scaling can be determined in the presence of rotation within 45° and scaling within a factor of two, but not for arbitrary rotation and scaling. There are some methods for registration of translation combined with arbitrary rotation and scaling. Most of them are based on brute force techniques, e.g. computing the translation for every possible rotation within a given region and using the best correlation result [5]. The drawback of these methods is that the computational load grows very fast with the increase of transformation complexity.

Another approach is to use a method which iterates on the resolution of the images [3]. With a log-polar transform on binned images of low resolution the translation, rotation and scaling of images can be computed simultaneously. We can use the result of the binned image as initial guess for the next iteration using a higher resolution, eventually resulting in a registration of arbitrary translation, rotation, and scaling of the original images.

4 Image alignment for tomography

For tomography, a series of images with different viewing angles (called ‘tilt angles’) is taken from a sample in order to reconstruct 3-dimensional information on the sample. The successive images within this series need to be aligned for deviations caused by tilting the sample.

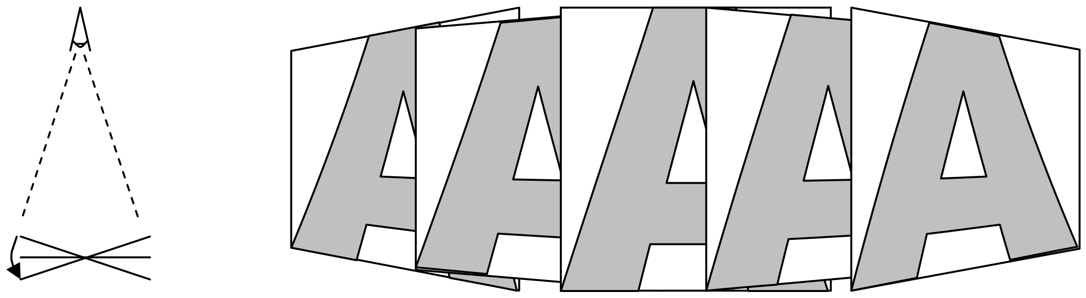


Figure 5: Example of tilt series for tomography reconstruction

A lot of components of the pipeline which was constructed for DCFA can be reused in the image alignment pipeline. The main difference lies in the deprojection to correct for the tilt of the image. The deprojection (3) stretches the image with the highest tilt to be in the same plane as the image with the lower tilt. The other image is just copied as is, so it is not possible to reuse the Fourier spectrum from the previous image. This means that Fourier spectra (4a and 4b) of both images and the filter (5a and 5b) of both Fourier spectra have to be recomputed every iteration and the intermediate result after windowing can be stored for computation in the next frame (6).

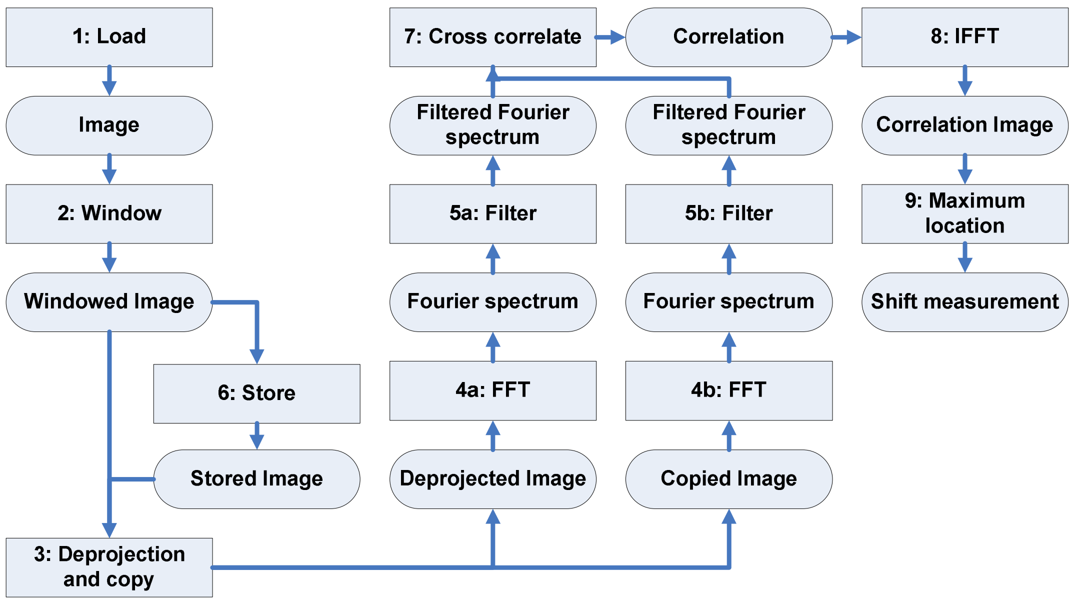


Figure 6: Example of tilt series for tomography reconstruction

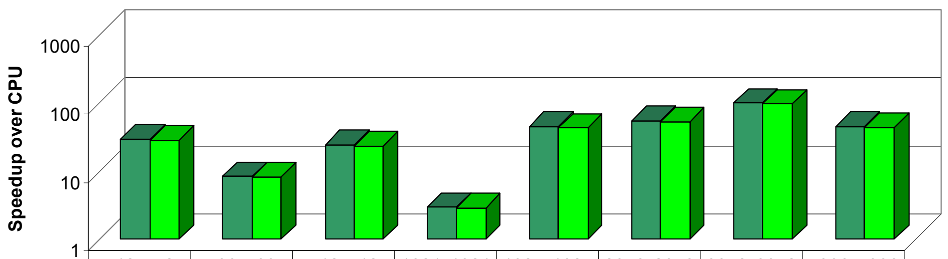
5 Results

The performance tests of the pipeline were carried out on a GeForce 8800GTX placed in a PCI Express x16 slot on an Asus M2N32-SLI Deluxe motherboard with an nForce chipset. The CPU comparison tests were performed on a Intel® Pentium® 4 2.8 GHz with 1 GByte RAM working with Microsoft® Windows® XP for the DCFA implementation and on a Intel® Xeon® 3.2 GHz processor with 4 GByte RAM working with Microsoft® Windows® XP 64-bit for the alignment.

5.1 DCFA

The implementation of DCFA on the CPU and the GPU were extensively tested to measure performance of both implementations. Tests were performed with the following image sizes: 512 x 442 pixels, which is used at this moment; 509 x 509 pixels, a prime image dimension; 512 x 512 pixels, the target image dimension; 1021 x 1021 pixels, another prime image dimension; 1024 x 1024 pixels, 2048 x 2048 pixels and 3096 x 3096 pixels for scalability measurements; and finally 4096 x 4096 pixels which is the maximum resolution currently needed for DCFA. The prime image dimensions were chosen because the tests of the FFT library have shown that prime image dimensions caused a drop in performance.
For all those image sizes the measurement was conducted over a DCFA of 1000 images on the GPUs and DCFA of 100 images on the CPU. Large sets of images were chosen to improve the accuracy of the timing results. Measurements were taken once only for DCFA and once for DCFA with a Gauss filter (F) and a Hann window (W) in the pipeline.

Speedup of DCFA over CPU implementation



	512x442	509x509	512x512	1021x1021	1024x1024	2048x2048	3072x3072	4096x4096
🟩 GeForce 8800GTX speedup	29,65	8,43	24,46	2,96	45,54	55,38	101,05	45,16
🟩 GeForce 8800GTX F W speedup	28,43	8,37	23,37	2,95	43,29	52,87	97,58	43,97

Image size (px)

Performance measurements DCFA (images per second)

	512x442	509x509	512x512	1021x1021	1024x1024	2048x2048	3072x3072	4096x4096
🟩 GeForce 8800GTX	210,13	19,13	257,46	2,21	79,41	15,85	5,24	2,28
🟩 GeForce 8800GTX F W	201,46	18,99	246,01	2,20	75,49	15,13	5,06	2,22
🟪 CPU DCFA W	7,09	2,27	10,53	0,75	1,74	0,29	0,05	0,05

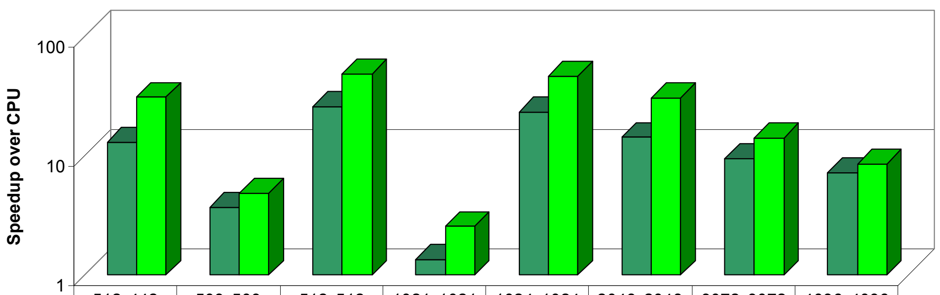
Figure 7: Performance measurement of DCFA on CPU and GPU

Results of the benchmark are given in Figure 7. A significant speedup (factor 25 to 100) with respect to the CPU implementation can be observed in common power-of-two sized images. The prime images show an increase of 4 to 8 times the CPU speed. The target performance of 50 images per second is easily reached for 512 x 512 pixels images and even for 1024 x 1024 pixels.

5.2 *Image alignment*

The performance of the alignment algorithm was measured for the same image dimensions. Since the alignment requires a window function to be applied to the image to produce reasonable results, the Hann window was applied in all measurements and only the Gauss filter (F) option was alternated for the measurements.

Speedup of Alignment over CPU implementation



	512x442	509x509	512x512	1021x1021	1024x1024	2048x2048	3072x3072	4096x4096
🟩 GeForce 8800GTX speedup	12,90	3,67	25,49	1,36	22,99	14,10	9,38	7,13
🟩 GeForce 8800GTX F speedup	30,59	4,85	47,49	2,55	45,76	30,15	14,04	8,42

Image size (px)

Performance measurements Alignment (images per second)

	512x442	509x509	512x512	1021x1021	1024x1024	2048x2048	3072x3072	4096x4096
🟩 GeForce 8800GTX	168,94	12,99	222,55	1,48	66,90	12,41	3,94	1,64
🟩 GeForce 8800GTX F	160,31	12,94	207,52	1,48	62,24	11,76	3,79	1,60
🟪 CPU Align	13,10	3,54	8,73	1,09	2,91	0,88	0,42	0,23
🟥 CPU Align F	5,24	2,67	4,37	0,58	1,36	0,39	0,27	0,19

Figure 8: Performance measurement of alignment on CPU and GPU

The results in Figure 8 show again large speedups with respect to CPU implementations. The target of 140 aligned images in 1 second is reached for 512 x 442 pixel and 512 x 512 pixel images, but not for 1024 x 1024 pixel images which we aimed at.

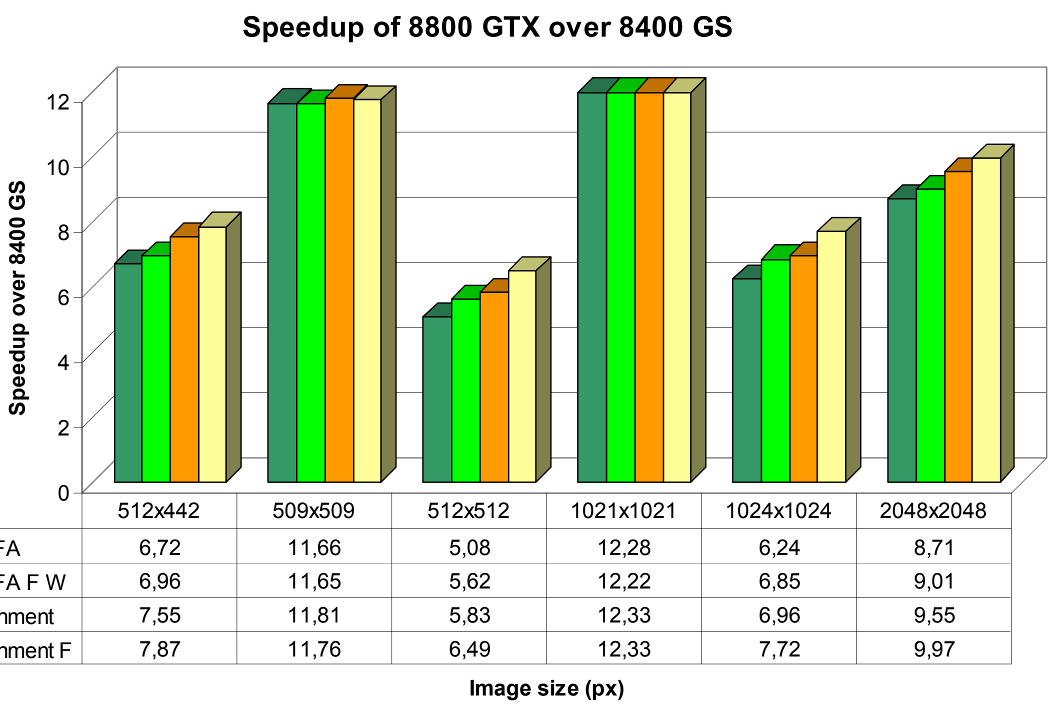
5.3 *Scalability*

Next to the performance increase gained with the GPU implementation, we are interested in the scalability of the implementation with respect to the configuration of the GPUs. There are two types of scalability that should be investigated. First there is the scalability in terms of the number of multiprocessors and the clock speed of the processing elements. Next to that there is scalability with respect to the memory usage of the algorithm, since GPUs have limited memory available and no paging mechanism.

5.3.1 Multiprocessor scalability

To test the scalability with respect to the number and speed of multiprocessors the previous tests were also performed on a GeForce 8400GS† on the same motherboard.

The fact that the GeForce 8400GS has 1/8th of the processing elements of a GeForce 8800GTX and is clocked at 2/3rd of that frequency, makes the performance estimation 1/12th of the 8800GTX performance. Also the memory bandwidth is 1/12th of the 8800GTX.



	512x442	509x509	512x512	1021x1021	1024x1024	2048x2048
🟩 DCFA	6,72	11,66	5,08	12,28	6,24	8,71
🟩 DCFA F W	6,96	11,65	5,62	12,22	6,85	9,01
🟧 Alignment	7,55	11,81	5,83	12,33	6,96	9,55
🟨 Alignment F	7,87	11,76	6,49	12,33	7,72	9,97

Figure 9: Speedup of implementations on 8800 GTX over 8400 GS

From Figure 9 we see that the actual measurements of the DCFA and Alignment give a more optimistic view, ranging the performance from 1/12th to 1/5th of the 8800GTX. The tests could be run without modifying the implementation of the algorithms, indicating a good scalability of the implementation of the pipelines.

† NVIDIA GeForce 8400GS; low range CUDA GPU,16 processing elements (PEs), 900 MHz, 256 MB memory, 16 PEs × 900 MHz × 3 FLOPs = 43.2 GFLOPs, available for approximately € 50,-

5.3.2 Memory scalability

Performing DCFA on the largest resolution of images (4096 x 4096 pixels) required a small modification to the pipeline, trading off efficiency for memory usage. This had to be done to be able to map the computation to the available memory on the GPU. Because the GeForce 8800GTX has 768 MB memory it was not expected that 4096 x 4096 pixel images would pose a problem. Allocation of memory for DCFA of 4096 x 4096 pixel images includes 7 computation buffers, which needs roughly 7 x 4096 x 4096 x 4 bytes = 448 Mbytes of memory in total. This leaves 320 Mbytes of memory for planning of the two Fourier transforms in the CUFFT library. A test is performed on the memory usage of the DCFA and the alignment algorithm for different image sizes. The results of these measurements give a practical overview of the scalability of the algorithm onto other graphics adapters, indicating the minimum amount of memory required for different image sizes.

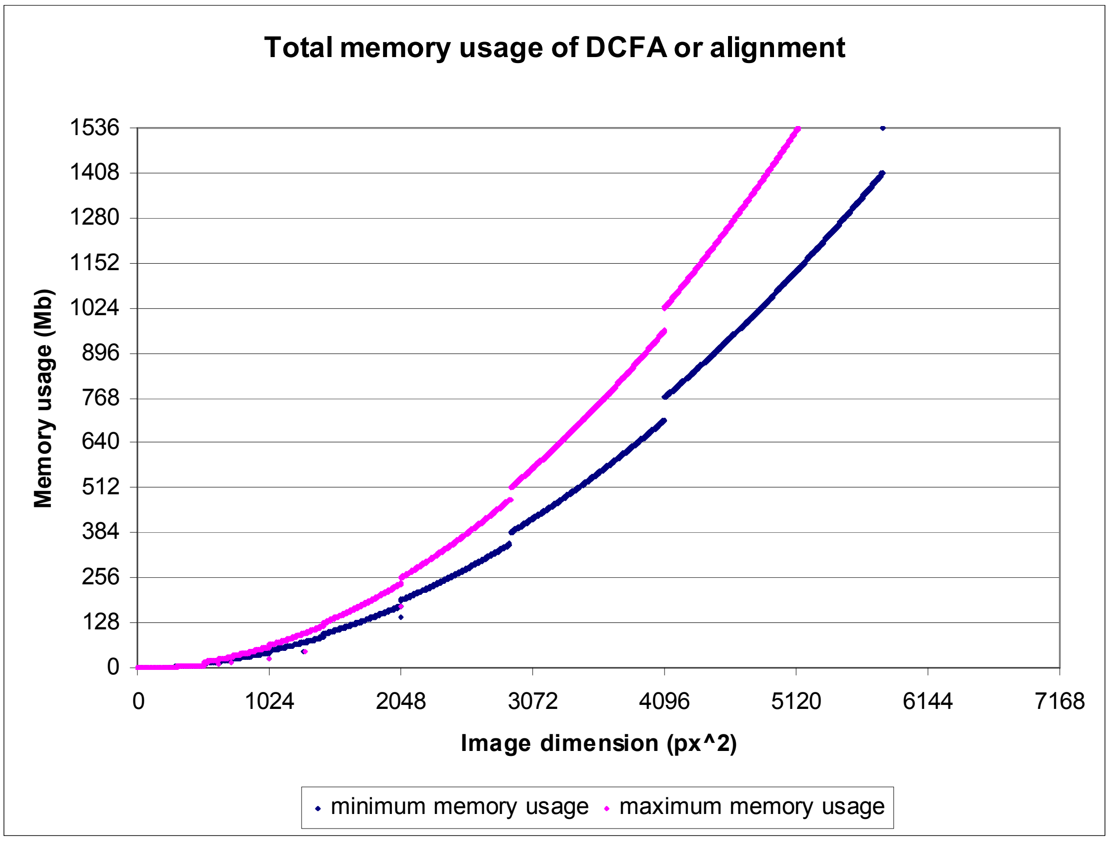


Figure 10: Memory usage measurement of the algorithms

In Figure 10 measurements of total memory usage of the DCFA algorithm and the alignment algorithm are plotted, once with maximum memory consumption and once with the abovementioned modification to reach minimal memory consumption handing in some efficiency.

6 Software engineering

6.1 RIPL integration

The implementation of the algorithms is integrated into the existing Raw Image Processing Library (RIPL). This is done by means of a high level application programming interface providing an abstraction over the complete algorithm implementations. It offers primitives for initialization, allocation of input and output buffers and configuration of filters and windows. Functions are available for performing a complete DCFA or an alignment using the configured windows and filters.

This high granularity is not an ideal method of integration in RIPL because the other functionality that is provided by RIPL consists of small blocks allowing the user to construct a pipeline from these blocks. It would be better to provide all blocks from the DCFA and alignment pipeline separately. Introduction of such generality will cause some inefficiency, because for example efficient reuse of buffers cannot be guaranteed. Since the whole project is intended to gain maximum performance this is clearly not desirable. Next to that is an issue of the place of computation. For users of RIPL there should be no need to know whether computation takes place on the GPU or on the CPU. However, transferring data to and from the GPU will cause serious performance loss, so optimally all GPU computations should follow each other, but this cannot be enforced. Trivial solutions to these problems will have a negative impact on the performance which is not desirable. Research into solutions that have a minimal impact on performance should be performed.

6.2 Pipeline adjustment

Adjusting the CUDA pipeline for other functionality is easily done using the components of DCFA and alignment. Most of the components work with an input and output buffer which should be allocated on the device in an initialization function for CUDA. Calling the components one after the other simply builds the pipeline. It is recommended to maximize the reuse of buffers in the pipeline to keep the solution scalable with respect to available GPU memory. An example of buffer usage for the alignment pipeline is given in Figure 11. The open dots indicate that the buffer is read in that function, the closed dots indicate that the buffer is written. The lines show the lifetime of the buffers. The three *Image buffers* store floating point data, the two *Complex buffers* store the Fourier spectra. The *Maximum buffer* is used during computation of the maximum and the *Deproject texture* is used by the deprojection algorithm.

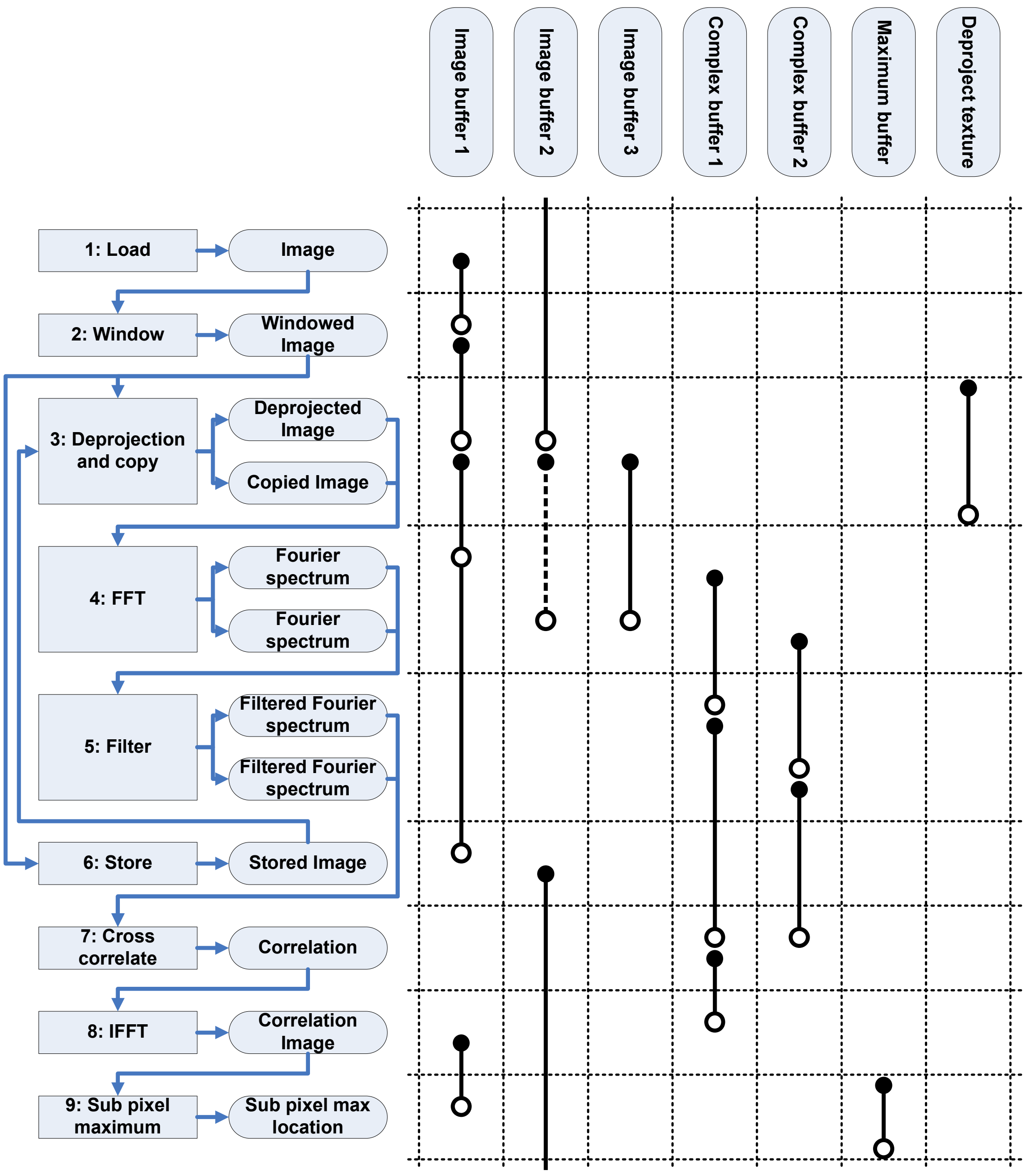


Figure 11: Buffer lifetime of alignment pipeline

7 Conclusion and recommendation

7.1 Conclusions

What hardware and software environment is appropriate for the implementation of image processing on a GPU?

In chapter 2 we discussed the CUDA hardware and software environment. From the FFT benchmarks in chapter 2.2 and the benchmarks from chapter 5 we learned that speedup of one or two orders of magnitude are possible in this environment, which proves CUDA to be very suitable for image processing on a GPU.

Can a DCFA implementation be constructed that achieves live display of images?

From the tests in chapter 5.1 we learned that live display of drift corrected images is certainly feasible with the DCFA implementation that was constructed. From Figure 7 we can even see that images of 1024 x 1024 pixels can be drift corrected at a live display rate.

Can the DCFA implementation be adapted to achieve image alignment for tomography in less than 1 second?

The benchmarks in chapter 5.2 show that we did not achieve the alignment of 140 images per second for images of 1024 x 1024 pixels. However we succeeded in adapting the DCFA implementation for alignment, which achieved more than 140 image alignments of 512 x 512 pixel images within 1 second. These are major advances in the speed of alignment software for tomography. The implementation can certainly be used to improve the usability of the software for example by providing the user with a 512 x 512 pixel preview of aligned images within 1 second and calculating the actual alignment in the background.

7.2 Recommendations

7.2.1 CUDA in the future

Because the CUDA technology is in constant development, it can be expected that hardware and software will improve in this area, providing again an increase in performance. In the near future 64 bit versions of CUDA are planned for release, which will support double precision arithmetic for more accuracy. However, these releases are not planned for all mainstream graphics adapters but merely for the compute-specific solutions. Driver support for 64 bit platforms is planned for release in the end of 2007.

The implementations of DCFA and alignment are designed in such a way that they scale automatically to newer or older CUDA GPUs which provide a different amount of multiprocessors. This is proven by the tests performed with the GeForce 8400GS, which required no modification to the code to reach a good performance.

7.2.2 Application of CUDA

CUDA is a very good architecture for image processing in general, because most image processing can be performed massively in parallel (per pixel). Therefore it should be used to speed up other computationally intensive image procession algorithms as well, for example registration of affine transforms or the complete tomography reconstruction.

To provide better support for CUDA in RIPL a tool needs to be developed which takes care of buffer management and data locations in the construction of a pipeline using CUDA (for example by performing an automatic just-in-time copy of data when a function is called). The optimal case would be that RIPL could take care of this, giving maximum flexibility to the user, but it would reduce the performance gain from using the GPU.

Appendix A. Classification of GPGPU environments

In this appendix a classification is made of available GPGPU environments that could be used for image processing. In A.1 an overview of hardware capable of GPGPU is given and A.2 lists the software environments. A.3 gives the combinations of hard- and software. Finally A.4 makes recommendations from this survey.

A.1 Hardware

For general purpose programming, there are various types of graphics adapters available. Also some extension cards for the PC are available with the same purpose. A classification of the most common cards is listed below. From the main GPU manufacturers, a low end card and a high end card are selected for this overview.

- **IBM Cell microprocessor**
  3.2 GHz stream processor with 8 processing elements, 230 GFLOPs peak processing power. There is a commercial PCI Express x16 board available for € 7000,-

- **ATI R5x** (low-end graphics adapter with CTM support)
  450 MHz stream processor with 6 processing elements, 14 GFLOPs peak processing power. PCI Express x16 board available for € 40,-.

- **NVIDIA G8x** (low-end graphics adapter with CUDA support)
  450 MHz stream processor with 16 processing elements, 43 GFLOPs peak processing power. PCI Express x16 board available for € 90,-.

- **ATI Radeon HD2900XT** (high-end graphics adapter)
  742 MHz stream processor with 320 processing elements, 512 MB memory and 475 GFLOPs peak processing power. PCI Express x16 board available for € 500,-.

- **NVIDIA GeForce 8800GTX** (high-end graphics adapter)
  1.35 GHz stream processor with 128 processing elements, 768 MB memory and 518 GFLOPs peak processing power. PCI Express x16 board available for € 500,-.

A.2 Software

Due to the difficult nature of general purpose programming on a graphics adapter, there are a lot of software solutions developed in this area. These are listed below:

- **Accelerator** Microsoft Research
  A data-parallel library in .NET based on DirectX, latest release in October 2006

- **Brook** Stanford University
  A streaming programming model written as a C extension which supports OpenGL, DirectX and ATI CTM which uses its own Brook compiler, latest release in October 2004

- **CTM** ATI/AMD
  A parallel low level assembly language for ATI specific GPUs (>= R5X) which uses its own ATI CTM compiler, latest release in November 2006

- **CUDA** NVIDIA
  A parallel high level language for NVIDIA specific GPUs (>= G8X) which uses its own NVIDIA CUDA compiler, latest release in June 2007

- **PeakStream** PeakStreamInc
  A commercial C++ application programming interface which supports computation on both GPU and Cell architecture via a Virtual Machine, latest release in March 2007

- **RapidMind** RapidMind
  A commercial C++ extension which supports computation on both GPU and Cell architecture via a Development Platform, latest release in July 2006

A.3 Graphical overview

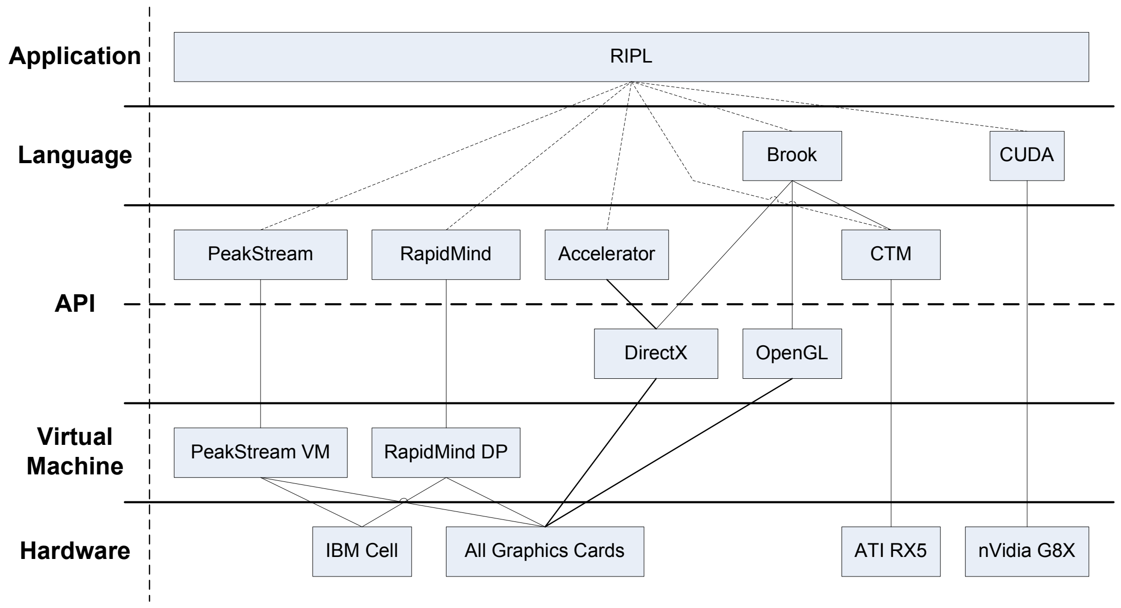


Figure 12: Overview of available GPGPU technology

A.4 Recommendations

The overall best software package claims to be PeakStream because is supports all graphics adapters and the Cell processor in a portable way. The main disadvantage of this package is that it is a commercial product, so dependency would always remain third party development and maintenance of the product. FEI products have a typical lifecycle of 10 to 15 years. During and even after that period they need to support their products, so all components need to be available and supported in the remote future. This results in a big vulnerability when you become dependant on (possibly small) third parties. For example, in June 2007 PeakStream was acquired by Google and its commercial activities stopped, proving it to be a good decision not to become dependant on their software. In the non-commercial software solutions Brook would be the best choice because it would support all graphics adapters. However this package is already outdated and does not seem to be developed any further, so it lacks the advantages of today's technology.

Therefore the choice that is recommended is NVIDIA CUDA. It has as disadvantage that it only supports NVIDIA graphics adapters, but the main advantage is that it is designed for general purpose programming and will take full use of all features of modern day graphics adapters.

At this moment the best choice in hardware also lies with NVIDIA in the GeForce 8800GTX. It has the highest number of GFLOPs, because of the large number of parallel processing elements and the high shader clock. The best performing low-end graphics adapter is also from NVIDIA, the GeForce 8500GT. These conclusions can only support the above recommendation to choose the NVIDIA-only software package.

Bibliography

[1]. A. Pluk, *Fourier transforms and cross correlation*, August 2007

[2]. R. Berthilsson, *Affine correlation,* Proceedings of the International Conference on Pattern Recognition ICPR'98, Brisbane, Australia, 1998, p.1458-1461.

[3]. G. Wolberg, S. Zokai, *Robust image registration using log-polar transform*, Proceedings of the IEEE International Conference on Image Processing, Canada, September 2000.

[4]. P. Thévenaz, U. Ruttimann, and M. Unser, *A pyramid approach to sub pixel registration based on intensity*, IEEE Transactions on Image Processing Vol 7, p. 27-41, January 1998.

[5]. E.D. Castro, C. Morandi, *Registration of translated and rotated images using finite Fourier transform*, IEEE Transactions on Pattern Analysis and Machine Intelligence 9, September 1987, p. 700-703.

[6]. E.W. Weisstein, *Cross-Correlation*, from MathWorld - A Wolfram Web Resource, http://mathworld.wolfram.com/Cross-Correlation.html