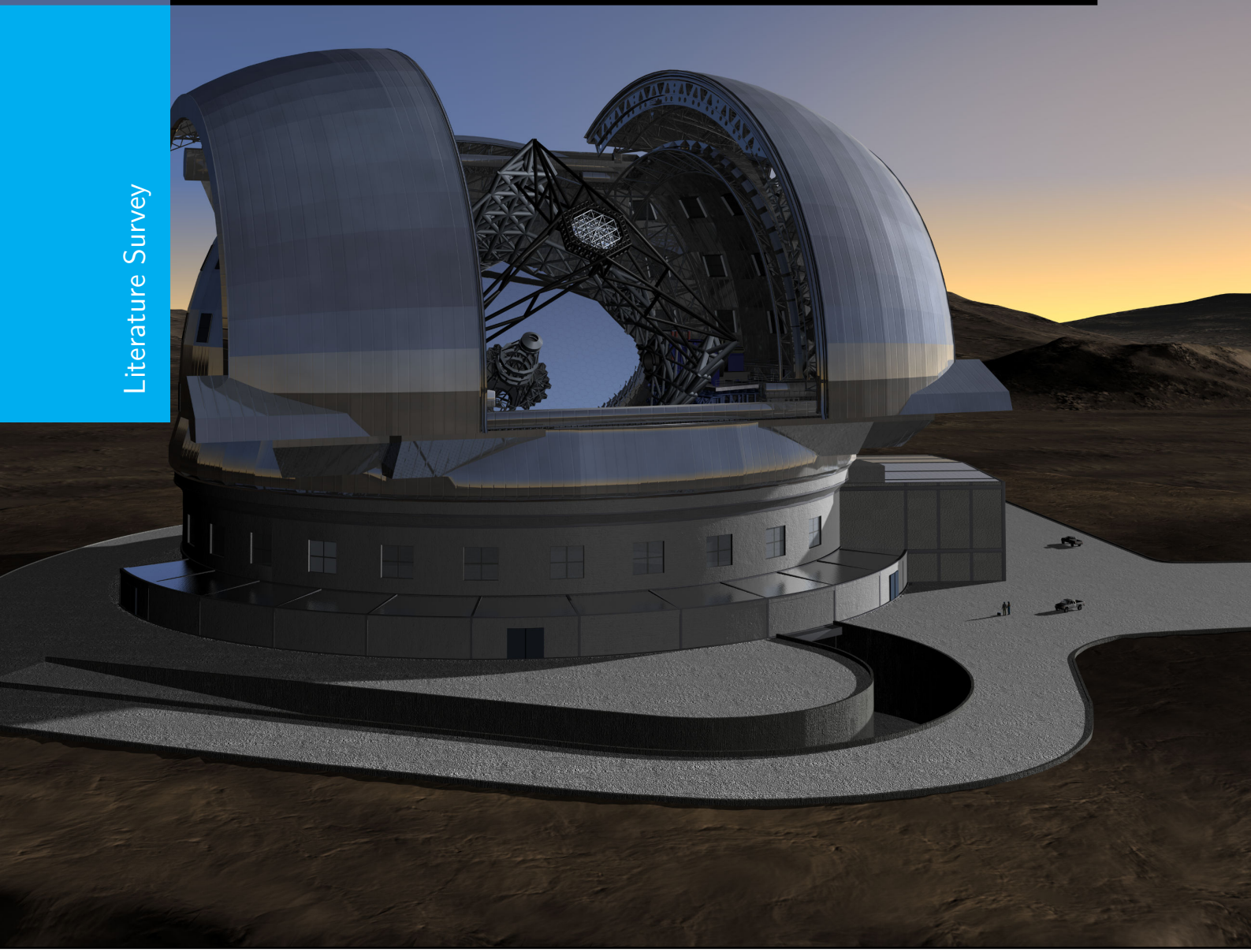
E-ELT under control

Adaptive optics low-level control solution for E-ELT

W.A. Klop

Literature Survey



TU Delft Delft University of Technology

Delft Center for Systems and Control

E-ELT under control

Adaptive optics low-level control solution for E-ELT

Literature Survey

W.A. Klop

May 26, 2011

Faculty of Faculteit Elektrotechniek, Wiskunde en Informatica (EWI) · Delft University of Technology

Abstract

Scientist are building increasingly larger telescopes and this is not only influencing the mechanical structure. An adaptive optics (AO) system is an essential instrument installed in large telescopes to prevent blurry images. The wavefront reconstructor is an component of the AO control process. When N defines the number of actuators, then the computational effort to calculate the wavefront reconstructor scales with the order $O(N^2)$ for standard vector matrix multiply (VMM) methods. For the class of extremely large telescope (ELT)s now under consideration these calculation become infeasible. Extensive research is already done on how to lower the order of wavefront reconstruction algorithms. However, expected is that lowering the order is not sufficient, emphasising the need for optimisation on all the levels of implementation. The parallel computing concept seems to be a excellent contender to further improve the performance on the level off hardware implementation. The link connecting the algorithms and the hardware lies in the conditions imposed on each technology that allows it to work effectively. During the research the conditions and properties of the different technologies were evaluated. Analysing to which extend there exists cohesion between a particular algorithm and hardware platform. From the three algorithms under consideration they all are suitable till a certain extend. The structured Kalman method excels with respect to performance and accuracy. However the parallel computing concept requires independency among data, preferable at a high level to achieve optimal performance. This aspect lacks in all three algorithms. Recommended is to perform additional research on the redesign or redevelopment of algorithms to fit better the parallel computing profile. Also is suggested to explore the option of a localised approach where only nearest-neighbour information is utilised to estimate a phase difference of a single segment.

Table of Contents

Preface & Acknowledgements **ix**

1 Introduction **1**
1-1 Telescopes . . . 1
1-2 Adaptive optics in astronomy . . . 2
1-3 Computational explosion . . . 3
1-4 Outline . . . 4

2 Parallel Computing **5**
2-1 General Aspects . . . 5
2-1-1 Background . . . 5
2-1-2 Basic Concepts . . . 6
2-2 Case Study . . . 8
2-2-1 Fine grained approach . . . 9
2-2-2 Coarse grained approach . . . 9

3 Parallel processors **11**
3-1 Development . . . 11
3-2 Architecture . . . 12
3-3 Applicability . . . 14
3-3-1 Overview . . . 16

4 Reconfigurable Computing **17**
4-1 Development . . . 17
4-2 Architecture . . . 18
4-3 Applicability . . . 20
4-3-1 Overview . . . 21

5 Adaptive Optics Algorithms **23**
5-1 Fast Fourier transform reconstruction . . . 23
5-1-1 General Info . . . 23
5-1-2 Algorithmic Structure . . . 24
5-1-3 Analysis . . . 25
5-2 Sparse minimum variance reconstruction . . . 27
5-2-1 General Info . . . 27
5-2-2 Algorithmic Structure . . . 28
5-2-3 Analysis . . . 28
5-3 Structured Kalman reconstruction . . . 30
5-3-1 General Info . . . 30
5-3-2 Algorithmic Structure . . . 30
5-3-3 Analysis . . . 31
5-4 Overview . . . 32

6 Findings **35**
6-1 Conclusions . . . 35
6-2 Recommendations . . . 36
6-3 Project Proposal . . . 37

A Additional Algorithm details **39**
A-1 Fast Fourier transform reconstruction . . . 39
A-2 Sparse minimum variance reconstruction . . . 41
A-3 Structured Kalman reconstruction . . . 41

Bibliography **43**

Glossary **47**
List of Acronyms . . . 47
List of Symbols . . . 48

List of Figures

1-1 Schematic representation of a Adaptive Optics system 2
1-2 Interweaved process of optimising a system design 4

2-1 Degradation of speedup by Amdahl's law. S_p is a function of (α) the fraction of non-parallelizable code. 7

3-1 Architecture overview of GPU [30] 13
3-2 Programming model of a the GPU programming language CUDA 14

4-1 Simple representation of a logic block 19
4-2 Architecture of FPGA 20

5-1 Structure FFT algorithm. Both the process steps of the Fried as the Hudgin geometry are depicted. The additional required by the Hudgin geometry are dashed. 24
5-2 Wavefront reconstruction algorithm 30
5-3 SSS matrix-vector multiplication as a series of subsystems 31

6-1 Graduation project proposal 37

List of Tables

3-1 Performance overview of the modern high performance GPUs: NVIDIA Tesla C2070 and the AMD FireStream 9270 12

4-1 Property overview from the FPGAs; Xilinx Virtex-6 SXT and the Altera Stratix V GS 18

4-2 Resource usage of floating point IP block when implemented in Xilinx Virtex-6 FPGA. Multiply and add operations in both single precision (SP) and double precision (DP) are presented. 21

5-1 Analysis of the conjugate gradient algorithm 29

5-2 An overview of the suitability wit respect to the categories; ELT, Order, FPGA and GPU. 33

Preface & Acknowledgements

The Literature Study is an element of the graduation process. The document at hand describes the findings I discovered during my research. The first step in the graduation process is selecting a topic. My personal preference of graduating within the Technical University led me towards the subject. After a consultation with Micheal Verheagen and Rufus Fraanje a global idea was defined. Finally the idea was refined and with this the subject became; Low level control of adaptive optics (AO) systems in relation with extremely large telescope (ELT)s. During the Literature Study I concerned myself with the issues related to a specific element of AO control. The implementation of the wavefront reconstruction process. To keep my research on track a received frequent supervision of my daily supervisor, Rufus Fraanje. For additional counsel I was able to discuss my problems with Professor Michel Verhaegen.

Delft, University of Technology W.A. Klop
May 26, 2011

"People asking questions, lost in confusion, well I tell them there's no problem, only solutions."

— *John Lennon* —

Chapter 1

Introduction

Exploring is a essential component of the human character, as long as we exist we are wondering what the stars in the universe will bring us. To find some answers we are building a wide variety of instrumentation. One of the most important ground-based instruments for astronomers is the telescope. Driven by our curiosity we are constructing increasingly larger telescopes. We are now arrived at the era of the extremely large telescope (ELT). This is a class of telescopes where the aperture exceeds the 20 meter range. Constructing telescopes from this kind of magnitudes comes with numerous challenges. Controlling the Adaptive Optics systems is one of these challenges which will be addressed in this document.

1-1 Telescopes

With the invention of the telescope in 1608 [35] it became possible to study the universe in much more detail. In contrast to the only possible way till then: the naked eye. The first telescope existed of an objective and ocular, also referred to as a refractor based telescope. The growth in diameter of the telescopes required another type of design. A new design based on a curved mirror was suggested by Isaac Newton in 1668. Despite the theoretical advantages, production technologies held back the introduction of the reflecting telescope till the 18th century. All current astronomical telescopes are based on reflective designs. Increasing the diameter of the telescopes is driven by two important properties in any optical imaging system: the light collecting power and the angular resolution. Where the Rayleigh criterion gives an estimate of the angular resolution and is defined by

$$sin\theta \approx 1.22\frac{\lambda}{D} \tag{1-1}$$

Where λ is the wavelength of the light emitted by the object that is observed, D defines the diameter of the aperture and θ gives the angular resolution. The smaller the angular resolution the more detail can be observed. Eq. (1-1) results in the fact that by increasing D the angular resolution will improve. Nowadays this makes us believe that an extremely

large telescope (ELT) can deliver an important contribution to our astronomical knowledge. Several of this type of ELT's are planned to be build in the coming decade. Among them are the astronomers of European Southern Observatory (ESO) who are designing the European version of an ELT since 2005.

1-2 Adaptive optics in astronomy

There is an issue that has to be resolved to be able to use the full extent of an ELT. When the diameter of the telescope is enlarged beyond approximately 0.2 meter the angular resolution in no longer diffraction limited, but seeing limited. Caused by turbulence in the earth's atmosphere. Each ground-based telescope has to pass the earth's atmosphere to retrieve the image. The light travelling through this atmosphere gets distorted caused by temperature differences, resulting in a blurry image. A solution would be to place the telescope outside the earth's atmosphere as was done with the Hubble telescope. Two major disadvantages that come with this solution is the inability to do rapid maintenance or repair and of course the extreme costs related to such a telescope. Another critical limitation is the size of the telescope. Bigger is advantageous for the light collecting power. However travelling to space is still complicated and does not allow for large payloads.

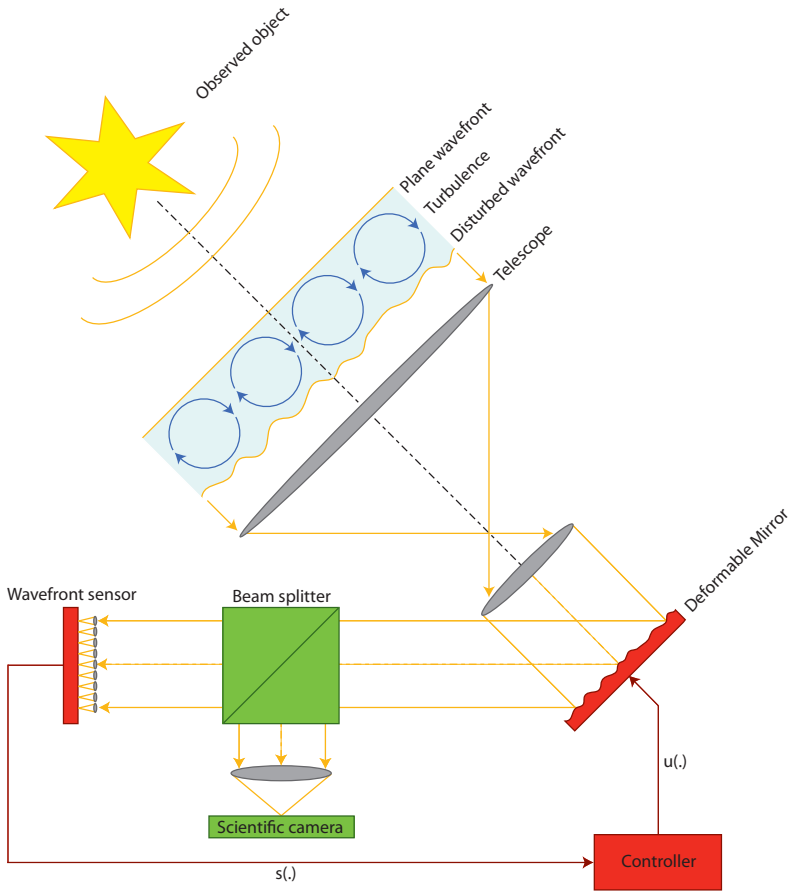


Figure 1-1: Schematic representation of a Adaptive Optics system

A more cost effective option which is often chosen at the moment is adaptive optics (AO), this is a technology to reduce the effects of wavefront distortions and improving imaging performance. AO achieves this by correcting phase differences originating from atmospheric disturbance. Atmospheric disturbance which can be represented as a state space innovation model of the form

$$\xi_{k+1} = A_d\xi_k + K_d e_k \tag{1-2}$$
$$\phi_k = C_d\xi_k + e_k \tag{1-3}$$

where ϕ is the phase difference, k is the time index and e is Gaussian distributed noise. Furthermore AO can be divided in three subsystems (Figure 1-1). The wavefront sensor (WFS) measures the phase distortion in the form of gradients $s_x[m,n]$ and $s_y[m,n]$. Where $s_x[m,n]$ and $s_y[m,n]$ define the first differences between adjacent phase points

$$s_x[m,n] = \phi[m+1,n] - \phi[m,n] \tag{1-4}$$
$$s_y[m,n] = \phi[m,n+1] - \phi[m,n] \tag{1-5}$$

A Shack-Hartman sensor is a commonly applied type of WFS. As each measurement is corrupted by noise the sensor can be modelled as a stochastic approximation

$$s_k = G\phi_k + n_k \tag{1-6}$$

Where G is the phase-to-WFS influence matrix and n is the WFS measurement noise. The second component is the controller. The WFS measurements are used to determine the optimal positions of the deformable mirror (DM) actuators. The DM compensates the measured distortion. Defining H as the DM influence matrix, u_k as the control inputs and z^{-1} is a discrete shift operator. The following equation represents the DM actuator correction.

$$\phi_{dm,k} = z^{-1}Hu_k \tag{1-7}$$

Hence, the residual phase error is defined Eq. (1-8). The objective in AO is to minimise this residual error resulting in a reconstruction of the associated wavefront and improving the image quality.

$$\phi_{\epsilon,k} = \phi_k + \phi_{dm,k} \tag{1-8}$$

1-3 Computational explosion

Not only the mechanical structure increases when building an ELT but also the control problem. Currently most control algorithms used in the adaptive optic systems of telescopes uses matrix inversions or other computational intensive computations to come to the solution.

Till now these methods were no issue. With the increasing size of the mirror also the number of WFS segments and the DM actuators increased. Keeping in mind the limit of approximately 0.2 meter for which telescopes are still diffraction limited. Resulting for the E-ELT in a total amount of actuators (n) that is expected to exceed the 40.000. Making an algorithm with order $O(n^2)$ or higher is not acceptable anymore. Already huge amount of research is done to more efficient algorithms. Even with these more efficient algorithms it is expected that solving the control problem is still not feasible in real-time considering current performance levels and Moore's law [11]. Each scientific problem can be optimised at several levels, already the top level has to be addressed. Also the lower levels can be improved, each algorithm can be converted to software in several ways, optimised for different hardware platforms. Note that software and hardware are tightly interconnected, changing the hardware often results in software changes. Even further, preference on the selected hardware can also influence the algorithm design or the other way around. Making the total system design from algorithm till hardware a complex interweaved process (Figure 1-2).

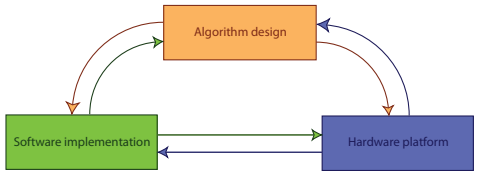


Figure 1-2: Interweaved process of optimising a system design

1-4 Outline

Already some experiments emphasising on the AO implementation with respect to ELTs were published. In [34] 16 Nvidia 8800 ultra fast graphics processing units (GPU) distributed over 8 dual core HP Optron computers are used to solve a 64×64 sized AO problem. Applying regular VMM methods they achieve a framerate of 2 kHz. Similar, in [19] 9 boards containing six field programmable gate array (FPGA)s each controlled by two general purpose computer boards packed with three Intel L7400 dual core processors solve approximately a 86×86 sized AO problem. In this case a conjugate gradient method is utilised to achieve a framerate of 800 Hz. Both options succeed for the given problem, however they are relatively small compared to the ELT related AO problem. Predicting whether the suggested designs are sufficient requires knowledge about bottlenecks related to technologies. This document describes issues and properties for both the technologies, as well for some algorithms. Allowing for the estimation of critical bottlenecks and in addition guiding towards a possible strategy. Chapter 2 explains the parallel computing concept. Dealing with the main aspects around parallel algorithms implementations. Chapter 3 considers parallel processors. Explicitly pointing out GPUs by addressing its advantages and disadvantages. Chapter 4 discusses reconfigurable computing. Where FPGA technology is considered as a potential candidate. Chapter 5 focusses on the AO algorithms, more accurate on the wavefront reconstruction component. Each of the three discussed algorithms is analysed for performance, accuracy and the expected success factor when mapped to a parallel version. Chapter 6 finishes the discussion with conclusions and recommendations based on the findings of the previous chapters. Also a proposal for the graduation project is given.

Chapter 2

Parallel Computing

Engineering often consist of making tradeoffs. Parallel computing is such a tradeoff, this technology trades space for time. Traditionally algorithms are performed sequentially. For scientific problems this approach is often impractical. The time it would take, before the results will be available can be enormous. Parallel computing addresses this issue and instead of executing the steps sequential the idea is to perform them simultaneous or in parallel. Related to the technology used to make the calculations known as integrated circuits, this means that available die area is used to solve the problem in a shorter time window. Especially in the case of real-time problems where time is sparse, like the AO systems in telescopes, problems can benefit from parallel computing technology.

2-1 General Aspects

2-1-1 Background

Usually when a problem has to be solved an algorithm is developed and then implemented in a sequential set of instructions. The execution of the instructions finds place on a single processing unit. The unit is executing one instruction at a time, after the first instruction is finished the next is started. This cycle keeps going till the algorithm is finished. The concept is easily comprehensible for programmers and as such it is straightforward to apply. This makes it an attractive option. On the other side we find parallel computing. This technology uses multiple processing units to work simultaneously on the same problem. To accomplish parallelism the algorithm has to be broken up into several independent subproblems. These subproblems then can be dived over the multiple processing units. Parallel computing is already a well establish field for a while and its primary domain of interest was high-performance computing, however this is shifting since physical limitations are preventing frequency scaling. Frequency scaling was from around 1985 till 2004 the dominant method used by IC manufactures to increase the performance. The execution time of a program can be determined by counting the number of instructions and multiplying this by the average time each instructions needs.

A way to decrease the average instruction execution time is by increasing the clock frequency of the processing unit. Resulting in a lower execution time of the overall program. This argument gave plenty of reason for manufacturers to let frequency scaling be the dominant design parameter. As already mentioned earlier this method became infeasible to hold as dominant design parameter. Power consumption is the fundamental reason for the ceasing enhance in frequency scaling. The power consumption of an IC is defined as Eq. (2-1) in [18, p. 18].

$$P = \frac{1}{2}CV^2F, \tag{2-1}$$

Where C is the capacitance switched per clock cycle. V is the supply voltage and F is the frequency the IC is driven by. The equation shows directly the problem, increasing the frequency will also increase the power consumption, if both the capacitance and the voltage are kept equal. Since also the voltage was reduced from 5 volt till 1 volt over the past 20 years some additional headroom was available. Slowly physical limitations were emerging. Burning vast amounts of power meant that temperature were rising to unacceptable levels. Gordon E. Moore made the observation that since the invention of the IC that the number of components in IC's doubled every year [26]. Moore also predicted that this trend would carry on, nowadays referred to as Moore's Law. Later on Moore refined the law and changed the period into two years. Till now Moore's law it still valid. From the beginning these new available resources were used to support higher frequencies. Since frequency scaling is no longer the dominant design parameter these components can be used for other purposes. Manufacturers are now aiming at increasingly larger numbers of cores favouring parallel computing.

2-1-2 Basic Concepts

When applying parallelism the design is dictated by the speed-up factor. The speed-up factor is defined by the relation between the algorithms original execution time and the execution time of the redesigned version.

$$S_p = \frac{E_{old}}{E_{new}} = \frac{1}{(1-F) + \frac{F}{S_{p(enhanced)}}} \tag{2-2}$$

Where S_p is the overall speed-up factor, E_{old} and E_{new} are the execution times of respectively the task without and with enhancement, F is the fraction of code, that could be enhanced and $S_{p(enhanced)}$ is the speedup factor of the enhanced code. Ideally the speed-up factor would be linear with respect to parallel computing, meaning that when the number of processing units increases the runtime decreases with the same factor. However, in practise this is almost never achievable. Several different issues play a role. The section of the algorithm that allows to be parallelized can have a major impact. Gene Amdalh observed this and formulated the potential speed-up factor on a parallel computing platform in Amdalh's Law [15, p. 66]. The law states that the speed-up of a program is dictated by the section of the algorithm which cannot be parallelized. The resulting fact is that the sequential part will finally determine the runtime. In case of a realistic scientific problem, which will often exists of both parallelizable and sequential parts, this has a significant influence on the runtime. The relationship defined by Amdalh's Law is formulated as

$$S_p = \frac{1}{\alpha + \frac{(1-\alpha)}{p}} \tag{2-3}$$

Where α is the fraction of the code that is non-parallelizable and p is the number of processing units. Putting the law in perspective: if for example 50 percent of the code cannot be parallelized, the maximal achievable speed-up will be 2x regardless of the number of processors added. The effect is shown in Figure 2-3, the number of processing units chosen here is 10 and the sequential portion is 50 %. The effect described by Amdahl's law will limit the usefulness off augmenting processing units.

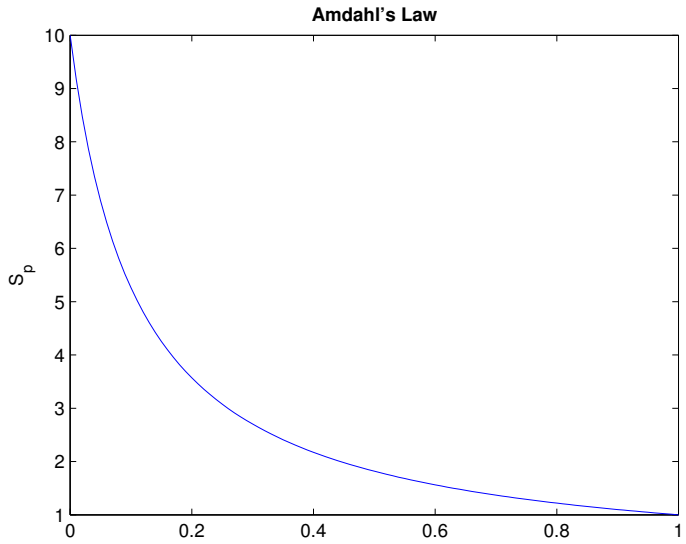


Figure 2-1: Degradation of speedup by Amdahl's law. S_p is a function of (α) the fraction of non-parallelizable code.

Data dependency or rather independency is the critical parameter in parallelism. Dependency in this context is the case where prior calculations have to be completed before the algorithm can proceed. The largest set of uninterrupted dependent instructions defines the critical path. The optimisation of an algorithm by parallelizing is limited to the runtime of the critical path. An example of a human analog to the critical path was called by Fred Brooks in [5]. It states "Nine women can't make a baby in one month". The same applies to certain sections of algorithms.
parallelizing algorithms can be performed at different levels. On an elementary operation level this is referred to as fine grained. Or on a level where a complete set of elementary operations form a subproblem, this is defined as coarse grained. Often it is more straightforward to parallelize on a fine grained level like matrix multiplications. There are often fewer data dependencies and certainly the insight in the functioning is better. However how tempting this may be, when feasible it will be better to follow the coarse grained approach.
Communication in the field of processing unit is expensive. When communicating the processing units have to use the memory to share their temporal results. Memory access is slow compared to the processing unit, causing the processing unit to stall. Communication is also seen as overhead since the calculation is the primary objective. The less communication, the

better. Knowing this argument, the advantage of coarse grained is quite obvious. When each processing unit can progress without intervention for a longer period of time the reduced communication will benefit the overall speedup.
Amdahls law assumes that the part that can be made parallel can be infinitely speed-up just by increasing the number of processing units. In practise, the observation will be that as soon as a certain number of processing units is reached, the speed-up factor will decline. In the book "The mythical man month" [5] this effect is also revealed with relation to late software development projects. Keep adding man power will not help to get the project finished in time and eventually will have a negative impact. The main cause is again the overhead of communication that increases, when then number of people working on a project is increasing. At a particular point meetings will dominate the time spend on a project. There is a large coherence to parallel computing, the only difference lies in the fact that people are replaced by processing units.
Load balancing plays another decisive role in the difference between the theoretical and practical speed up factor. Hardware comes frequently with a fixed number of processing units. When the algorithm is split up, the resulting number of subsections does not always match the number of processing units defined by the hardware. In case of insufficient subsections, it means that several processing units will be idle. In the opposite case when there are to much subsections, they have to split up in two groups. First executing one group followed by the next. In either case there is a inconsistency in comparison to the theoretical feasible upper limit.
The hardware architecture has impact on how an algorithm is converted to a parallel version. Michael J. Flynn [12] was one of the first to define a classification for computation platforms. The two classes which are interesting in relation to parallel computing are the single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD). SIMD has the same control logic for different processing units, this means that it performs the same instruction on distinct data. An advantage is that there is less die area required for the logic unit, making space available for more or advanced processing units. Opposite to SIMD stands MIMD, These types of hardware platforms have its own control unit for each processing unit. Allowing to perform different instructions on different data set creating a set of completely independent processing units. Providing much more flexibility in contrast to SIMD. Both technologies have their advantages and specific application area's. For example the MIMD architecture is applied in current multi-core CPU technology. Where the SIMD architecture can be recognised in GPU hardware designs.

2-2 Case Study

To show the difference between a fine grained and a coarse grained approach an example case is defined. The case is selected for its ability to fit for both approaches and has no direct resemblance with reality. Assume we have a square grid of 16 x 16 sensors that measures the wavefront phase shifts. These phase shifts are monitored over a period of time. A measurement from a single time instant can be represented as a matrix which is from now on referred to as ϕ, such that $\Phi \in \{\phi(1), \phi(2), ..., \phi(N)\}$. We are interested in the average phase shift over a 1 second time period, were the sample frequency is 500 Hz. The hardware used to solve the problems has 200 processing units and a SIMD architecture. A sequential algorithm that would solve the illustrated case is defined by AveragePhaseShift (version 1).

Algorithm 1 $AveragePhaseShift(\Phi)$ (version 1)

1: $A \leftarrow zeros(16, 16)$
2: **for** $k = 0$ **to** N **do**
3: $\quad A \leftarrow A + \phi(k)$
4: **end for**
5: $A \leftarrow A./N$
6: **return** A

2-2-1 Fine grained approach

The suggested algorithm is build from two elementary operations; a matrix summation and a elementary divide. Both operations are completely independent and can easily be converted to a parallel version.

Algorithm 2 $AveragePhaseShift(\Phi)$ (version 2)

1: $A \leftarrow zeros(16, 16)$
2: **for** $k = 0$ **to** N **do**
3: $\quad << initparallel(i, j) >>$
4: $\quad A[i, j] \leftarrow A[i, j] + \phi(k)[i, j]$
5: $\quad << synchronise(A) >>$
6: **end for**
7: $<< initparallel(i, j) >>$
8: $A[i, j] \leftarrow A[i, j]/N$
9: $<< synchronise(A) >>$
10: **return** A

First clarify some elements, on line 3 and 7 we find an $<< initparallel(i, j) >>$ command. The command resembles the procedure of distributing the proceeding code over a parallel processors. Similar the $<< synchronise(A) >>$ command on line 5 and 9 retrieves the results from the parallel processors and progresses sequentially. Note that each matrix element of a single elementary operation can now be calculated by a different processing unit. Obvious will be the fact that the main path of the algorithm is still sequential. Also each iteration a parallel start-up procedure and synchronising procedures has to be performed causing a reasonable amount of overhead for a relatively simple operation. The grid size was 16 x 16, meaning that this would result in 256 parallel processes, however the number of available processing units is hard defined by the hardware at 200. So each parallel calculation process has to be split in two batches and scheduled to be executed.

2-2-2 Coarse grained approach

In opposite to the fine grained approach, the algorithm is converted to a parallel version at top-level. The algorithm is an example of the type of algorithms that allows to be completely parallelized and promises to have a linear speed-up factor. There are two options, parallelize temporally or spatially. Since there are data dependency constraints in the time domain this

could be difficult. In contrary in the space domain each of the operations on the elements of the matrix are completely independent and can be computed separately.

Algorithm 3 $AveragePhaseShift(\Phi)$ (version 3)

1: $<< initparallel(i, j) >>$
2: $A[i, j] \leftarrow 0$
3: **for** $k = 0$ **to** N **do**
4: $\quad A[i, j] \leftarrow A[i, j] + \phi(k)[i, j]$
5: **end for**
6: $A[i, j] \leftarrow A[i, j]/N$
7: $<< synchronise(A) >>$
8: **return** A

Clearly the communication has been drastically reduced in comparison to the fine grained approach. While the load balancing issue is still there and cannot be resolved by adapting the software alone. In the sketched case study, the coarse grained approach would favour the fine grained approach. This is mainly caused due to the fact that all the operations performed are independent over the different matrix elements.

Chapter 3

Parallel processors

Parallel processors refers to a wide variety of hardware that supports parallel computing. To apply the parallel computing concept not, only the algorithms have to be converted. The underlying hardware has to have the ability to cope with this as well. With the current development in CPU technology the MIMD architecture is becoming more and more common. Nowadays desktop computers come standard with a dual-core or quad-core CPU. When applying parallel technologies, optimal performance is a crucial factor. Choosing the right hardware architecture that matches the problem is thus important. As discussed earlier the multi-core CPU devotes a lot of die area to control logic. For common scientific calculations this makes them less attractive. Usually it is necessary to execute the same operations over and over again. Exactly the phenomenon SIMD architecture is specialised in. GPUs, which belong to the class of processors that is also referred to as massively parallel processors, are an example of SIMD architecture based processors. With the advances in GPU technology they have become popular for general purpose computing. They are relatively inexpensive in comparison to e.g. a supercomputer. Making them a potential candidate to solve a diversity of problems. These factors make GPUs a promising technology for solving the WFS reconstruction algorithm in time.

3-1 Development

Since several years GPUs have become of interest for general purpose computing. The processors graphics heritage reveals some of its strength and weaknesses. The GPU originates from the early 1980s, when three-dimensional graphics pipeline hardware was developed for specialised platforms. These expensive hardware platforms evolved to graphics accelerators in the mid-to late 1990s allowing it to be used in personal computers. In this era the graphics hardware consisted of fixed-function pipelines which were configurable but still lacked the ability to be programmable. Another development in this decade was the increase of popularity of graphics application programming interface (API)s. APIs allow a programmer to use software or hardware functionality at a higher level, such that the programmer does not

have to know the details of the system he or she is using. Several APIs popped up during the 1990s, e.g. OpenGL® as an open standard and DirectX^{TM} which was developed by Microsoft. Starting in 2001 with the introduction of the NVIDIA GeForce 3 [10], the GPU development took the turn towards programmability. Slowly evolving to a more general purpose platform. The first scientists noted the potential to solve their computational intensive problems. At this phase the GPU was still completely intended for graphics processing which made it challenging to use them for other purposes. A new field of research arises with the focus on how to map scientific problems to fit in the graphics processing structure. This field is referred to as general-purpose computing on graphics processing units (GPGPU) [29]. Manufactures jumped into the trend by developing tools to acquire easy access to all the resources and further improved the hardware. Compute Unified Device Architecture (CUDA) is an example of a programming language developed by NVIDIA. Also under initiative of Apple, Open Computing Language (OpenCL) is developed which should be the first step towards standardisation in GPU tools. At this moment GPUs are almost true unified processors and their development is still continuing. Table 3-1 shows the properties of two modern high performance GPUs: the NVIDIA Tesla C2070 [28] and the AMD FireStream 9270 [1].

Property	**NVIDIA Tesla C2070**	**AMD FireStream 9270**
Cores	448	800
Peak Performance (Single precision)	1.03 TFLOP	1.2 TFLOP
Peak Performance (Double precision)	515 GFLOP	240 GFLOP
Memory	6GB GDDR5	2GB GDDR5
Memory Interface	384-bit @ 1.5 GHz	256-bit @ 850 MHz
Memory Bandwidth	144 GB/sec	108.8 GB/s
System Interface	PCIe x16 Gen2	PCIe x16 Gen 2

Table 3-1: Performance overview of the modern high performance GPUs: NVIDIA Tesla C2070 and the AMD FireStream 9270

3-2 Architecture

It is common that CPUs consist of a few cores, where the main part of the die area is devoted to control logic and cache. This allows the processor to handle a wide scale of problems, often in a sequential approach. Where the CPU is a generalist, the GPU is a specialist. What are the architectural differences that distinct them from each other? The approach taken for the multi-core CPU is to support a few heavy weight threads that are optimised for performance. To achieve this, supplementary hardware is added to minimise latency, e.g cache, branch predictors, instruction and data prefetchers. Latency arises mainly by accessing the global memory. Memory is several factors slower, depending on the type of memory than the processor itself, causing it to stall. Reducing stalls results in an improvement of performance. For the GPU architecture another approach was taken. There is chosen for lightweight threads with poor single-thread performance. Now by using the vast number of available threads to fill up the gaps, it hides latency and achieves good overall performance.

As noted earlier the GPU is based on the SIMD architecture. Slightly refining this, it can be reformulated as a single instruction multiple thread (SIMT) architecture. The treads are divided by the GPU in groups, such a group of threads is called a warp (Figure 3-2). Exactly the same instruction is performed for each thread in a warp. When a stall in a single thread occurs, the complete warp is temporally replaced by another warp available at the pipeline. To maximise performance, it is thus essential that the number of threads exceeds the number of available cores with several factors.

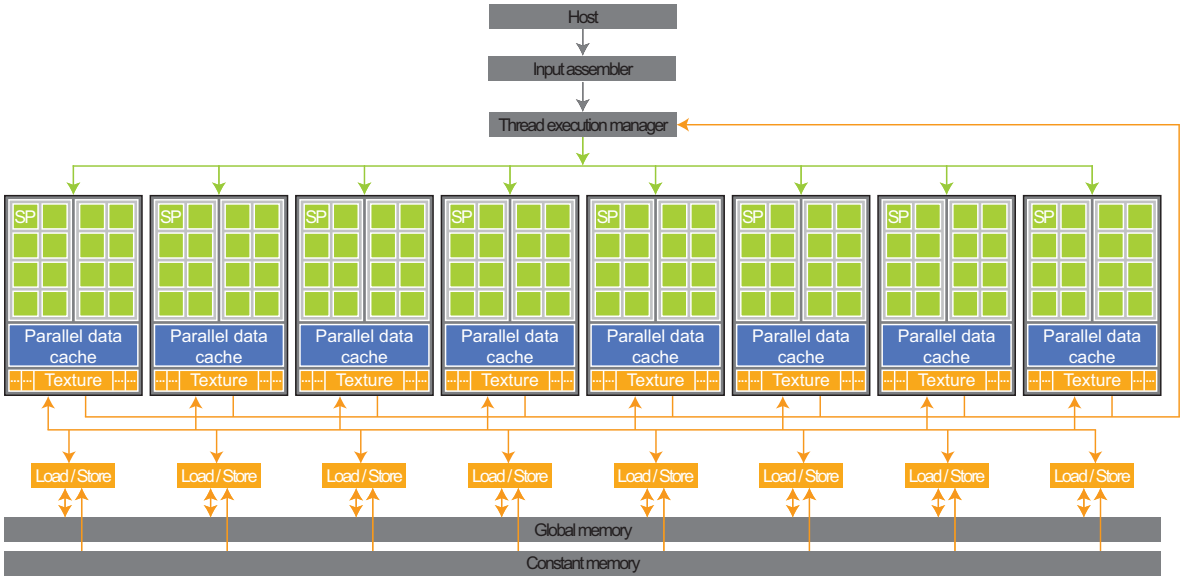


Figure 3-1: Architecture overview of GPU [30]

Like addressed before the performance of the memory affects the overall performance. Usually the principle applies that, when the dimension of the memory is increased also the latency increases. Various aspects play a role here. The smaller the memory, the closer it can be place by the processor and the faster it can be addressed. Second is the technology used to produce the memory that plays a role. For small memories like registers it is still cost effective to use expensive but fast memory types. While for the global memory each component per memory bit counts. Reason enough to design the memory architecture with a few levels with various types of memory to be able to supply programmers with fast and sufficient memory. The GPU uses such a memory model as well (Figure 3-1). Close to the processing units we find the fastest memory in the form of registers. Slightly slower is the shared memory which can be seen as a small cache that is shared by a group of cores. At an even higher level is the global and constant memory. These memories are shared by all the cores of the GPU and are by far the slowest memories. In designing parallel programs it is crucial to keep the data as local as possible. Accessing the global memory will give a severe load on the bandwidth of the memory. Observe that when a run to the global memory is performed, all threads in a warp do this at exactly the same moment [9]. Consequently resulting in a higher latency compared to when a single processor access the memory alone.
Besides the physical architecture also some knowledge of a programming model will help understanding the behaviour of the GPU. Figure 3-2 presents the programming model. At the host a kernel defines some parallel functionality. On the GPU side a kernel resembles a grid. Kernels are always executed sequential, making it impossible that on a GPU two

separate grids exists at the same moment. The gird is subdivided into blocks. Again each block is partitioned into threads. Single blocks in a grid operates completely independently from each other, in contrast to the threads in a block. The shared memory allows the threads in a block to communicate and can cooperatively work on a subproblem. All threads and blocks are uniquely identified by an index. The index can be used to select the different data, creating uniqueness. As denoted earlier, it is for the hardware mapping required to create groups existing out of 32 threads. The threads are selected from a single block forming groups called warps. Warps are scheduled to be executed on the GPU.

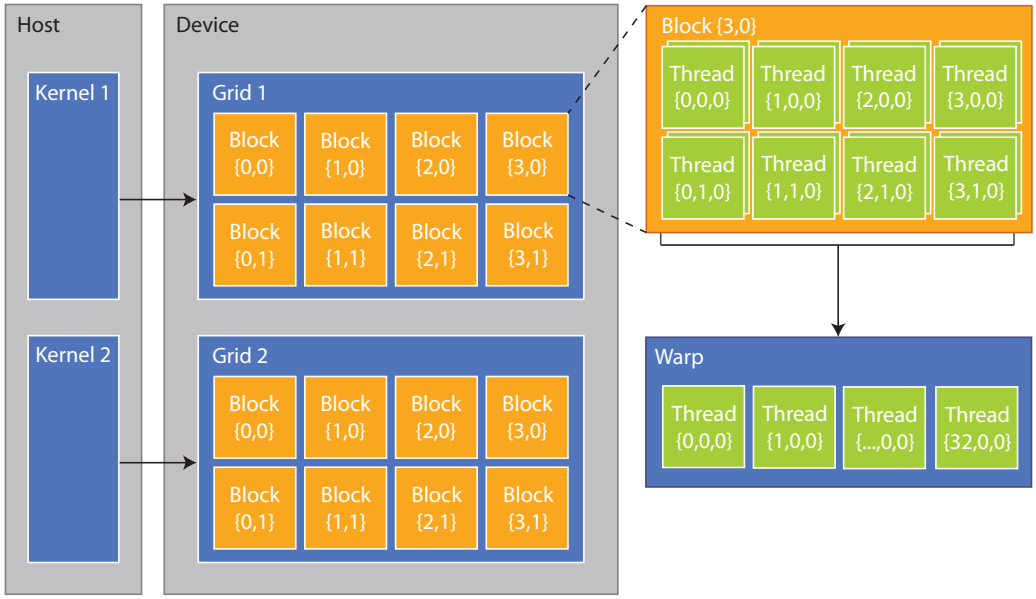


Figure 3-2: Programming model of a the GPU programming language CUDA

3-3 Applicability

As denoted, the development of the GPU, with respect to general purpose applications, started fairly recently. Using new technologies always brings additional risks. However they have also huge potential as well. Both have the same origin. The technology is still in its infancy. Researcher and manufactures are currently doing research and development to develop the GPU itself, tools supporting the development and documentation. All based on advancing insights and user feedback [30]. In the beginning this results often in impressive improvements in reasonable short periods of time. When a technology is already well established these improvements are followed up in a much slower rate. All this applies also to the GPU technology: profilers and debuggers are still primitive. Programming languages and compilers are in the first, or second stage of release. The hardware itself is adapted with each new generation, to better fit the needs of general purpose computing, with for example improved double-precision support [28, 1]. Stepping in at this moment will probably not give the highest performance in comparison to other technologies. Yet it can mean that you will be a step ahead in the future.
The GPU can not run on its own, it needs a CPU to be controlled by. In practise this

means when a CPU is executing a (sequential) program and encounters a parallel section the GPU is requested to solve it. All the data required to solve the parallel section first needs to be transferred to the GPU memory. Afterwards the GPU is able to execute the parallel statements and will, when finished, copy the result back from the GPU memory to the memory addressable by the CPU. An example of this process is shown in listing 3.1 which is based on a matrix multiplication example defined in [20, p. 50], the kernel code is sited in the appendix. The impact hereof is that each so called kernel start-up takes a non ignorable time and can give a severe load on the host. The solution lies in parallizing entire sections of the algorithm preventing kernel start-ups. As discussed in the parallel computing chapter this can be problematic. Also the SIMT architecture structure involves that each thread of kernel performs exactly the same. These combined factors can have vast influence on the performance, causing the fact that the GPU approach will greatly benefit of an appropriately chosen algorithm.

```
1  void MatrixMultiplication(float* M, float* N, float* P, int Width){
2    int size = Width * Width * sizeof(float);
3    float* Md, Nd, Pd;
4
5    //Transfer M and N to device memory
6    cudaMalloc((void**) &Md, size);
7    cudaMemcpy(Md, M, size, cudaMemcpyHostToDevice);
8    cudaMalloc((void**) &Nd, size);
9    cudaMemcpy(Nd, N, size, cudaMemcpyHostToDevice);
10
11   //Allocate P on the device
12   cudaMalloc((void**) &Pd, size);
13
14   //kernel invocation code
15   dim3 dimBlock(Width, Width);
16   dim3 dimGrid(1,1);
17
18   MatrixMulKernel<<<dimGrid, dimBlock>>>(Md, Nd, Pd, Width);
19
20   //Transfer P from device host and free resources
21   cudaMemcpy(P, Pd, size, cudaMemcpyDeviceToHost);
22   cudaFree(Md); cudaFree(Nd); cudaFree(Pd);
23 }
```

Listing 3.1: Example of a kernel startup process

The memory model determines that it is advantageous to keep data as local as possible. However, the size of these local memories is limited and not the only restricting factor for the amount of memory available to each thread. A kernel can exist out of more threads than the GPU cores contains. In combination with the memory this means that it has to share the resources over the threads. Resulting in the fact that amplifying the number of threads diminishes the number of registers available to each thread. Balancing the number of threads in such a way that latency can be completely hidden, but sufficient local memory is available. This is part of the design parameters that should be determined when converting the algorithm.
CUDA and OpenCL are both extensions to the programming language C. The difference

between them is that CUDA is specific designed for the NVIDIA GPUs. Where OpenCL is vendor independent. A major advantage of CUDA is that it can use the full extend of the GPU. For OpenCL compromises were made, to support all the different vendor specific implementations, which resulted in a slight negative effect on the performance. For both applies that the integration with an existing programming language introduces flexibility and low-learning curves. Libraries are already widely available and there is plenty of documentation for general C. Creating a relatively easy platform to program in.

3-3-1 Overview

Consequences to the algorithm design which are imposed when using GPU technology.

- Every process is initialised and controlled by the host. Copying data to and from the GPU is a required step at each process. Resulting in significant latencies. Using the coarse grained approached in parallelizing will reduce the number of start-up procedures.
- The GPU performs best with straight-forward arithmetic, due to its small control units. Meaning that every thread executes exactly the same code. So preferably avoiding branches especially those that diverge. Recursion is even not feasible.
- Global memory access is expensive and when possible it should be avoided. This can be achieved by either using the available local memory (shared memory and registers) or it might even be justified to do some additional arithmetic. On a higher-level it is important to recognise this in the design stage to try to lower the data dependencies and always take a coarse grained approach.
- The GPU performance benefits from several factors of more threads then available processing units to be able to hide latency. So when feasible it is preferable to split the algorithm up in a higher number of threads. Note that is should not be at the expense of memory access.
- There is a fixed amount of local memory and to each thread subsection of memory is assigned. The size of the local memory is dependent on the number of threads.

Chapter 4

Reconfigurable Computing

The technologies which were dominating the world of computer science and electronics, could be classified in either the category software or hardware. Since two decades a new category emerged, reconfigurable devices became available diminishing the gap between the hardware and software category. The process of optimally exploiting a reconfigurable device is referred to, as reconfigurable computing. Each of the categories; software, hardware and reconfigurable computing are covered by diverse technologies. Respectively (micro)processors, application specific integrated circuits (ASIC)s and field programmable gate array (FPGA)s are examples of these technologies. Hardware is completely directed towards a single application. It provides a highly optimised but permanently configured solution. In contrast software characterised by its flexibility allows for a wide range of applications. The compromise is that software solutions are several orders of magnitude worse with relation to power consumption, spatial efficiency and performance. A reconfigurable device tries to get the best out of both worlds. The fact holds that for each technology making compromises is an element of the design process. For example, FPGAs are reprogrammable establishing flexibility, however due to its fine grained level, programming is more demanding compared to a microprocessor. The versatile character of the FPGA implicates that it could be capable of solving the WFS reconstruction algorithm.

4-1 Development

Gerald Estrin introduced the concept of reconfigurable computing in his paper [7, 8] during the 1960s. He suggested a hybrid design containing a standard processor augmented with configurable hardware which could be reprogrammed to fulfil a specific task. The idea was slightly ahead in time. At this stage the combined technology of microprocessors and ASICs were sufficient to provide in all the needs. The significance was not recognised and reconfigurable computing lost interest. In the 1980s the concept revived, research in both the industry and the academic world increased tremendously, leading to the first commercially viable FPGA in 1985. The first FPGA was designed by the co-founders of Xilinx which is

together with Altera one of the leading manufacturers at the moment. The newly developed FPGA was based on two existing technologies; programmable read-only memory (PROM)s and programmable logic devises (PLD)s. In the following decade the FPGA technology exploded. The sophistication and performance improvement during the 1990s formed a versatile device. Causing a increase in the demand from the industry. While the first generations where merely based on logic elements referred to as a fine grained design, later generations were augmented with coarse grained modules like multipliers. Multipliers constructed from programmable logic were relatively slow and took reasonable amount of space to implement. Since designs frequently use multipliers they were added as special function units. Modern FPGAs can be equipped with a mixture of different coarse grained modules, like digital signal processor (DSP)s, analog-to-digital converter (ADC)s or digital-to-analog converter (DAC)s. Providing the possibility to construct complete single chip solutions. Properties from two FPGAs of high-end FPGA manufacturers are depicted to show the current status (Table 4-1). The devices under evaluation are the Xilinx Virtex-6 SXT (XC6VSX475T) [37] and Altera Stratix V GS (5SGSB8) [3].

Property	**Xilinx Virtex-6 SXT**	**Altera Stratix V GS**
Logic	476,160 Logic Cells	706,000 Logic Elements
Slices	74,400 Slices	274,000 (ALMs)
Multipliers (DSP)	2,016 (25 x 18)	3,510 (18 x 18)
Embedded Memory	37Mb	34Mb
Memory Blocks	2,128 x 18Kb or 1,064 x 36Kb	1,755 x M20K

Table 4-1: Property overview from the FPGAs; Xilinx Virtex-6 SXT and the Altera Stratix V GS

4-2 Architecture

Understanding the architecture of a device is an essential aspect in effective designing and implementing systems. As reconfigurable computing does extend to a broad range of hardware only the FPGA will be considered. Since the FPGA is the most common reconfigurable device. Also each manufacture has its own variations in implementation which is not relevant at this level hence the emphasis will lie on a typical device. The resources describing an FPGA can be divided in logic, interconnect, memory and special function units, where logic and interconnect are off primary interest. Logic is classified as the components that supports the arithmetic operations and logical functions, whereas the interconnect is considered to take care of the data transportation between blocks.
The idea behind FPGAs is based at the assumption that every problem can be broke down to a set off Boolean equations. By using truth tables in the form of look-up table (LUT)s, the Boolean equation can be expressed. These fundamental properties of digital logic form the basic building blocks of an FPGA. To view this from an integrated circuit (IC) perspective, a LUT can be constructed out of a N-bit memory and a N:1 multiplexer. The inputs of a logic block form the select bits of the multiplexer. Where the multiplexer selects out of the table the corresponding output. As LUTs are the smallest computational resources their design is crucial to the success of a FPGA. The size of an LUT is defined based on the number of inputs and plays a decisive role. Choosing for relative large LUTs allows for more complex logic to be implemented in a single LUT at the cost of slower multiplexers. On the other hand

smaller LUTs cause less overhead. Yet they pay the price when multiple LUTs are required for complex logic, increasing wiring-delay between blocks. LUTs based on four inputs are at this moment considered to be the most effective [17, p. 5]. To achieve a sufficient level of functionality the FPGA should be able to maintain a sense of state. The answer is to add a memory element to the basic building block. A common choice would be to use a D flip-flop. Figure 4-1 shows a schematic view of the derived layout which has a fair resemblance to the reality as well.

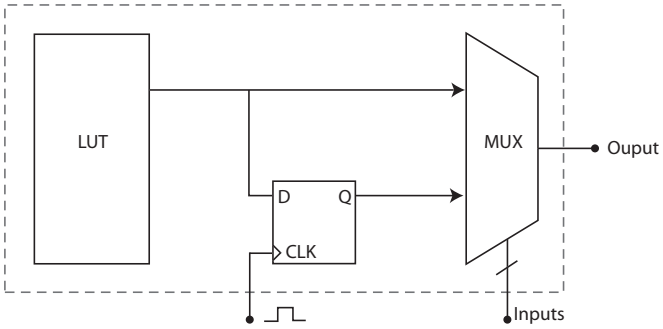


Figure 4-1: Simple representation of a logic block

Without communication, produced results are useless, interconnect refers to the communication back-bone of the FPGA. Nearest-neighbour is a simple structure used to communicate. Logic blocks are only connected to their immediate neighbours in all four directions. An implication of the structure appears when data has to travel to the other end of the FPGA. It has to pass all logic blocks, resulting in a linear scaling between distance and time. Bypassing the logic block would benefit the connectivity. When the design is segmented, an island-style architecture is created (Figure 4-2). Communication is established by introducing connectivity blocks and switch boxes while separating logic blocks from each other. Now the ability is created for each logic block to make a direct link to any other logic block inside the FPGA. As anticipated, segmented interconnect takes up a major fraction of the die space. Diminishing the interconnections would be favourable, however a design must be still routable. Vendors cannot predict the exact purpose where their FPGAs will be applied . Rent's rule [22] defines a rule of thumb that prescribes a sufficient amount of interconnect such that an average application can be routed.

$$N_{io} = KB^r \tag{4-1}$$

Where N_{io} is the number of input/output pins or the number of external signal connections to a block in the case of an FPGA, K is a constant defining the average number of interconnect per block, r is called Rent's exponent for which $0.57 \leq r \leq 0.75$ holds and B is the number of logic gates in the block.
FPGAs have many advantages, however in comparison to ASICs, they still use up to 18 times more area, draw seven times more power and are around three time slower [21]. Besides that certain functionality (e.g ADC and DAC) is not suitable to be established in the FPGA logic structure. Reasons to augment the FPGA with special purpose modules to overcome these limitations. Memory and multipliers are the most common modules as, they are required in almost every design. A trend shown in the latest designs is that increasingly more complex special purpose modules are added directed towards a specific application area. Notice that ASIC are extremely expensive for low volumes, hence not a suitable option.

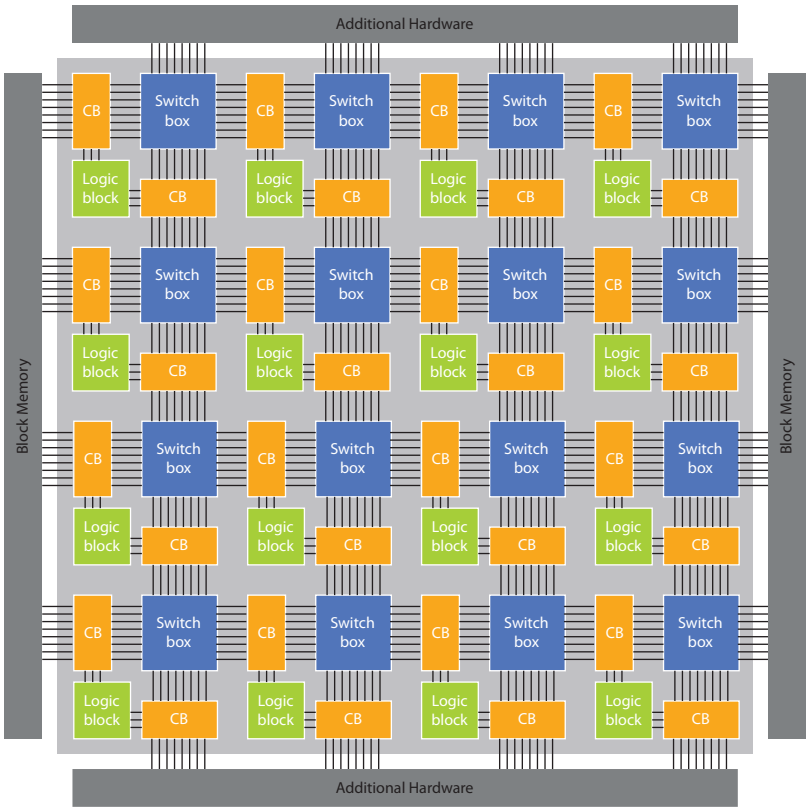


Figure 4-2: Architecture of FPGA

4-3 Applicability

The fine grained architecture allows virtually any application that can be transformed to Boolean equations to fit in the FPGA structure. Programming or describing, as it is actually called is usually accomplished via a hardware description language (HDL). Such a language is reasonably low level. Compared to a processor language the level is equivalent to assembly or even lower. The reconfigurability is associated with flexibility, however the rather comprehensive describing model suggests the opposite. Regarding these aspects, the technology places itself, with respect to flexibility, between software and hardware category.
FPGAs do not confine themselves to a specific domain. The technology benefits largely, if the application under consideration, can be mapped to a parallel version. However, FPGAs can cope with sequential sections as well. By producing special purpose modules, the specialisation factor affects the speed-up in a positive manner resulting in a faster version as could be achieved on a general purpose microprocessor. The sequential approach still allows for spatial implementations by running different special purpose modules simultaneous. The versatile character of the FPGA reduces the number of constraints imposed on the algorithm.
Almost any scientific application or algorithm uses matrix multiplications. The elementary building block of a matrix multiplication is a multiply-add operation. The question is how well suited is an FPGA to perform such operations also referred to as FLOP. Since the hardware multipliers inside modern FPGAs, as denoted above, do not fulfil the 32-bit requirement for single precision FLOPs let alone for double precision. Manufactures offer intellectual

property (IP) blocks (comparable to libraries in C) to support a variety of FLOPs. These blocks use several multipliers and logic units for a single FLOP unit. Using the IP block manual and device datasheets from the vendors [36, 37] we are able to determine the theoretical floating point performance [2, 32]. Taking the Xilinx Virtex 6 FPGA as an example. Table 4-1 gives us the necessary information about the FPGA, where Table 4-2 presents the properties of the floating point IP block.

Operation	DSP	Slices	LUTs	FFs	Maximum Frequency (MHz)
Multiply (SP)	3	110	107	114	429
Add (SP)	2	295	287	337	380
Multiply (DP)	11	357	328	497	429
Add (DP)	3	849	834	960	421

Table 4-2: Resource usage of floating point IP block when implemented in Xilinx Virtex-6 FPGA. Multiply and add operations in both single precision (SP) and double precision (DP) are presented.

Depending on the configuration, the theoretical performance can vary between about 306 GFLOP single precision and 121 GFLOP for double precision. Assumed is that a multiplier and an add operation always occur in pairs. Take into account that in these calculations, no space was reserved for control logic or other functionality. It is also common that the allowed maximal frequency drops when the size of the design increases. It is attractive to use these findings as comparison measure for other devices. Doing this neglects the FPGAs ability of being full application specific, whereas other platforms are directed toward a application area.

4-3-1 Overview

Consequences to the algorithm design which are imposed when using FPGA technology:

- Single precision and double precision floating point operations are not directly supported and requires IP blocks. For each block substantial resources are utilised, limiting the number of operations that can be performed parallel.
- Embedded memory is confined in a FPGA. Embedded memory can be used for e.g. registers. Algorithm designers should considers this limitation. Still there is the possibility to augment a FPGA with memory, however usually global memory is several factors slower.
- Depending on the type of function some are more expensive in relation to the available resources than others. A multiply operation requires more resources in contrast to an add operation. Efficient implementations involves algorithm design which considers the operation expense.
- A FPGA allows to for tailor made implementations on a hardware level. Reducing overhead and incompatibility issues. This property is what makes FPGAs a suitable platform for a wide variety of applications.

Chapter 5

Adaptive Optics Algorithms

A critical performance component of the adaptive optics (AO) system is the control algorithm and for existing AO systems this is a well established field. However till now there was no real need for computational efficient implementations. This has changed with the increase of interest in ELT's. The wave front reconstructor which is an essential part of the control algorithm where slopes (s_k) are used to estimate wave front phases ($\hat{\phi}_k$) based on Eq. (1-6). We have seen in the introduction chapter we need to optimise the wavefront reconstruction process, as it contributes significant to the computational efficiency of the control algorithm. Often the choice in algorithm lies in a tradeoff between performance and accuracy. There are already several suggestions for algorithms that are optimised and try to get the best out of both worlds, e.g. the FFT solution proposed by Poyneer et al. [31] is relatively efficient however it compromises in accuracy. On the other side of the spectrum are the model based predictors which are accuracy wise optimal [6, 13]. Recently also a new approach is suggested by Thiébaut and Tallon [33] which uses fractal iterative method (FRiM) as a augmentation to the minimum variance method. To make a fair judgement three algorithms have been selected and analysed for their performance and accuracy. The chosen algorithms are expected to be spread throughout the performance versus accuracy spectrum. Each of the algorithms are discussed in a separate section, partitioned by three subsections. The subsection General Info describes the basis and the origin of the algorithm. In Algorithmic Structure, a summary is given on the operation of the algorithm. Finally an analyse considering the accuracy, performance and suitability for parallel implementation is described.

5-1 Fast Fourier transform reconstruction

5-1-1 General Info

Freischlad and Koliopoulus suggested to use Fourier transformations as a wavefront reconstruction method [14]. In Poyneer et al. [31] the authors investigate and demonstrate the feasibility of the method for AO systems with at least 10.000 actuators. The motivation

behind using Fourier transformations is that certain operations are easier to perform in the frequency domain. In contrary stands the fact that now two transformations has to be performed. First from the time to the frequency domain and later back again. The use of Fourier transforms appears to have a second drawback, they are mainly designed to be fast and never proven to be optimal. subsequent the accuracy has to be verified to be sufficient for the specific application. Research performed by the authors of [24] and [25] show the results of experiments carried out on respectively a Virtex 4 FPGA and a Ge-Force 7800 GTX GPU. They claim that the suggested method of L.A. Poyneer can be solved on both platforms. When a problem size of 256x256 is considered, the FPGA needs around 2.0 ms to come to a solutions and for the GPU it takes 5.2 ms.

5-1-2 Algorithmic Structure

The method is primarily depending on the inverse spatial filter, complementary operations, like the Fourier transform, are obligated to allow the filter to be applied. Furthermore, to make the method applicable to a AO system of a telescope some pre-processing and post-processing steps has to be undertaken. In Figure 5-1 the complete structure of the algorithm is depicted. As the algorithm is specialised towards an specific sensor geometry each change in this geometry has effect on the algorithm. In the paper the Fried geometry is taken as foundation and some additional aspects of the Hudgin geometry are elucidated (in the figure shown with dotted lines).

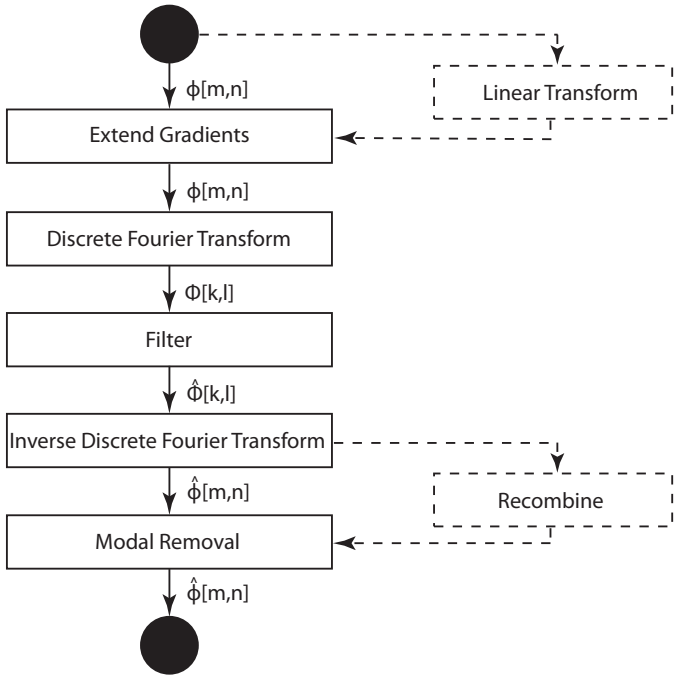


Figure 5-1: Structure FFT algorithm. Both the process steps of the Fried as the Hudgin geometry are depicted. The additional required by the Hudgin geometry are dashed.

The filter works under the assumption that the aperture is square. However in reality the aperture is always circular. To compensate for this Poyneer et al. [31] suggest two methods. The boundary method where the gradients on the edge of the aperture are used to calculate

the gradients that cross the edge. This is achieved by setting up a set of equations and solving the unknowns. The second one is the extension method, this method extends the outer gradient down, up, left and right. The still missing seam gradients are calculated by setting the row or column to zero. The boundary method as well as the extension method considers only the noiseless case. In reality noise will always be present and this can influence the suggested methods. E.g. the boundary method should use linear least squares instead to approximate a solution. The next step is the transformation to the frequency domain. The standard Fourier transform and inverse Fourier transform scales as $O(N^2)$ while a lower order would be desirable. Fast Fourier transform (FFT) methods, like the Cooley-Tukey method described in [23, p. 44] or the multidimensional approach [23, p. 149] achieves exactly that, but under the assumption that the problem size is a power of 2. The suggested FFT scales as $O(Nlog_2N)$ instead. In the frequency domain it is possible to apply the inverse filter. The filter consists out of calculating the gradients by determining the first differences between adjacent phase points, denoted by Eq. (1-4). Transformation to the frequency domain leaves us with equations 5-1.

$$S_x[k,l] = \Phi[k,l][exp(\tfrac{j2\pi k}{N}) - 1] \tag{5-1}$$
$$S_x[k,l] = \Phi[k,l][exp(\tfrac{j2\pi l}{N}) - 1] \tag{5-2}$$

The inverse filter can be deduced by applying linear least squares in the frequency domain Eq. (5-3).

$$\hat{\Phi}[k,l] = \begin{cases} 0, & k,l = 0 \\ \{[exp(-\frac{j2\pi k}{N}) - 1]S_x[k,l] + [exp(-\frac{j2\pi l}{N}) - 1]S_y[k,l]\} & \\ \times[4(sin^2\frac{\pi k}{N} + sin^2\frac{\pi l}{N})]^{-1}, & else \end{cases} \tag{5-3}$$

Transforming the result back to the time domain using a inverse Fourier transformation leaves only the modal removal to be applied. For certain modes it is beneficial to eliminate them from the reconstruction. Due to the fact that Fourier transform introduces significant errors into the estimates of particular modes, like waffle and piston. The modal removal process is done separately, however it can be performed efficient as is depicted in the appendix.

5-1-3 Analysis

A good approach when optimising any algorithm is to analyse to which part of the algorithm the greatest amount off time is devoted. By selecting this part of the algorithm as first candidate to be redesigned, will probably give you the highest gain. Each time an optimisation iteration is completed the previous described step should be repeated, based on the fact that the part of the algorithm selected before does not have to have the longest execution time anymore. The analysis only considers the FFT and filter process of the method described by L.A. Poyneer as these processes are rather demanding, the remaining steps can be found in the appendix and shall be discussed briefly.
The pseudo-code off the inverse filter process is depicted in Algorithm 4, studying the code reveals a reasonable amount of calculations. As the calculations are in the frequency domain,

the function variables are complex numbers. Consequently each complex addition or subtraction involves two floating point operation (FLOP)s. Complex multiplications have even a more severe impact with four regular multiplications and two regular additions they require six FLOPs. In the filter equations of the form $exp(\frac{j2\pi x}{N})$ are frequently used. These can be rewritten as $cos(\frac{2\pi x}{N}) + i.sin(\frac{2\pi x}{N})$ resulting in three computations. From this knowledge follows that for example line 3 of the algorithm shall require a total of 10 FLOPs. Setting the overall number, for a single iteration at 49 FLOPs. To make a fair assessment of the severity of the computation intensive filter step, the order of the filter should be determined. The algorithm contains a nested for-loop were the main loop requires N_x and the nested loop N_y iterations. Let's define $N = N_x N_y$, resulting in an order of $O(49N)$ despite the large constant it is assumable that the inverse filter step will not behave as bottleneck. Besides the knowledge that Fourier transformations always produce conjugated pairs allowing for simplifications in a further stage. Also, a closer observation shows us that the computations are completely independent allowing for an easy transformation to a parallel version.

Algorithm 4 $FilterFFT(\Phi)$

1: **for** $k = 0$ **to** N_x **do**
2: **for** $l = 0$ **to** N_y **do**
3: $S_x \leftarrow \Phi[k,l][\exp(\frac{j2\pi k}{N_x}) - 1]$
4: $S_y \leftarrow \Phi[k,l][\exp(\frac{j2\pi l}{N_y}) - 1]$
5: **if** $k == 0$ **and** $l == 0$ **then**
6: $\hat{\Phi}[k,l] \leftarrow 0$
7: **else**
8: $\hat{\Phi}[k,l] \leftarrow [[\exp(-\frac{j2\pi k}{N_x}) - 1]S_x + [\exp(-\frac{j2\pi l}{N_y}) - 1]S_y][4\sin^2(\frac{\pi k}{N_x}) + 4\sin^2(\frac{\pi l}{N_y})]^{-1}$
9: **end if**
10: $l \leftarrow l + 1$
11: **end for**
12: $k \leftarrow k + 1$
13: **end for**
14: **return** $\hat{\Phi}$

FFT referrers to a class of computational efficient (discrete) algorithms to perform Fourier transformations. Algorithm 5 is a deduction of pseudo-code of an FFT known as Cooley-Tukey [23]. The order of the suggested FFT is $O(Nlog_2N)$. Still it should be notified that there are again several complex computations involved. While $O(Nlog_2N)$ is a substantial progression, large N still requires a vast amount off effort or more important time. Parallel computation could resolve this issue if the FFT allows the conversion. Through the existence of operations that affect multiple indices, it is possible to conduct that data dependencies are present. As discussed before GPU technology is far more sensitive to parallel computing issues than FPGA technology. Already suggestions as for example by Moreland and Angel [27] are done to apply FFT on GPU technology. Their performance results are however comparable to a highly optimised CPU FFT.

Algorithm 5 $FFT(\phi, flag)$ and $IFFT(\Phi, flag)$

for $q = 1$ **to** t **do**
  $L \leftarrow 2^q$
  $r \leftarrow \frac{N_y}{L}$
  $L_* \leftarrow \frac{L}{2}$
  $\Phi \leftarrow \phi$
  $\omega_L^k \leftarrow \cos(\frac{2\pi k}{L}) - j\sin(\frac{2\pi k}{L})$
  $\Omega_L* \leftarrow diag(1, \omega_L, ..., \omega_L^{L*-1})$
  $B_L \leftarrow \begin{bmatrix} I_L* & \Omega_L* \\ I_L* & -\Omega_L* \end{bmatrix}$
  for $k_2 = 0$ **to** $r - 1$ **do**
    for $k_1 = 0$ **to** $r - 1$ **do**
      $\alpha_1 \leftarrow k_1 L_* : (k_1 + 1)L_* - 1$
      $\alpha_2 \leftarrow (k_1 + r)L_* : (k_1 + r + 1)L_* - 1$
      $\beta_1 \leftarrow k_2 L_* : (k_2 + 1)L_* - 1$
      $\beta_2 \leftarrow (k_2 + r)L_* : (k_2 + r + 1)L_* - 1$
      $rows \leftarrow k_1 L : (k_1 + 1)L - 1$
      $cols \leftarrow k_2 L : (k_2 + 1)L - 1$
      $\Phi(rows, cols) \leftarrow B_L \begin{bmatrix} \phi(\alpha_1, \beta_1) & \phi(\alpha_1, \beta_2)* \\ \phi(\alpha_2, \beta_1) & \phi(\alpha_2, \beta_2) \end{bmatrix} B_L^T$
    end for
  end for
end for
if flag == -1 **then**
  $\Phi \leftarrow \Phi / N^2$
end if
return Φ

Both the pre- and post-processing steps can be performed in $O(N)$, for a detailed derivation is referred to Poyneer et al. [31]. Overall is the computational effort acceptable for the wavefront reconstruction process, even for larger N. The accuracy is the second parameter determining the algorithms applicability. As for FFT methods accuracy does not come naturally. Accuracy has to be validated. In the paper of L.A. Poyneer a noise propagation analyse was performed. They compare the FFT to the VMM method, what can be concluded is that the FFT method becomes progressively worse when the number of actuators increase. Also low fill factors of the aperture in relation to the grid has negative influences, including the requirement for the FFT that the grid size has to be a power of 2 this can be tedious.

5-2 Sparse minimum variance reconstruction

5-2-1 General Info

Minimum variance reconstructors are currently favoured by astronomers, however the method is based on VMM and it consequently does not permit scaling to telescopes that belong to

the class of ELTs. Brent L. Ellerbroek suggest in his paper [6] that sparse matrix techniques can be applied to overcome this problem. Sparse technologies cannot directly be applied, the paper describes how to deal with these difficulties. A few times the suggestion is made to use approximations. For example the turbulence is modelled based on the Kolmogorov turbulence spectrum, where κ defines the spatial-frequency the suggestion is to approximate $\kappa^{-11/3}$ as κ^{-4}. These approximations are affecting the accuracy as is addressed in the paper. The gain achieved in contrast to conventional matrix inversion methods is substantial, when considering a system consisting out of 40,000 actuators, the FLOPs required for the computation of sparse minimum variance versus conventional matrix inversions is respectively 10^8 and $10^{14.5}$ operations.

5-2-2 Algorithmic Structure

The wavefront reconstruction process is a subsection of the complete control algorithm described by the paper. A minimum variance estimator is derived to solve Eq. (1-6). Resulting in the wavefront reconstruction described by Eq. (5-4). All matrixes are constructed such that they have a sparse structure. Where G is defined as the phase to WFS influence matrix, C_{xx}= $E[\phi_k \phi_k^T]$ is the phase covariance matrix and its inverse is approximated to assure sparsity. C_{nn}= $E[n_k n_k^T]$ is the noise covariance matrix, s are the measurements of the WFS stacked in a vector.

$$\underbrace{(G^T C_{nn}^{-1} G + C_{xx}^{-1})}_{A} u = \underbrace{G^T C_{nn}^{-1} s}_{b} \tag{5-4}$$

To solve the equation, an efficient approach would be to use conjugate gradients by observing that it fits the format $Ax = b$.

5-2-3 Analysis

The conjugate gradient method is converted to pseudo-code for easy analysis, the algorithm 6 is based on [16, p. 529]. For determining the order of an algorithm, loops are crucial. The conjugate gradients algorithm is build from a single while loop that has two criteria: $\sqrt{\rho_k} > \epsilon$ and $k < k_{max}$. The first criteria is satisfied when the result is sufficient close to the desirable value. The second is to prevent the algorithm from running infinitely and indicates that convergence is not guaranteed. The algorithm also recommends to supply an x_0 as initial condition. Since the convergence time depends on the initial x, choosing the right x is important. A possible solution can be to chose it as the previous solution, this is more likely to be close when the sample time gets smaller.

Algorithm 6 $ConjugateGradientsMV(A, b, x_0)$

1: $k \leftarrow 1$
2: $r \leftarrow b - Ax_0$
3: $p \leftarrow r$
4: $\rho_0 \leftarrow \|r\|_2^2$
5: **while** $\sqrt{\rho_k} > \epsilon$ **and** $k < k_{max}$ **do**
6: $w \leftarrow Ap$
7: $\alpha_k \leftarrow \frac{\rho_{k-1}}{p^T w}$
8: $x \leftarrow x + \alpha_k p$
9: $r \leftarrow r - \alpha_k w$
10: $\beta_k \leftarrow \frac{\rho k-1}{\rho k-2}$
11: $p \leftarrow r + \beta_k p$
12: $\rho_k \leftarrow \|r\|_2^2$
13: **end while**
14: **return** x

Inside the loop several computations are carried out, among them is a sparse matrix vector multiplication. An example algorithm to perform sparse matrix multiplications can be found in the appendix. With sparse matrix computation technologies, the effort depends also on a fill factor denote as f. The fill factor is the percentage of nonzero elements in the matrix. The fill factor does however not affect the order of a matrix computations, the order of a sparse matrix vector multiplication can be expressed as $O(2fN^2)$. Table 5-1 shows that summing the effort required (5-1) and using the knowledge that the loop is executed maximal k_{max} times, leaves us with $O(k_{max}(2 + 10N + 2fN^2))$ or simplified $O(k_{max}N^2)$.

Line Number	Description Computations	Order
6	matrix*vector (Sparse)	$2fN^2$
7	vector*vector and scalar divide	$1 + 2N$
8	scalar*vector and vector+vector	$2N$
9	scalar*vector and vector-vector	$2N$
10	scalar/scalar	1
11	scalar*vector and vector+vector	$2N$
12	vector*vector	$2N$
Total		$2 + 10N + 2fN^2$

Table 5-1: Analysis of the conjugate gradient algorithm

When evaluating the order of an algorithm the upper bound is always used as measurement. Directly revealing a bottleneck of the method. Depending on the circumstances the computation can take up to $O(k_{max}N^2)$, however generally speaking it will perform much better through the sparse matrix computations and due to the fact that k_{max} will not be reached. In a control loop the worse-case has to serve as leading choice for the control frequency so this property will be a major disadvantage unless there exists the ability to prove that in this specific application the worst-case does not exist. Applying parallel computing to conjugate gradient, is due to its dependency on results of previous iterations difficult. In the paper by Bolz et al. [4] a application of conjugate gradients on a GPU is described. To avoid the

data dependencies the auteur's used a fine grained approach. Consequently the performance limit is finally determined by the bandwidth of the GPU. As already denoted is the accuracy influenced by the approximations. The residual mean squared error is a measure related to accuracy which was used. In the evaluations Brent L. Ellerbroek has performed there are no significant deviations. It should however be denoted that he did not compare the conventional matrix inversion with the sparse minimum variance method for larger AO grid sizes.

5-3 Structured Kalman reconstruction

5-3-1 General Info

Rudolf E. Kalman developed a numerical method to estimate true values based on measurements corrupted by noise, referred to as the Kalman filter. A dynamical model describing the behaviour of the system is the basis of the Kalman filter. Under the assumptions that the model is linear and the measurements are corrupted by white noise, the Kalman filter can make an optimal prediction. In the paper by Fraanje et al. [13] the suggestion is to use a structured Kalman filter for the wavefront reconstruction process. To accomplish a structure their work is based on a frozen flow turbulence model. By the introduction of sequentially semi separable (SSS) matrices it is allowed to use the property that a multiplication with an arbitrary vector can be formatted in a series off subsystems. The gain in structuring the Kalman filter is the reduction of complexity at the cost of a slight loss in accuracy.

5-3-2 Algorithmic Structure

A dynamic model always stands at the bases of the Kalman filter. Consequently this approach requires that together with calculating the Kalman gain a model has to be determined. To calculate the kalman gain a computational heavy Ricatti equation has to be solved. Fortunately the update rate for the model is presumably rather slow, such that it can be performed offline. Placing the emphasis on the estimation and prediction of the wavefront phases (Figure 5-2).

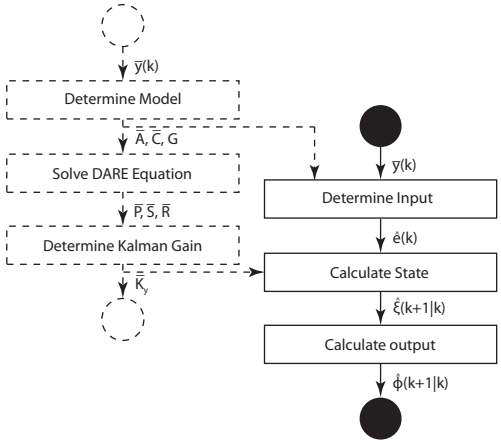


Figure 5-2: Wavefront reconstruction algorithm

Depending on the delay between the reading of the sensor and the control of the deformable mirror (DM), it is advantageous to predict several steps ahead. Given the disturbance model Eq. (1-2) and the measurements Eq. (1-6) a predictor can be deduced. Equations 5-5 - 5-7 describe a one-step ahead predictor.

$$\hat{e}(k) = \bar{y}(k) - \bar{C}^y \hat{\xi}(k|k-1) \tag{5-5}$$
$$\hat{\xi}(k+1|k) = \bar{A}\hat{\xi}(k|k-1) + \bar{K}_y \bar{e}(k) \tag{5-6}$$
$$\hat{\phi}(k+1|k) = \bar{C}\hat{\xi}(k+1|k) \tag{5-7}$$

As we are dealing with SSS matrices, equation (5-8) can be applied to compute for example $\hat{u}(k) = \bar{K}_y \hat{e}(k)$

$$\begin{bmatrix} v_{i+1}^m(k) \\ v_{i-1}^p(k) \\ u_i(k) \end{bmatrix} = \begin{bmatrix} K_i^{mm} & 0 & K_i^{me} \\ 0 & K_i^{pp} & K_i^{pe} \\ K_i^{um} & K_i^{up} & K_i^{ue} \end{bmatrix} \begin{bmatrix} v_i^m(k) \\ v_i^p(k) \\ \hat{e}_i(k) \end{bmatrix}, i = 1, ..., N \tag{5-8}$$

Hence it is possible to reconfigure the equation to form a series of subsystems as is depicted in Figure 5-3. The suggested structure is suitable for an distributed implementation.

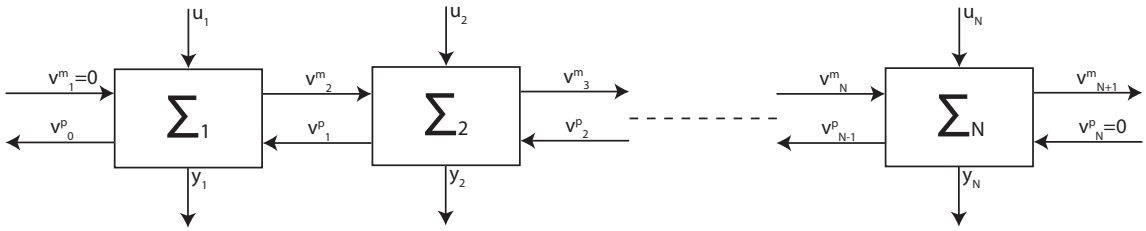


Figure 5-3: SSS matrix-vector multiplication as a series of subsystems

5-3-3 Analysis

Matrix vector multiplications utilising the matrices $\bar{A}$ and $\bar{C}$ can be implemented efficient by observing the block structure. Deducing the fact that it is sufficient to determine the matrix vector multiplication result of the block matrices containing information about the innovation model simplifies the computation. The remaining blocks are either filled by identity matrices or zeros, resulting in a simple projection of the vector. If N_m defines the order of the innovation model the complexity of $\bar{A}\hat{\xi}(k|k-1)$ and equation 5-7 are respectively $2N_m^2$ and $2N_yN_m$. Assume that the WFS resolution corresponds to the grid size and the maximal values for n_m and n_p are defined as $N_y/2$. For equations 5-5 and 5-6 the complexity is then respectively $4N_mN_y$ and $6NN_y + 6NN_m + 2N_m^2$. $6NN_y$ and $6NN_m$ determine the order, Both N_m and N_y are assumed to be at most $N^{1/2}$. Leaving us with an order of $O(12N^{3/2})$. Algorithm 7 shows how a SSS matrix multiplications can be implemented in a sequential fashion. Despite the distributed design there are dependencies, the intermediate values v^m and v^p are depended on a previous subsystem. So the system can be distributed but each subsystem has to wait till its neighbour supplies him with the variable values. The dependency prevents a straightforward

mapping to a parallel version. Note that the matrix vector multiplications related to $\bar{A}$ and $\bar{C}$ can be parallelized on a fine grained level like any other standard matrix multiplication.

Algorithm 7 $MultiplySSS(A, y(k))$

$n \leftarrow Size(v^m(k))$
$v_0^m(k) \leftarrow 0$
$v_n^p(k) \leftarrow 0$
for $i = 0$ **to** n **do**
  $v_{i+1}^m(k) \leftarrow A_i^{mm} v_i^m(k) + A_i^{me} y_i(k)$
  $v_{n-i-1}^p(k) \leftarrow A_{n-i}^{pp} v_{n-i}^p(k) + A_{n-i}^{pe} y_{n-i}(k)$
  $i \leftarrow i + 1$
end for

$n \leftarrow Rows(A)$
for $j = 0$ **to** n **do**
  $u_i(k) \leftarrow A_i^{um} v_i^m(k) + A_i^{up} v_i^p(k) + A_i^{ue} y_i(k)$
  $j \leftarrow j + 1$
end for
return $u(k)$

The Kalman filter is optimal for systems that are both linear and corrupted by Gaussian distributed noise. As the paper shows the Kalman filter is achieving Strehl ratio's close to one. During the redesign to achieve the structured matrices some assumptions were made. Despite the assumptions the impact on the accuracy was minimal, only with extremely low signal to noise ratio's there is a significant difference.

5-4 Overview

Table 5-2 gives an overview of the three algorithms and their suitability with respect to the ELT application, order, FPGA implementation and GPU implementation. For the ELT category the judgement criteria were the performance and accuracy. E.g. the FFT method scored unsatisfactory on accuracy which explains the "- -". The order is only based on the expected worst-case scenario hence the poor score for the sparse minimum variance method. The categories FPGA and GPU implementation considers aspects like risk, demonstrated feasibility, (expected) performance and implementation effort. Hardware trends are not included in the score so only based on current technology. The GPU scores low on each algorithm, which is mainly caused by the lack on coarse grained parallelism in the algorithms. An aspect where GPUs are highly dependent on.

Algorithm	ELT	Order	FPGA	GPU
FFT reconstructor	- -	-	++	-
sparse MV reconstructor	-/+	- -	+	-
Structured Kalman reconstructor	-/+	++	+	-

Table 5-2: An overview of the suitability wit respect to the categories; ELT, Order, FPGA and GPU.

Chapter 6

Findings

The objective off the research was to discover the issues that play a role in resolving the thesis; which algorithm in combination with an implementation technology would be suitable for solving the adaptive optics (AO) related problem of reconstructing the wavefront. Such that kilohertz (real-time) performance is feasible? All with respect to the development of extremely large telescope (ELT)s. To achieve these requirements the focus must lie on the scalability of the design such that it allows for at least 40.000 actuators. During the research several different technologies at different levels are examined on suitability. The technologies could be classified into two categories algorithms and hardware. In the class algorithms, three methods were addressed: fast Fourier transform (FFT), sparse minimum variance by conjugate gradients and a structured Kalman filter. For the hardware category, graphics processing unit (GPU)s and field programmable gate array (FPGA)s were considered. A vital discovery is that due to the requirements each technology encompasses, there are strong dependencies between the lower level hardware and higher level algorithm.

6-1 Conclusions

Parallel computing is already an essential technology for solving large scale scientific problems. In the future the application segment will evolve to an even wider scope. The majority of CPU manufacturers already altered their direction towards parallel computing, driven by physical limitations related to frequency scaling. Parallel computing is therefore a technology exceptionally suitable to satisfy the real-time requirements related to large scale wavefront reconstruction of AO systems.
GPUs and FPGAs are hardware platforms which support parallel computing. Both parallel devices are capable to cope with such a task. Each of the platforms have their advantages and disadvantages, e.g. where the FPGA is a safer choice in the sense that it is more developed, the GPU is more interesting for its potential. The properties of both technologies have their requirements on the application implementation. As a consequence the preference off the hardware technology depends primarily on the selected algorithm.

Concerning the algorithms, the method suggested by Poyneer et al. [31], using FFT is above all fast, while the accuracy is coming on the second place. Especially when larger grid sizes are considered the accuracy issue is playing a significant role, reducing the likelihood that the method can be suitable for ELT applications. In contrast the sparse minimum variance method designed by Ellerbroek [6] is accurate. However, despite the fact that the general case is probably significantly smaller, the worst case is in the order of $O(N^2)$. Augmenting the data dependency issue related to the conjugate gradient method makes this also a non ideal candidate. The sequentially semi separable (SSS) Kalman method suggested by Fraanje et al. [13] achieves decent results on both performance and accuracy. Hence, it could be a suitable method. When reflecting the algorithm on parallel computing aspects there arises an issue, the interconnection structure imposes several data dependencies introducing sequential behaviour. Also it only allows for a fine-grained parallelization approach. Making the FPGA the technology were the greatest benefit can be achieved. Still implementing this algorithm in FPGA technology is merely a patch than a optimal solution. To use parallel computing to its full extend, the algorithm should rather be adapted or an alternative algorithm should be considered. For example, if it would be possible to introduce delays in the communication channels between subsystems, parallelism would improve significantly.

6-2 Recommendations

In the analysis of the algorithms, possible bottlenecks were indicated in relation to parallel computing. However a more in-depth study can decide whether there exists a parallel solution for the used algorithms or it should be developed. Emphasising on optimality should be vital here, due to the fact that when an algorithm merely uses a subset of the available resources it will not be sufficient for the AO application under consideration. The suggested research is necessary to allow future successful implementations on devices like FPGAs and GPUs.
An alternative option would be to adopt another approach. The recent suggested algorithms take existing methods as staring point, to tackle the problem globally. Mapping methods are used to transform the sequential algorithms to the parallel computing domain, often resulting in dependency issues. To avoid dependencies a bottom-up approach could be helpful. First defining the requirement that the algorithm should at least be fully parallel. The method suggested by Rufus Fraanje utilises neighbour data under the assumption of frozen flow turbulence. Perhaps this is an indication that information of nearest-neighbours is sufficient to predict phase differences provided that the sampling frequency is adequate. An approach is suggested that when achievable could result in a distributed system with only very local dependencies. The approach works under the assumption that each segment can be seen as a loosely coupled system. Starting with developing a local dynamical model where the inputs are the WFS values of the nearest-neighbours and the output the phase difference of the segment under consideration. Subsequently by applying Kalman filtering the prediction of the phase difference on a single grid point is estimated. This process is performed for each segment in parallel completing the wavefront reconstruction. The computational effort to solve a local problem is expected to be low due to the minor size. Generating an additional advantage that the order of the problem will never exceed $O(N)$, allowing for maximal scalability.

6-3 Project Proposal

Since the parallel computing concepts imposes strong requirements on the algorithm it is sensible to take these requirements as starting point to attain a successful implementation. For the graduation project I would propose to start a study to a nearest-neighbour approach suggested in the recommendations section. Hence the thesis will become: Is it possible to design and implement a fully parallel wavefront reconstruction algorithm solely based on nearest neighbour information? To answer the thesis a phased approach is suggested as depicted in Figure 6-1.

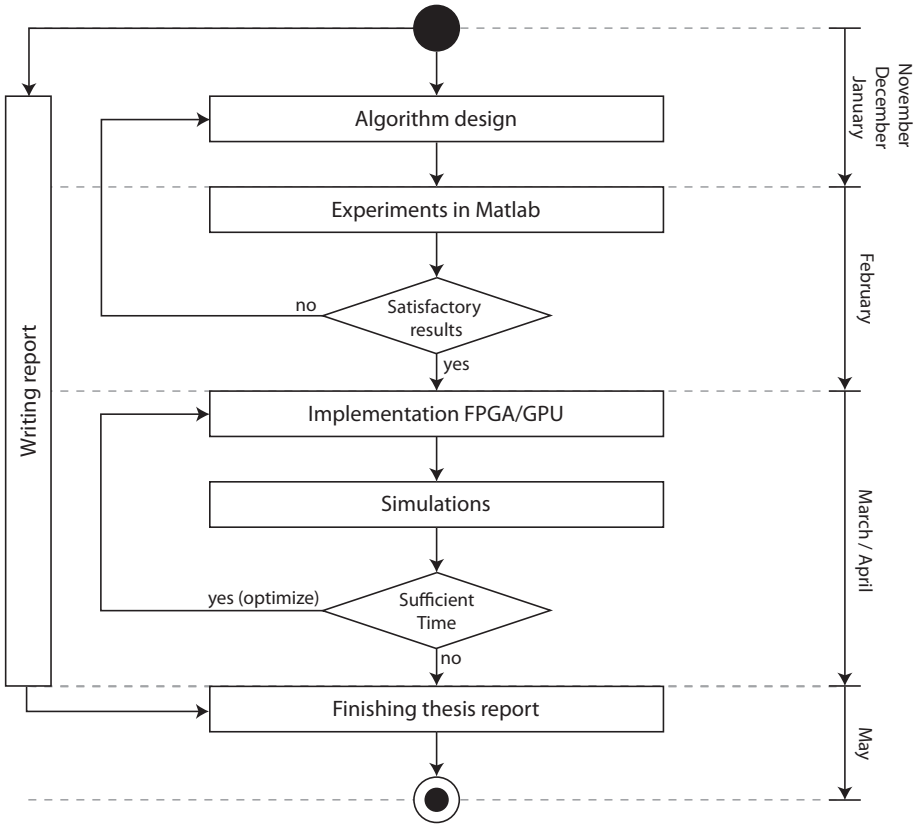


Figure 6-1: Graduation project proposal

As the suitability of hardware technology hugely depends on the algorithm structure choosing the hardware platform is still a open question. Since time is limited the most suitable platform will be implemented considering the developed algorithm.

Appendix A

Additional Algorithm details

Some elements of the algorithms were kept out of the main document for clarity. The pseudo-code of these elements can be found in this appendix chapter.

A-1 Fast Fourier transform reconstruction

Algorithm 8 $ExtendGradientsFFT(\phi)$

for $m = 0$ **to** N **do**
  $x \leftarrow FindBorder(n)$
  for $n = 0$ **to** x **do**
    $\phi[m, n] = \phi[m, x]$
    $\phi[n, m] = \phi[x, m]$
    $\phi[m, N - n] = \phi[m, N - x]$
    $\phi[N - n, m] = \phi[N - x, m]$
    $n \leftarrow n + 1$
  end for
  $m \leftarrow m + 1$
end for

for all seams **do**
  $s_x \leftarrow 0$
  $x \leftarrow AppretureBorder()$
  for $n = x$ **to** $N - x$ **do**
    $s_x \leftarrow s_x + \phi[n, m]$
  end for
  $\phi[m, n] \leftarrow -s_x$
end for
return ϕ

Algorithm 9 $ModalRemovalFFT(\hat{\phi})$

$\sum_{\phi v} \leftarrow 0$
$\sum_{vv} \leftarrow 0$
for $m = 0$ **to** N **do**
  for $n = 0$ **to** N **do**
    $\sum_{\phi v} \leftarrow \sum_{\phi v} + p\hat{h}i[m,n]v[m,n]$
    $\sum_{vv} \leftarrow \sum_{vv} + v[m,n]v[m,n]$
    $n \leftarrow n + 1$
  end for
  $m \leftarrow m + 1$
end for
$c_v \leftarrow \frac{\sum_{\phi v}}{\sum_{vv}}$
for $m = 0$ **to** N **do**
  for $n = 0$ **to** N **do**
    $\hat{\phi}_{final}[m,n] \leftarrow \hat{\phi}[m,n] - c_v v[m,n]$
    $n \leftarrow n + 1$
  end for
  $m \leftarrow m + 1$
end for
return $\hat{\phi}_{final}$

A-2 Sparse minimum variance reconstruction

Algorithm 10 $SparseMatrixMulMV(A, B)$

```
1: for i = 0 to NumberOfRows(A) do
2:   for j = 0 to NumberOfColumns(B) do
3:     row_i ← A.GetRow(i)
4:     col_j ← B.GetCol(j)
5:     sum ← 0

6:     while not row_i.IsEmpty() and not col_j.IsEmpty() do
7:       if row_i.item.index == col_j.item.index then
8:         sum ← sum + row.item.value * col.item.value
9:         row_i.item.GetNext()
10:        col_j.item.GetNext()
11:      else if row_i.item.index < col_j.item.index then
12:        row_i.item.GetNext()
13:      else
14:        col_j.item.GetNext()
15:      end if
16:      C[i, j] ← sum
17:    end while
18:    j ← j + 1
19:  end for
20:  i ← i + 1
21: end for
22: return C
```

A-3 Structured Kalman reconstruction

Algorithm 11 $VectorAdition(z(k), y(k))$

```
1: n ← Size(z(k))
2: for i = 0 to n do
3:   u(k) ← z_i(k) + y_i(k)
4:   i ← i + 1
5: end for
6: return u(k)
```

Bibliography

[1] Advanced Micro Devices, Inc. AMD FireStream^{TM} 9270 GPU Compute Accelerator . www.amd.com/stream, May 2010.

[2] Altera Corporation. White paper - designing and using fpgas for double-precision floating-point math. www.altera.com/literature/wp/wp-01028.pdf, August 2007.

[3] Altera Corporation. Stratix v device family overview. www.altera.com/literature/hb/stratix-v/stx5_51001.pdf, July 2010.

[4] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schroder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM transactions on graphics*, 22, 2003.

[5] Frederick P. Brooks. *The mythical man-month*. Addison Wesley Publishing, 1995.

[6] Brent L. Ellerbroek. Efficient computation of minimum-variance wave-front reconstructors with sparse matrix techniques. *Optical Society of America*, 19(9), September 2002.

[7] Gerald Estrin, editor. *Organization of Computer Systems - The Fixed Plus Variable Structure Computer*, New York, 1960. Western Joint Computer Conference, Proc. Western Joint Computer Conf.

[8] Gerald Estrin. Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. *IEEE Annals of the History of Computing*, 24(4), October 2002.

[9] Kayvon Fatahalian and Mike Houston. A closer look at GPUs. *communications of the acm*, 51(10), oktober 2008.

[10] Kayvon Fatahalian and Mike Houston. NVIDIA tesla: a unified graphics and computing architecture. *Hot Chips*, April 2008.

[11] Enrico Fedrigo and Robert Donaldson. Sparta roadmap and future challenges. *Adaptive Optics Systems II*, 7736(1):77364O, 2010.

[12] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on computers*, 1972.

[13] Rufus Fraanje, Justin Rice, Michel Verhaegen, and Niek Doelman. Fast reconstruction and prediction of frozen flow turbulence based on structured kalman filtering. *Optical Society of America*, 2010.

[14] Klaus R. Freischlad and Chris L. Koliopoulos. Modal estimation of a wave front from difference measurements using the discrete fourier transform. *Optical Society of America A*, 3(11), November 1986.

[15] Gene Golub and James M. Ortega. *Scientific Computing An Introduction with Parallel Computing*. Academic Press inc, London, United Kingdom, 1993.

[16] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[17] Scott Hauck and Andre DeHon. *Reconfigurable Computing*. Morgan Kaufmann Publishers, Burlington, 2008.

[18] John L. Hennessy and David A. Patterson. *Computer Architecture; A Quantitative Approach*. Morgan Kaufmann, 4th edition, 2007.

[19] G.J. Hovey, R. Conan, F. Gamache, G. Herriot, Z. Ljusic, D. Quinn, M. Smith, J.P. Veran, and H. Zhang. An fpga based computing platform for adaptive optics control. *1st AO4ELT conference*, 2010.

[20] David Kirk and Wei-Mei Hwu. *Programming Massively Parallel Processors*. Elsevier Science and Technology, 2010.

[21] Ian Kuon and Jonathan Rose. Measuring the Gap between FPGAs and ASICs. *FPGA*, 06, February 2006.

[22] Bernard S. Landman and Roy L. Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *IEEE transactions on computers*, c-20(12), December 1971.

[23] Charles Van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM, Philadelphia, 1992.

[24] Eduardo Magdaleno, Manuel Rodríguez, José Manuel Rodríguez-Ramos, and Alejandro Ayala. Modal fourier wavefront reconstruction using fpga technology. *Micro and Nanosystems*, 1(1), 2009.

[25] José G. Marichal-Hernández, José M. Rodríguez-Ramos, and Fernando Rosa. Modal fourier wavefront reconstruction using graphics processing units. *Journal of Electronic Imaging*, 16(2), June 2007.

[26] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 1965.

[27] Kenneth Moreland and Edward Angel. The FFT on a GPU. *Graphics Hardware*, 2003.

[28] NVIDIA Corporation. NVIDIA Tesla C2050/C2070 Datasheet. www.nvidia.com/tesla, July 2010.

[29] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics forum*, 26(1), 2007.

[30] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5), May 2008.

[31] Lisa A. Poyneer, Donald T. Gavel, and James M. Brase. Fast wave-front reconstruction in large adaptive optics systems with use of the fourier transform. *Optical Society of America*, 19(10), October 2002.

[32] Dave Strenski. Fpga floating-point performance - a paper and pencil evaluation. www.hpcwire.com/hpc/1195762.html, January 2007.

[33] Eric Thiébaut and Michel Tallon. Fast minimum variance wavefront reconstruction for extremely large telescopes. *Optical Society of America*, 25(5), May 2010.

[34] Tuan N. Truong, Antonin H. Bouchez, Richard G. Dekany, Stephen R. Guiwits, Jennifer E. Roberts, and Mitchell Troy. Real-time wavefront control for the palm-3000 high order adaptive optics system. *Proceedings of the SPIE*, 7015, 2008.

[35] R. N. Wilson. *Reflecting Telescope Optics I.* Springer-Verlag Berlin Heidelberg, Germany, 1996.

[36] Xilinx Inc. Floating-point operator v5.0 - product specification. www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf, June 2009.

[37] Xilinx Inc. Virtex-6 family overview - advance product specification. www.xilinx.com/support/documentation/data_sheets/ds150.pdf, January 2010.

Glossary

List of Acronyms

ADC	analog-to-digital converter
AO	adaptive optics
API	application programming interface
ASIC	application specific integrated circuits
CPU	central processing unit
CUDA	Compute Unified Device Architecture
DAC	digital-to-analog converter
DSP	digital signal processor
DM	deformable mirror
E-ELT	European extremely large telescope
ELT	extremely large telescope
ESO	European Southern Observatory
FF	flip-flop
FFT	fast Fourier transform
FLOP	floating point operation
FPGA	field programmable gate array
FRiM	fractal iterative method
HDL	hardware description language
GPGPU	general-purpose computing on graphics processing units

GPU	graphics processing unit
IC	integrated circuit
IP	intellectual property
LUT	look-up table
MIMD	multiple instruction multiple data
OpenCL	Open Computing Language
PLD	programmable logic devises
PROM	programmable read-only memory
SIMD	single instruction multiple data
SIMT	single instruction multiple thread
SSS	sequentially semi separable
VMM	vector matrix multiply
WFS	wavefront sensor

List of Symbols

α	The fraction of code that is not parallelizable
λ	Wavelength of the light emitted by the object under observation
ϕ	Phase difference vector
θ	Angular resolution
B	Number of logic gates in the block
C	The capacitance switched per clock cycle
C_{nn}	Noise covariance matrix
C_{xx}	Phase covariance matrix
D	The diameter of the aperture
e	Gaussian distributed noise
E_{new}	Executing time for entire task using the enhancement when possible
E_{old}	Execution time for entire task without using the enhancement
F	The frequency at which the IC is running
G	The phase-to-WFS influence matrix
H	The DM influence matrix
K	Constant defining the average number of interconnect per block
k	Time index

N_{io}	The number of input/output pins or the number of external signal connections to a block
p	The number of processing units
r	Rent's exponent
S_p	Speedup factor related to the run time of the original code
$s_x[m,n]$	Measured phase gradients in the x direction
$s_y[m,n]$	Measured phase gradients in the y direction
u_k	Control inputs
V	The supply voltage
z^{-1}	Discrete shift operator