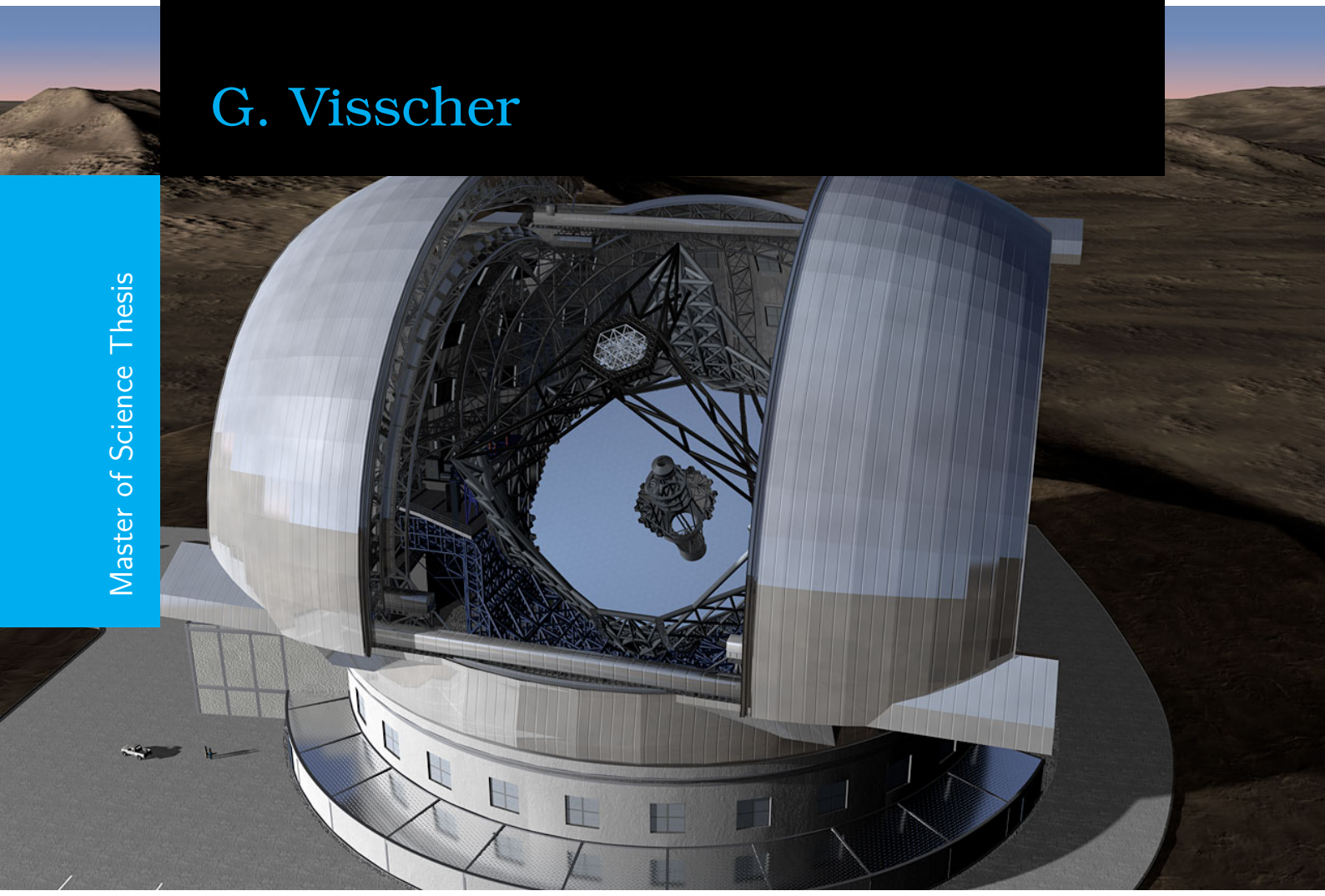
Large scale wavefront reconstruction for the next generation of Extremely Large Scale Telescopes

G. Visscher



Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Large scale wavefront reconstruction for the next generation of Extremely Large Scale Telescopes

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

G. Visscher

February 27, 2014

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The spatial resolution of an astronomical telescope is limited by either by the diffraction of light and the amount of aberrations caused in the atmosphere. To increase the diffraction limited resolution, the size of telescopes increases such as the to be build European Extremely Large Telescope (E-ELT). To fully benefit from the increased diffraction limited resolution, also the ability of adaptive optics to compensate for the wavefront aberrations should increase. This is for example done by increase the number of spatial wavefront measurements. This demands algorithms that are computationally efficient and highly parallelizable. In this thesis it is discussed whether the recently introduced Spline based ABeration REconstruction (SABRE) method can be parallelized. It is discussed that SABRE can be solved with the null-space method, resulting in a system of equations similar to the Poisson equation. The Conjugate Gradient (CG) method with different preconditioners, namely the Approximate INVerse (AINV), Algebraic Multigrid (AMG) and Incomplete Cholesky factorization (ICHOL), are used that make use of the sparsity of SABRE are discussed, their scalability and parallelism seem to be promising but the high amount of communication lowers the expectations. Experiments on the Graphics Processing Unit (GPU) give results, that matches with these expectations. Although there is plenty of parallelism, the sparse solvers are limited by the communication and therefore, for the grid sizes discussed in this thesis, slower than their counter ones on the Central Processing Unit (CPU). But there is lot of optimization possible and especially the CG with the AINV has a lot of potential. Further research is required.

Table of Contents

List of Figures

List of Tables

Preface and acknowledgments

The main goal of this research to improve the resolution of the astronomical telescopes sited on earth. Describing it more down-to-earth, the goal is to have a better picture of the space around us. More then a year I spend my time on this research, it was a time with up and downs. I would like to thank David Blom, my roommate, he had to listen every day about my progress of my research. I learned a lot this year and not only about wavefront reconstruction and simplex B-splines. I learned to be more thou rough and work more carefully. Here I need to thank Michel Verhaegen, he encouraged me and inspirated me in this process. I also would like to thank Elisabeth Brunner who gave me a lot of feedback on my report. At last I would like to thank my parents who financially supported me.

Delft, University of Technology G. Visscher
February 27, 2014

He moves mountains without their knowing it and overturns them in his anger.
He shakes the earth from its place and makes its pillars tremble.
He speaks to the sun and it does not shine; he seals off the light of the stars.
He alone stretches out the heavens and treads on the waves of the sea.
He is the Maker of the Bear and Orion, the Pleiades and the constellations of the south.
He performs wonders that cannot be fathomed, miracles that cannot be counted

JOB 9:5-10 (NIV)

.

Chapter 1

Introduction

With respect to the image quality of an astronomical telescope, there are two main issues that can deteriorate the quality of the image. The first one is that an ideal point source, if projected, leads to a diffifuse circular disk known as the airy disk. Which means that from a certain resolution, individual photons will start to excite multiple detector cells and it becomes difficult to relate photons to individual detector cells. The highest resolution one can then achieve is the the diffraction limited resolution. The other main problem, that can deteriorate image quality, is the aberration of the wavefront of the light waves. This is due to the changing refractive index because of turbulence in the atmosphere. With the increasing size of telescope, the size of the airy disc decreases, which improves the diffraction limited resolution. To fully benefit from the increased diffraction limited resolution it is a necessity to increase the number of measurements. The European Extremely Large Telescope (E-ELT), a to be build telescope by European European Southern Observatory (ESO), is an example of the increasing size of telescopes. The main mirror has a size of 39 meters in diameter. By actively compensating the wavefront using wavefront measurements and actuating deformable mirrors, it possible to correct for wavefront abberations. The increasing number of actuators and measurements, demands algorithms that are computationally efficient and highly parallelizable.
To correct for wavefront abberations, one needs to measure the wavefront. Currently it is not possible to measure the actual wavefront real-time, since it is a non-convex optimization problem [60]. Instead of that slopes or curvatures of the phase of the wavefront are measured. This results in a numerical integration problem to obtain the actual phase of the wavefront. This is the key problem, finding an algorithm that can do this numerical integration problem efficiently. Recently Visser and Verhaegen introduced a new wavefront reconstruction method called Spline based ABeration REconstruction (SABRE) [13]. They used Simplex B-splines to approximate the wavefront. The main advantage is that it does not require a grid. Also due to the local nature of SABRE it is also possible to calculate SABRE in a parallel way. In this thesis a new parallel algorithm will be presented that fully benefits from the local nature of the simplex B-splines.
The outline of this thesis is as follows. First the wavefront and the atmospheric turbulence is discussed. Next it will be discussed how the wavefront can be sensed. After that it will be shown how one can obtain the actual wavefront from the measured wavefront. It will be shown why Simplex B-splines are a good to choice to reconstruct the wavefront. Then some notions about parallel programming are given. It will be discussed, what the best way is to calculate

SABRE in parallel. Finally a conclusion is given, which also will introduce areas for further study.

Chapter 2

Introduction to wavefront sensing

In this chapter an introduction is given about wavefront sensing. First it is discussed how the wavefront can be described mathematically. After that different wavefront sensors will be introduced.

2-1 Wavefront and the atmospheric turbulence

What is the wavefront? A mathematical description is given and it is shown that it is possible with some assumptions to obtain a model with only the information of the phase. The wavefront ψ, non-chromatic, if coordinate in the direction of the propagating waves is fixed, is given by the following equation: [23]

$$\psi(t,x,y) = \int_{-\infty}^{\infty} \psi_0(x,y,v) \mathrm{e}^{\mathrm{i}(\phi(x,y,v)-2\pi v t)} \mathrm{d}v, \tag{2-1}$$

with t the time $\in \mathbb{R}$. In case of a Cartesian coordinate system, $x, y \in \mathbb{R}$ are space coordinates, ϕ is the phase and v is the frequency.

2-1-1 Atmospheric turbulence

Turbulence in the atmosphere causes local fluctations in the refractive index of the air which creates spatial phase differences in the wavefront. The following assumptions are made.

- The amplitude of the light-waves, $\psi_0(v)$, is not affected by the turbulence.
- The turbulence causes the phase ϕ to be Gaussian random and zero mean [45].
- The refractive index does not depend on the frequency of light, there only one wavelength of the light is considered.
- The wavefront is flat before it enters the atmosphere, figure 2-1.

The contribution of a single layer of turbulent air to the wavefront can then be given by

$$\psi(x, y) = \mathrm{e}^{\mathrm{i}\phi(x,y)}. \tag{2-2}$$

If the turbulence consist out of only one turbulent layer then the disturbed wavefront is given by equation (2-2). This results in the fact that for one turbulent layer the wavefront is fully determined by the phase ϕ. Temporal effects will not be taken into account. For a more detailed overview see [22].

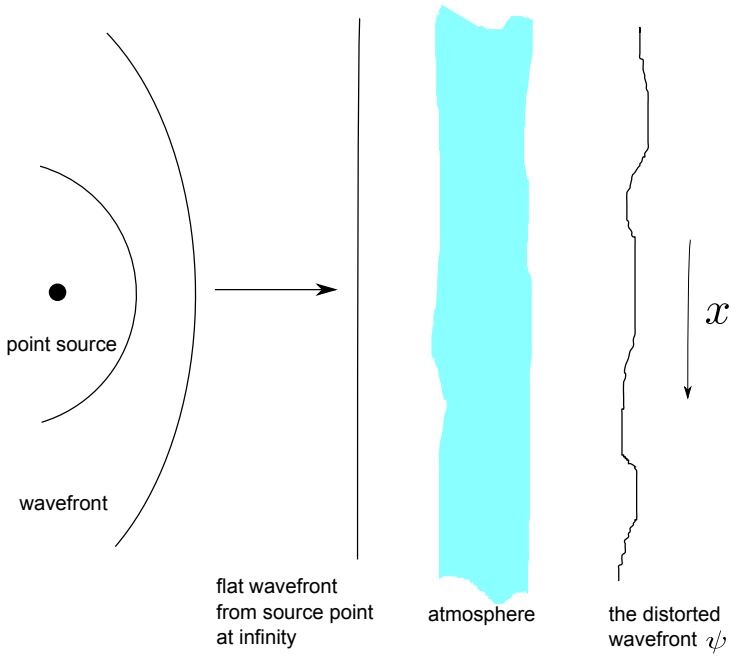


Figure 2-1: Visual representation of the wavefront

2-1-2 Qualitative view of the optical resolution

In order to have insight in how well adaptive optics is able to improve the resolution by correction for wavefront aberrations, the Strehl ratio and the point spread function will be introduced as a metric to evaluate the performance of wavefront reconstruction.

The point spread function

The point spread function is the spatial intensity distribution of a projected point source. It is given by $p(\alpha)$. Where α is the angular coordinate. For the diffraction limited case, the point spread function corresponds to the airy disk. At the origin the point spread function for the diffraction limited case is

$$p_0(0) = \frac{\pi}{4}\left(\frac{D}{\lambda}\right)^2,$$

where D is the diameter of the telescopes aperture and λ is the wavelenght of the light.

Strehl ratio

The Strehl ratio is a measure of how close, in the case of wavefront aberrations, the resolution is, to the case that the resolution is diffraction limited. The intensity at the origin of the actual point spread function function is compared to the intensity at the origin of point spread function in the diffraction limited case. The strehl ratio is given by the actual intensity at the origin of the point spread function, divided by the intensity at the origin of the point spread functin in the diffraction limited case,

$$\mathcal{S} = \frac{p(0)}{p_0(0)}.$$

The Strehl ratio can be approximated by [52, p. 302]

$$\mathcal{S} = \mathrm{e}^{-\sigma_\phi^2}, \tag{2-3}$$

with σ_ϕ the the mean square error of the phase points,

$$\sigma_\phi^2 = \frac{1}{N_\Phi} \left|\left| \Phi - \hat{\Phi} \right|\right|_2^2, \tag{2-4}$$

where $\hat{\Phi}$ is a vector with the estimate phase at different locations, Φ a vector with the actual phase at the same locations and N_Φ the number of estimated phase points.

2-2 Adaptive optics

Using adaptive optics it is possible to reduce the distortion of the wavefront and therefore improve the resolution. Using the assumptions in section 2-1, that is that the wavefront is flat before it enters the atmosphere, it possible to improve the sptatial resolution of the the telescope, by reducing the phase error between a flat wavefront and the observed wavefront. This is possible by measuring the wavefront and using deformable mirrors to adapt the phase of the wavefront according to the measurments. A telescope with the optical components can be seen in figure 2-2.

2-3 Sensing the wavefront

In this section it will be discussed how one can measure the wavefront using different wavefront sensors. Before that, different performance critieria for wavefront sensor will be discussed.

2-3-1 Performance criteria for wavefront sensing

In this section it will be shown how the criteria for optical efficiency, sensitivity of wavefront sensors can be quantified.

Optical efficiency

The optical efficiency gives information about the light loss within the sensor. It is given by the optical power of the light that arrives at the wavefront sensor, divided by optical power that ends at the detector.

$$\eta_{\mathrm{OE}} = \frac{P_{\mathrm{in}}}{P_{\mathrm{out}}} \times 100\%$$

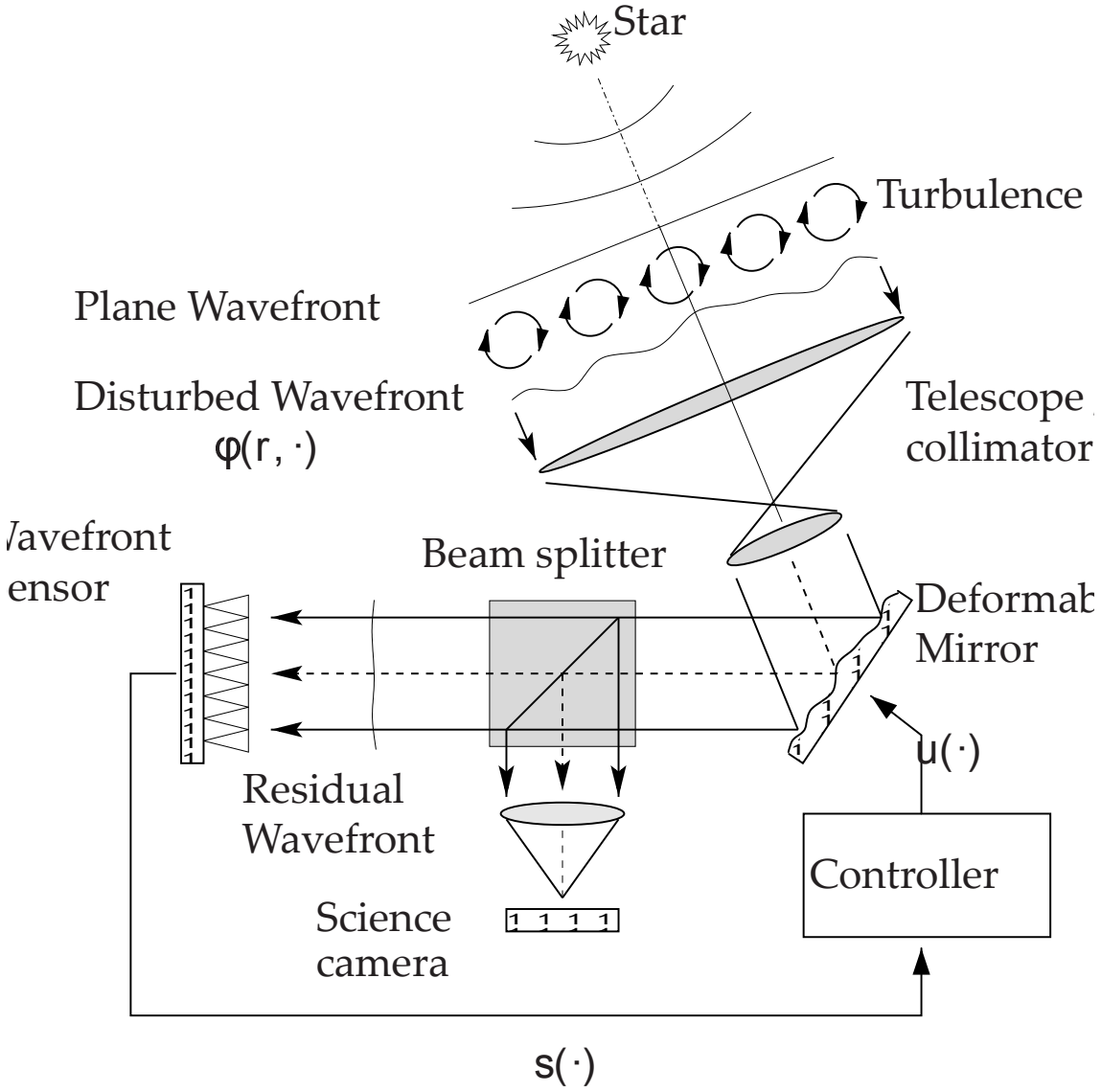


Figure 2-2: Optical setup of a telecope that uses adaptive optics *M. Verhaegen, Control for high resolution imaging, 2012*

Sensitivity of a wavefront sensor

The measurement sensitivity of a wavefront sensor is the sensitivity to photons in the spatial frequency domain. For more details see [25, 64]. Some wavefront sensors have the ability to change the sensitivity on-line. Since the sensitivity functions are not well defined in literature, the mean square error of the phase points can be used, which is defined by equation (2-4).

2-4 Wavefront sensors

Wavefront sensors can be classified in two main categories. Either pupil plane or focal plane sensors. For a pupil plane sensor, the beam is split up using a beam splitter. The detector is in a plane conjugate to the beam splitter. The focal plane sensor uses the whole-aperture to measure. Focal plane sensors require solving a non-linear non-convex problem, which is at this

moment not doable in real-time applications. Therefore these are not discussed. The pupil plane wavefront sensors that are discussed are the Lateral shearing interferometer, Shack Hartmann sensor, Curvature Sensor and Pyramid wavefront sensor.

2-4-1 Lateral shearing interferometer

The incoming wavefront will be combined with a shifted version of itself. Using the fringe pattern one can determine the phase difference between the shifted wavefront and non shifted wavefront. The phase differences are related to the slope of the wavefront. To measure both in x and y direction two interferometers are required, one for each direction. The interferometers are placed in a grid to measure at multiple locations. For a long time this was the most used sensor. But due to the following reasons not used anymore [46]:

- Optical efficiency limited to about 70% due to the use of gratings [46, p. 98].
- Complex hardware and related implementation difficulties.
- Similar measurement error for small sub apertures in comparison with the Shack Hartmann sensor, for larger sub apertures the Shack Hartmann sensor performs better [65].

2-4-2 Shack Hartmann sensor

The Shack-Hartmann sensor consists of an array of lenses, lenslets, which all have the same fixed focal length. Next to that, there is a photon sensor, either a quad-cell or a Charged Coupled Device (CCD), which is located a focal distance away from the array. Assuming that the wavefront is flat at the entrance of the lenslets, then the light going through a lenslet will create a spot at the photon sensor, with as spatial location the center of the lenslet. For a disturbed wavefront, the spot deviates from the center of lenslet. Using the center of gravity method one is able to estimate the location of the spot. By a first approximation the lens lets receive a tilted wavefront.

Relationship of slopes and spot locations In this paragraph it is discussed how the slope or phase difference relate to the spot locations. It is shown for one lenslet. The x and y location of the spot can be determined by the center of gravity method which is given by

$$c_x = \frac{\sum_{i,j} x_{i,j} I_{i,j}}{\sum_{i,j} I_{i,j}}, \quad c_x = \frac{\sum_{i,j} y_{i,j} I_{i,j}}{\sum_{i,j} I_{i,j}},$$

with $I_{i,j}$ denoting the image intensities at the CCD subarray and the $x_{i,j}$ and $y_{i,j}$ the corresping coordinates of the CCD subarray. The angle of arrivals are then given by

$$\alpha_x = \frac{c_x}{f}, \quad \alpha_y = \frac{c_y}{f}, \tag{2-5}$$

with f the focal length of the lenslets. The relation between the angle of arrival and the slope/phase difference is given by

$$s_x = \frac{\partial \phi(x,y)}{\partial x} \approx \frac{2\pi}{\lambda} d\alpha_x, \quad s_y = \frac{\partial \phi(x,y)}{\partial y} \approx \frac{2\pi}{\lambda} d\alpha_y, \tag{2-6}$$

with d the diameter of the lenslet. Coming equation 2-5 and 2-6 results in

$$s_x \approx \frac{2\pi}{\lambda} d \frac{c_x}{f}, \quad s_y \approx \frac{2\pi}{\lambda} d \frac{c_y}{f}.$$

Note that the magnification caused by the telescope is not taking into account and be corrected afterwards.

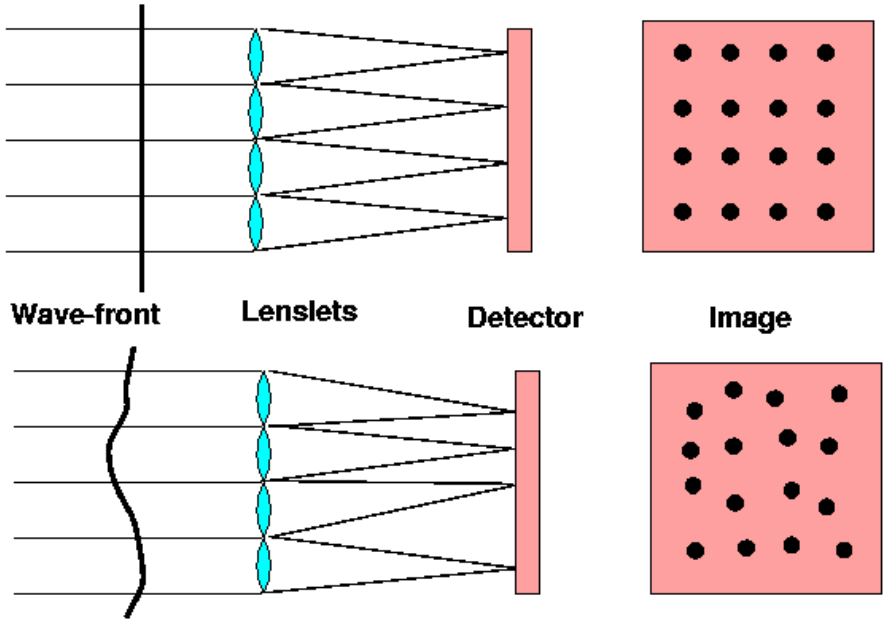


Figure 2-3: Working principle of the Shack Hartmann sensor *Cerro Tololo Inter-American Observatory* http://www.ctio.noao.edu/ atokovin/tutorial/part3/shwfs.gif (feb. 2013)

Using the location of the spots, the phase of wavefront slopes for each sub-aperture can be determined. Using a finite difference approximation of the phase of the wavefront , the following relation between the slope and the phase of the wavefront is obtained.

$$\mathbf{s} = \mathbf{A}_{\text{fin}} \phi + \mathbf{n}, \tag{2-7}$$

where $\mathbf{n}$ is the noise, assumed to be Gaussian with zero mean, as disscussed in section 2-1. $\mathbf{s}$ are the slopes, the interaction matrix $\mathbf{A}_{\text{fin}}$ is determined by the geometry and ϕ is a vector with the to be estimated phase at the grid location $(x_{i..\text{n}}, y_{j..\text{n}})$ which also depend on the geometry at a time instant t where n is the grid size. To use the finite different approximation a geometry must be defined. In the next section different geometries are disscussed.

Geometries

The geometry determines the location of the actuators with respect to the location of the measurements.

- Fried geometry, figure 2-4a [18]

– The slopes are given by

$$s_{x(i,j)} = \frac{\partial \phi(x_i, y_j)}{\partial x} \approx \frac{1}{2D_{\mathrm{L}}} \left[\left(\phi(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}) + \phi(x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}}) \right) - \left(\phi(x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}}) + \phi(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}}) \right) \right],$$
$$s_{y(i,j)} = \frac{\partial \phi(x_i, y_j)}{\partial y} \approx \frac{1}{2D_{\mathrm{L}}} \left[\left(\phi(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}) + \phi(x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}}) \right) - \left(\phi(x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}}) + \phi(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}}) \right) \right], \tag{2-8}$$

where D_{L} is the grid size length

– The location of the measured slopes do not coincidence with the location of the estimated phase slopes.

– Not waffle mode, discontinuous checkerboard pattern phase error, sensitive.

- Southwell, figure 2-4b, [54]

– The slopes are given by

$$\begin{aligned} \frac{s_x(x_{i+1}, y_j) + s_x(x_i, y_j)}{2} &\approx \frac{1}{D_{\mathrm{L}}} \left[(\phi(x_{i+1}, y_j) - \phi(x_i, y_j)) \right], \\ \frac{s_x(x_{i+1}, y_j) + s_y(x_i, y_j)}{2} &\approx \frac{1}{D_{\mathrm{L}}} \left[(\phi(x_i, y_{j+1}) - \phi(x_i, y_j)) \right]. \end{aligned} \tag{2-9}$$

– Equation (2-9) results in the following relation between slopes and the phase points

$$\mathbf{D}_{\mathrm{s}}\mathbf{s} = \mathbf{A}_{\mathrm{s}}\boldsymbol{\phi} + \mathbf{D}_{\mathrm{s}}\mathbf{n},$$

which is different from equation (2-7).

– Measured location coincide with estimated phase locations .

– Harder to calibrate the sensor than using a Fried geometry [42, p. 120].

– Superior in noise propagation compared to the Fried Geometry[68].

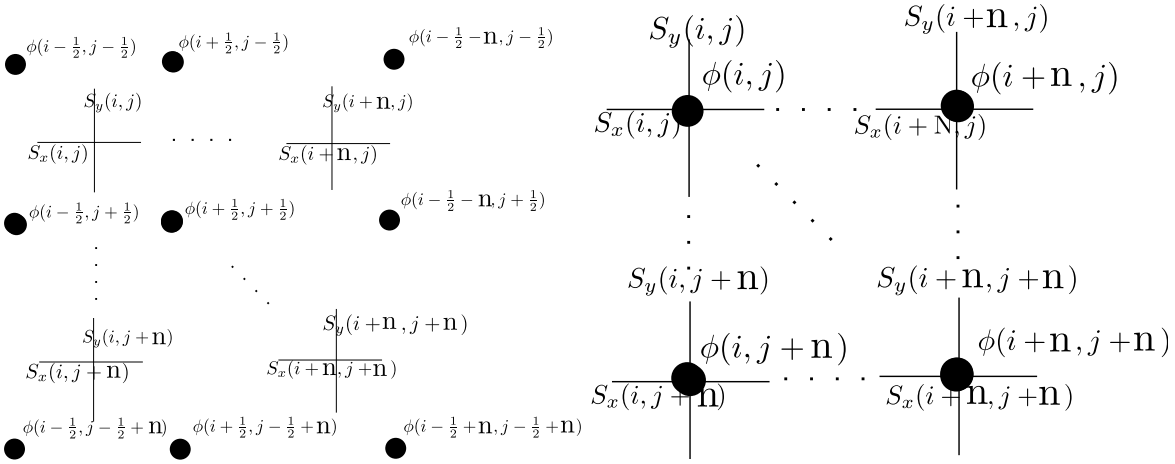
There is also the Hudgin geometry [29], it is originally used for the shearing interferometer. Though a modified version of it is used for the Shack Hartmann sensor, in combination with the fast Fourier transform [43].

Performance of a Shack Hartmann sensor

- One of the main advantages of using the Shack Hartmann is its high optical efficiency [52, p. 291] .
- No option to adjust sensitivity and dynamic range on line
- Wavefront error, σ_ϕ^2, for one sub aperture is similar to that of a curvature sensor. For all sub apertures it has to be multiplied with ln(*number of sub apertures*)

2-4-3 Curvature sensor

The curvature sensor is very similar to the Shack Hartmann sensor. Again the aperture is divided into sub-apertures. The intensity is measured at two detector arrays, one for the focal plane I_1 and one behind the focal plane I_2, see figure 2-5. A practical implementation requires an oscillating variable curvature mirror with only one detector [47]. A local wavefront curvature



(a) Fried geometry **(b)** Southwell geometry

Figure 2-4: Geometries for the Shack Hartmann sensor, lines are the measured phases, the dots are the estimated phase points and n are the number of lenslets in one direction.

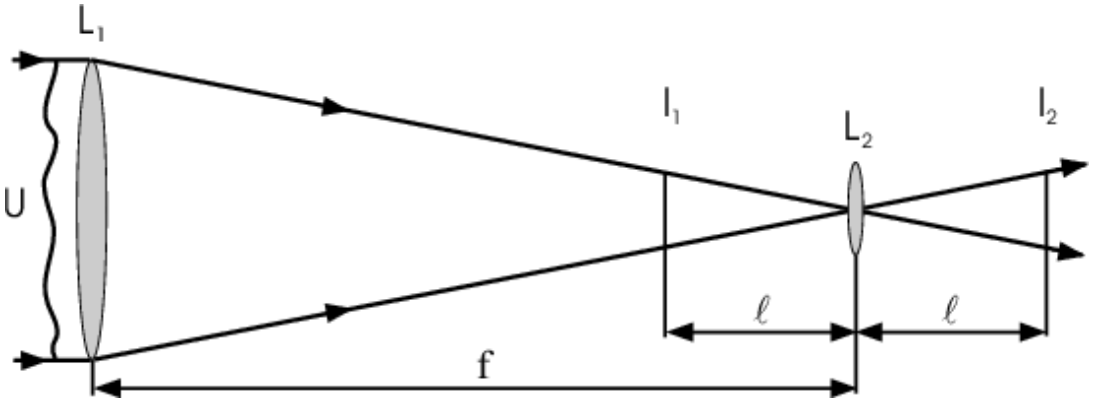


Figure 2-5: Curvature sensor working principle with *U* being the incoming disturbed wavefront, *V. V. Voitsekhovich and L. J. Sánchez* (2002) Effect of scintillations in curvature sensing [59]

leads to a local difference in intensity at the detector arrays, which is related to the curvature of the wavefront.

It is possible to avoid a computational step and directly actuate the mirrors when using bimorph mirrors. Though modal control approach which will be discussed in section 3-2 has a better performance [48]. The relation between the intensity I_1 and I_2 at the detectors, see figure 2-5, and the phase can be written as a Poisson relation with Neumann boundary conditions:

$$\frac{I_1(x,y)-I_2(x,y)}{I_1(x,y)+I_2(x,y)} = \frac{f(f-l)}{l}\left(\frac{\partial\phi(x,y)}{\partial\vec{n}}\delta_c - P(x,y)\nabla^2\phi(x,y)\right), \tag{2-10}$$

with f being the focal length, l the distance of the detector to the focal point, see figure 2-5, $\frac{\partial\phi(x,y)}{\partial\vec{n}}\delta_c$ represent the normal derivatives on the edge and *P(x,y)* is the pupil transmission function. When using finite difference method, this can be split up in two parts. Which can be rewritten in matrix form using the discrete time Poisson equation. Within the grid

$$\begin{aligned}\nabla^2\phi(x,y) =& \frac{I_1(x,y)-I_2(x,y)}{I_1(x,y)+I_2(x,y)}\frac{1}{P(x,y)}\frac{l}{f(f-l)}\\ \mathbf{A}_\mathrm{k}\phi =& f_\mathrm{k}(x_i,x_j),\end{aligned} \tag{2-11}$$

where $\mathbf{A}_\mathrm{k}$ is the discrete Laplacian matrix. On the boundary grid the following equation satisfies

$$\begin{aligned}\frac{\partial\phi(x,y)}{\partial\vec{n}} =& \frac{I_1(x,y)-I_2(x,y)}{I_1(x,y)+I_2(x,y)}\frac{l}{f(f-l)}\frac{1}{\delta_c}\\ \mathbf{A}_\mathrm{b}\phi =& f_\mathrm{b}(x_i,x_j),\end{aligned} \tag{2-12}$$

where $\mathbf{A}_\mathrm{b}$ is the discrete time version of the derivative in the normal direction at the boundaries. Combining these equations results in

$$\begin{bmatrix}\mathbf{A}_\mathrm{k}\\ \mathbf{A}_\mathrm{b}\end{bmatrix}\phi = \begin{bmatrix}f_\mathrm{k}(x_i,x_j)\\ f_\mathrm{b}(x_i,x_j)\end{bmatrix}. \tag{2-13}$$

Performance of a curvature sensor

- Since curvature is measured, tilt/tip cannot be observed.
- The use of second derivative requires higher numerical precision than the use of the first derivative. With non-direct methods, more iterations are required to obtain a good precision.
- The wavefront error σ_ϕ^2, is similar to the Shack-Hartmann sensor. However instead of logarithmic, the variance for multiple sub-apertures scales linearly with the number of sub-aperture [32, 27].
- Sensitivity can be changed on line by varying the distance of the detector arrays to the focal point l, see figure 2-5. Though changing sensitivity is subject to the spatial resolution.
- Due to the oscillating curvature mirror, slightly more complex than the Shack Hartmann sensor.

2-4-4 Pyramid wave-front sensor

At the focal point a prism is placed, which splits the beam of light into four parts. Then using a relay lens, these parts are re-imaged at a high resolution detector as depicted in figure 2-6. The spatial sampling of the pupil can be adjusted in case a zoom relay lens is used. The relay lens extends the optical path. Normally, in case of a flat wavefront, a cell located in one of the four areas of the detector has the same intensity as the other three cells located at the same location in the other areas, see figure 2-7. If there is a slope change in the wavefront, then some cells in the four parts I_1, I_2, I_3 and I_4 have not equal light intensities anymore. From those different intensities the sign of the derivatives of the wavefront can be determined, since the system can be seen as a Foucault knife edge test. To retrieve the derivative itself, modulation of the prism is required. The modulation is done on the center of the prism. The prism moves in a circle with a constant speed around a center-point. In this case all the four areas are illuminated when there is a distorted wavefront. This moving part is not desirable since it makes the mechanical design difficult. Some alternatives are proposed, using the natural modulation caused by the atmosphere [10, 9] or using an additional light source [44]. It is shown that the following geometrical approximation equation is satisfying [6]

$$\begin{aligned}\frac{\partial\psi(x,y)}{\partial x} =& \frac{\xi_0}{f}\frac{I_1(x,y)-I_2(x,y)-I_3(x,y)+I_4(x,y)}{I_1(x,y)+I_2(x,y)+I_3(x,y)+I_4(x,y)},\\ \frac{\partial\psi(x,y)}{\partial y} =& \frac{\xi_0}{f}\frac{I_1(x,y)+I_2(x,y)-I_3(x,y)-I_4(x,y)}{I_1(x,y)+I_2(x,y)+I_3(x,y)+I_4(x,y)},\end{aligned} \tag{2-14}$$

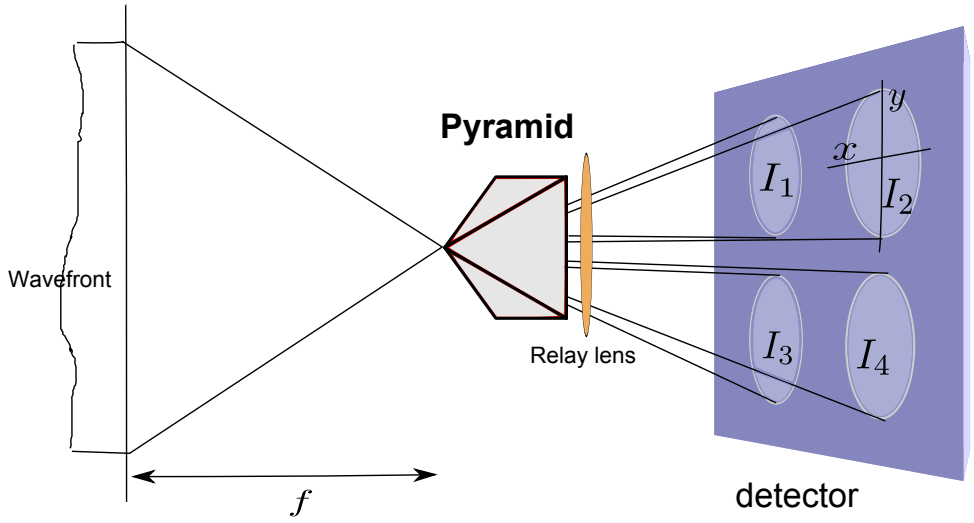


Figure 2-6: Working principle of the pyramid sensor

where ξ_0 is the modulation amplitude. For large amplitudes the system is less nonlinear but also less sensitive. The pyramid sensor can have a higher sensitivity than a Shack-Hartmann sensor [17]. In contrast to the Shack Hartmann sensor, the resolution of the Pyramid sensor is not determined by the number of the sub apertures but by the resolution of the CCD sensor, figure 2-7. Instead of using a pyramid as a prism, it also possible to use multi-sided prisms. The prism must be of a high optical quality which makes it difficult to fabricate. An acceptable quality can be achieved with the use of X-ray lithography [41]. Similar to the Shack Hartmann sensor there is a linear relationship between the slopes and the phase estimates (equation (2-7)).

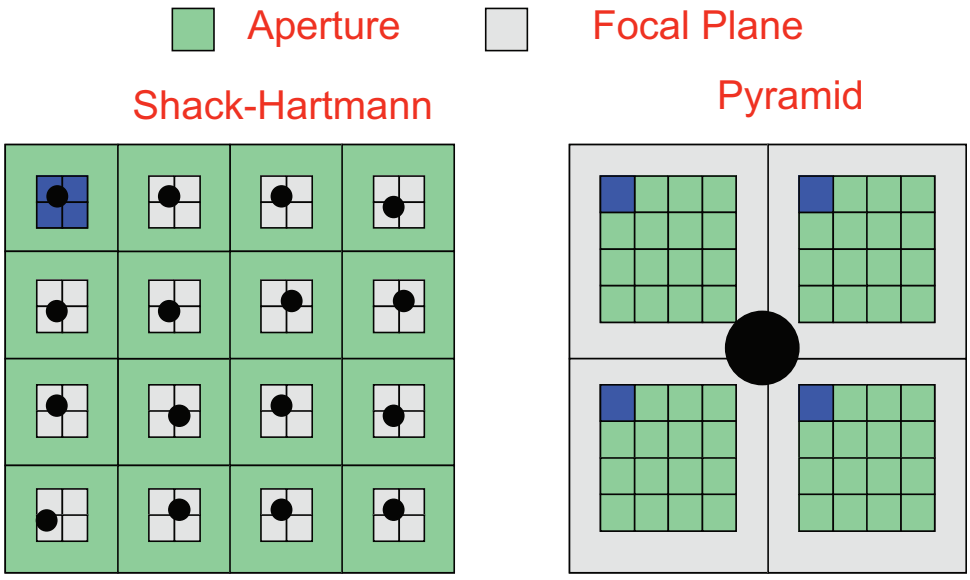


Figure 2-7: Comparison of the pyramid sensor with the Shack Hartmann sensor *Marcos van Dam & Richard Clare* Center for Adaptive Optics University of California http://www.cfao.ucolick.org/aosummer/2007/pdfs/Wavefront_Sensing_van_Dam.pdf

Performance of pyramid wavefront sensor

- Can have a higher sensitivity than a Shack Hartmann sensor [17].
- Spatial sampling and sensitivity can be adjusted on-line.
- Usually it requires a moving part.

2-4-5 Summary and comparison of the wavefront sensors

A short overview is given of the performance of the Shack Hartmann sensor, curvature sensor and the pyramid sensor.

Table 2-1: Comparison of Shack Hartmann sensor, curvature sensor and a pyramid sensor

	Shack Hartmann sensor	curvature sensor	pyramid wavefront sensor
Type of measurements	slopes	curvature	slopes
optical efficiency [a]	high	high	high
on line sensitivity adjustment	no	yes by adjusting detector to focal point, though subject to aperture resolution	yes by adjusting the amplitude modulation, though subject to nonlinearity
on line spatial aperture resolution adjustment	no	yes subject to sensitivity	yes using the relay lens
wavefront error σ_ϕ^2, see equation (2-4) for number of sub-apertures N, κ_i is a constant and N_p are the number of photons	$\kappa_1 \left(\kappa_2 + \frac{\ln(N)}{N_\mathrm{p}}\right)$	$\kappa_3 \left(\kappa_4 + \frac{N}{N_\mathrm{p}}\right)$	$\frac{\kappa_5}{N_\mathrm{p}}$
practical implementation	very sensitive for misalignment's of the lenslets	more complex because of moving curvature mirrors	complex if mechanical modulation is used

[a] Compared to the lateral shearing interferometer all these sensors have a high optical efficiency since they have no components which can limit the optical efficiency significantly.

Currently the Shack Hartmann sensor is the most used wavefront sensor and also well-established. In this thesis it is assumed that the Shack Hartmann is used. It will be shown that Spline based ABeration REconstruction (SABRE) will in theory also work for curvature measurements. For the distributed version this is still something to investigate. In the next chapter it will be discussed how the slope measurements can be used to reconstruct the wavefront.

Chapter 3

Wavefront reconstruction

In this section it is shown how the phase of the wavefront can mathematically be described. Several methods will be described. To go from the sensed slopes to a function of the wavefront, there are two widely used approaches: modal and zonal. It assumed that the Shack Hartmann is used to obtain the slopes which is addressed in section 2-4-2.

3-1 Zonal methods

For the zonal approach, the wavefront is described by local functions.

3-1-1 Finite difference method

It is possible using finite difference to obtain the following relationship, for both curvature and slopes measurements $\mathbf{s}$, and the phase points ϕ:

$$\mathbf{s} = \mathbf{A}_{\text{fin}}\phi.$$

A definition of the $\mathbf{A}_{\text{fin}}$ matrix can be found in section 2-4-2.

The estimate can be found by solving the following least squares problem:

$$\min_{\phi} ||\mathbf{s} - \mathbf{A}_{\text{fin}}\phi||_2^2 .$$

Regularized least-squares can be used, which takes also statistics into account:

$$\min_{\phi} \left(||\mathbf{s} - \mathbf{A}_{\text{fin}}\phi||^2 + \alpha\phi^{\mathrm{T}}\mathbf{C}_{\phi}^{-1}\phi \right),$$

with α a signal to noise parameter and $\mathbf{C}_{\phi}$ the covariance matrix of ϕ. It can be obtained with knowledge about the turbulence in the air. Note that the inverse of the covariance matrix is required. Which is a non-sparse matrix. Though it can be approximated by a sparse matrix [63, 16]. With the already sparse $\mathbf{A}_{\text{fin}}$ it is possible to solve the wavefront reconstruction problem with linear time complexity using the multi-grid Preconditioned Conjugate Gradient (PCG) [21, 55]. Some limitations of using finite differences are:

- Using Finite difference one assumes a linear wave-front while it is actually non linear .
- Not trivial for non-rectangular arrays.

3-1-2 SABRE

A recent method proposed by de Visser and Verhaegen [13] is Spline based ABeration REconstruction (SABRE). The phase is defined at every location in contrast to the Finite Difference method where only the phase at the grid locations is known. A second difference is that it does not use the finite differences but multi-variate B-splines to model the wavefront, which makes it generally applicable. The B-splines are only valid locally, which SABRE a zonal method. Some advantages are.

- Invariant for wavefront sensor geometry
- Non-linear
- Noise smoothing
- Not subject to waffle mode

In the next chapter, the SABRE method will be explained in more detail. After that the different wavefront reconstructions methods will be compared.

3-2 Modal approach

In the case of a modal representation, the estimate represent a coefficient of an aperture function. The wavefront is represented by a set of functions. Each function is valid over the whole aperture. They have to be linearly independent but not necessarily orthogonal [58, p. 229]. The phase can then be given by

$$\phi(x,y) = \sum_{k=1}^{K} a_k Z_k(x,y),$$

where a_k are the coefficients and Z_k are functions. Several sets of functions will be discussed to approximate the wavefront. This results in the following problem:

$$\mathbf{s} = \mathbf{B}_{\text{mod}}\mathbf{a} + \mathbf{n} \tag{3-1}$$

which can be written as a linear least squares problem. $\mathbf{a}$ is a vector with the coefficients of the basis functions and $\mathbf{B}_{\text{mod}}$ contains the derivative of every basis function evaluated at the measurements locations.

3-2-1 Polynomial functions

Polynomial functions can be used to approximate the wavefront. Two widely used functions are:

- Zernike polynomials
  - Most natural way, since lower order terms are related to the classical aberrations.
- Karhunen Loeve functions
  - Diagonal covariance matrix contrary to the Zernike polynomials. This reduces the error. [38]

But using polynomials to approximate functions have major drawbacks that makes them not a very good option.

- With a choice of a set of polynomials, one is restricted to either a rectangular grid or a circular grid.
- Limited in approximation power.
- Might be unstable for large number of data.
- Runge phenomenon implies large oscillations around the boundaries of the grid.
- **B** matrix, in equation 3-1 is a dense matrix

3-2-2 Radial basis functions

Radial basis function are widely used for approximation two dimensional planes, since they are simple. They can be formulated by

$$f = \sum_{i=1}^{n} c_i F(|x - x_i|),$$

where F is a radial basis function and $r = |x - x_i|$ is a distance in d-dimensional space. It can be rewritten in the form of equation (3-1), with $\mathbf{B}_{\text{mod}}$ given by $B_{\text{mod}ij} = F(|x_i - x_j|)$

- Although radial basis function can decay quickly, they still hold globally which makes the $\mathbf{B}_{\text{mod}}$ in equation 3-1 dense.
- for large number of data points, the $\mathbf{B}_{\text{mod}}$ matrix that determines the coefficients of the radial basis function is ill-conditioned [34, p. 58]
- does not require a grid

3-2-3 Transforms

Another approach is going to the Fourier domain or the Wavelet domain. It does not involve solving a least squares problem. The integration is done in the Fourier domain or the Wavelet domain, this simplifies computation. To integrate the slopes, a geometry, as was discussed for the Shack Hartmann sensor in section 2-4-2, is required.

- Fourier domain
  - Uses a set of complex exponential functions .
  - Computational complexity of $\mathcal{O}(N \log N)$.
  - Complex for non-circular, non-homogeneous arrays .
- Haar Wavelet [26]
  - Similar to Fourier transform but than using square shaped functions.
  - Complex for non-rectangular, non-homogeneous arrays.
  - Computational complexity $\mathcal{O}(N)$.

Chapter 4

SABRE

The previous chapter ended with good reasons to use the Spline based ABeration REconstruction (SABRE) method. Here we introduce the bivariate simplex B-splines as proposed by [12]. It will also be linked to wavefront reconstruction. A very brief description is given. For a more detailed overview of simplex B-splines, see [12, 13].

4-1 Triangulation

Simplices are used to build up a grid. There are different methods to place the simplices using a set of given vertices. Widely used is the Delaunay triangulation, it maximizes the minimum angles inside the simplices and is also it is extremely flexible, in the sense that there are no restrictions on the locations of the vertices. Type I/II triangulations are constructed using subdivision, filling symmetric n-cubes. The simplicity of these Type I/II guarantees well defined simplices in contrast to the Delaunay triangulation.

4-1-1 Example Delaunay Triangulation

For an unstructured grid, the Delaunay triangulation is a relatively easy method to obtain a triangulation. The measurement locations of the slopes are the locations of the spots created at the CCD sensor. These locations can be used as the vertices of the simplices of the triangulation. If one uses linear B-splines, the slope of B-splines within a simplex can be given by the average of the three measured slopes at the vertices of the simplex.

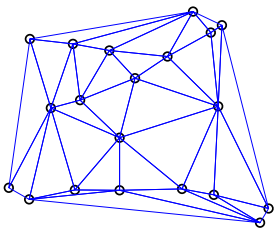


Figure 4-1: Delaunay triangulation, **O** are the measurement locations

4-1-2 Trade-off between noise and approximation power

In the case that one wants to smooth noise, you would use many measurements per simplex and you would use linear B-splines. To get the highest approximation power one would use only a few measurements per simplex and one would use second order or higher order B-splines. Obviously, there is a trade-off between the former and the latter.

4-2 Basis functions and B-coefficients

First the Cartesian coordinate system within the triangle are transformed to the Barycentric coordinates. This is for example essential for the affine property of the B-spline functions. For a triangle, this is then

$$\begin{aligned} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} =& \mathbf{V}^{-1}(v_0, v_1, v_2) \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}}, \\ b_0 =& 1 - b_1 - b_2, \\ b(\mathbf{x}) :=& (b_0, b_1, b_2), \end{aligned} \tag{4-1}$$

where v_0, v_1, v_2 are the locations of the vertices of one triangle, $\mathbf{x}$ are coordinates within the triangle in the Cartesian coordinate system and $b(\mathbf{x})$ are the transformed normalized coordinates in the Barycentric coordinate system. The basis functions of the simplex B-splines are Bernstein

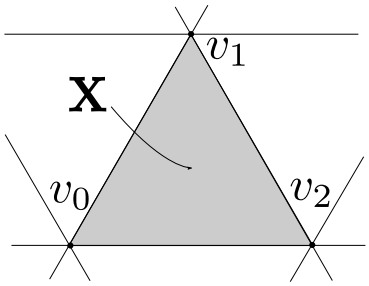


Figure 4-2: One triangle with vertices v_0, v_1, v_2 and Cartesian coordinates $\mathbf{x}$.

polynomials. Using the multinomial theorem any polynomial of total degree d can be rewritten in a sum of monomials

$$\begin{aligned} (b_0 + b_1 + b_2)^d =& \sum_{\kappa_0+\kappa_1+\kappa_2=d} \underbrace{\frac{d!}{\kappa_0!\kappa_1!\kappa_2!} b_0^{\kappa_0} b_1^{\kappa_1} b_2^{\kappa_2}}_{B_\kappa^d(b(\mathbf{x}))} \\ \kappa :=& (\kappa_0, \kappa_1, \kappa_2) \in \mathbb{Z} \\ \kappa_0 + \kappa_1 + \kappa_2 =& d,\ \kappa_0 \geq 0, \kappa_1 \geq 0, \kappa_2 \geq 0 \end{aligned} \tag{4-2}$$

Each function has a maximum at a different location. If each function is now linked to a B-coefficient $c \in \mathbb{R}$. Then locally within the triangle the shape of the function can be changed. The polynomial $p(b(\mathbf{x}))$ is then given by

$$p(b(\mathbf{x})) := \begin{cases} \mathbf{B}^d(b(\mathbf{x})) \cdot \mathbf{c}^t & \mathbf{x} \in \text{ triangle} \\ 0 & \mathbf{x} \notin \text{ triangle} \end{cases} \tag{4-3}$$

where $\mathbf{B}^d(b(\mathbf{x}))$ is a list of the monomials given in equation (4-2) and $\mathbf{c}^t$ is a vector with the B-coefficients corresponding to each monomial function. Extending this for multiple triangles

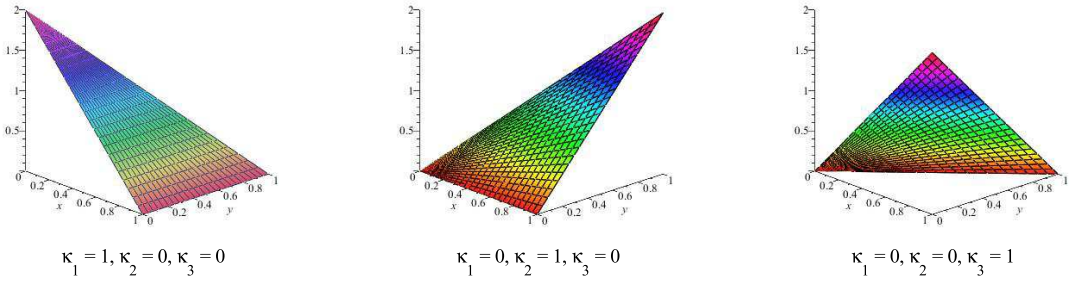


Figure 4-3: Linear B-spline functions

for continuity order r gives where

$$s_d^r(b(\mathbf{x})) := \mathbf{B}^d \cdot \mathbf{c} \quad \mathbf{x} \in \text{triangulation},$$

$\mathbf{B}^d$ is a vector with all basis-functions for each triangle $\begin{bmatrix}\mathbf{B}_{t_1}^d & \mathbf{B}_{t_2}^d & ....\end{bmatrix}$ and $\mathbf{c}^t$ are the B-coefficients for each triangle $\begin{bmatrix}\mathbf{c}^{t_1^{\mathrm{T}}} & \mathbf{c}^{t_2^{\mathrm{T}}} & ...\end{bmatrix}^{\mathrm{T}}$ as defined by equation (4-3).

4-3 Continuity between triangles

To explain how the continuity between triangles is described, first the B-coefficient net is introduced.

The B-coefficient net The B-coefficients are ordered in a unique spatial structure called the B-coefficients net, or B-net for short. The spatial location of each B-coefficient corresponds to the location of the maximum of the corresponding B-spline. For an example of the B-net see figure 4-4. Using the B-net the set up of continuity conditions is simplified. On each vertex, there are a number of continuity conditions depending on the continuity order r and degree d. This results in a constraint matrix $\mathbf{H}$. The continuity constraints are then formulated by

$$\mathbf{Hc} = 0. \tag{4-4}$$

If t_i and t_j are triangles with two shared vertices then each line in equation (4-4) is a continuity condition given by the following equation:

$$-c^{t_i}_{(\kappa_0,\kappa_2,m)} + \sum_{|\gamma|=m} c^{t_j}_{(\kappa_0,\kappa_1,0)+\gamma} \mathbf{B}^m_\gamma \left(b\left(\mathbf{w}\right)\right) = 0, \quad 0 \leq m \leq r,$$

with $\gamma := (\gamma_0, \gamma_1, \gamma_2)$ a multi-index independent of κ and $\mathbf{w}$ is the location of the non-shared vertice of triangle t_i. The constraint matrix $\mathbf{H}$ can easily become over determined. By building the constraint matrix $\mathbf{H}$, for each added continuity condition it must be checked whether it makes the matrix $\mathbf{H}$ over determined, if that is the case that condition must be dropped.

4-4 Derivatives of B-splines functions

The mth order directional derivative is given by

$$D^m_{\mathbf{u}} p(b(\mathbf{x})) = \frac{d!}{(d-m)!} \mathbf{B}^{d-m}(b(\mathbf{x})) \mathbf{P}^{d,d-m}(\mathbf{a}) \cdot \mathbf{c}^t,$$

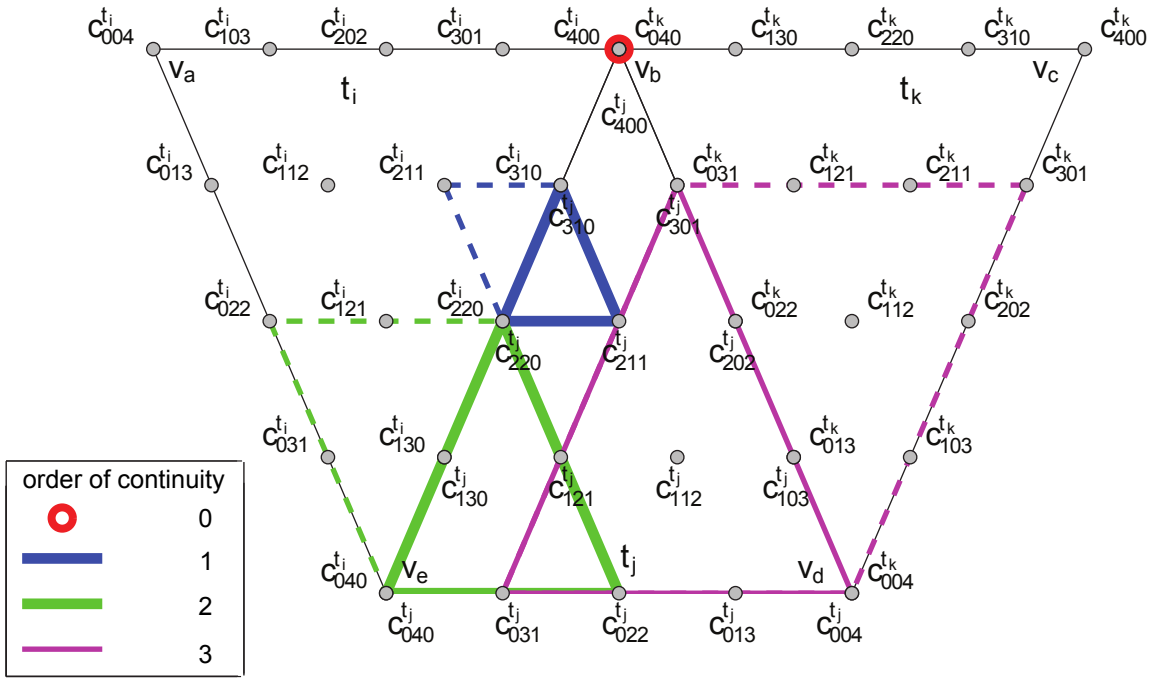


Figure 4-4: B-net for fourth order B-spline functions and an example of the structure of continuity conditions for different orders of continuity. *Cornelis C. de Visser and Michel Verhaegen, Wavefront reconstruction in adaptive optics systems using nonlinear multivariate splines*

with $\mathbf{u}$ a unit directional coordinate vector in Cartesian coordinates, $\mathbf{u} = \mathbf{v} - \mathbf{w}$ and $\mathbf{a}$ is the corresponding coordinate vector using Barycentric coordinates, $\mathbf{a} = b(\mathbf{v}) - b(\mathbf{w})$. $\mathbf{P}$ is the Castle Jau matrix. The first order derivative is given by

$$\begin{aligned} s_x(x,y) =& \frac{d!}{(d-1)!}\mathbf{B}^{d-1}(b(x,y))\mathbf{P}^{d,d-1}(\mathbf{a}_x)\cdot\mathbf{c} + \mathbf{n}, \\ s_y(x,y) =& \frac{d!}{(d-1)!}\mathbf{B}^{d-1}(b(x,y))\mathbf{P}^{d,d-1}(\mathbf{a}_y)\cdot\mathbf{c} + \mathbf{n}, \end{aligned} \tag{4-5}$$

with $\mathbf{n}$ being measurement noise. and the 2D Laplacian is given by

$$\mathcal{L}(x,y) = \frac{d!}{(d-2)!}\mathbf{B}^{d-2}(b(x,y))\left(\mathbf{P}^{d,d-2}(\mathbf{a}_x) + \mathbf{P}^{d,d-2}(\mathbf{a}_y)\right)\cdot\mathbf{c} + \mathbf{n}. \tag{4-6}$$

This equation can be simplifed to

$$\mathbf{s} = \mathbf{D}\mathbf{c}, \tag{4-7}$$

where

$$\mathbf{D} = d\mathbf{B}^{d-1}\mathbf{P}_{\mathbf{u}}^{d.d-1}. \tag{4-8}$$

4-4-1 SABRE first derivative measurements

Anchor constraint

The anchor constraint is used to fix unknown integration constants. Integrating the first derivative results in

$$\begin{aligned}\int_{\mathcal{T}} D^1_{\mathbf{u}} p(b(\mathbf{x}))\mathrm{d}\mathbf{u} &= \int_{\mathcal{T}} d\mathbf{B}^{d-1}\mathbf{P}^{d,d-1}\mathbf{c}\mathrm{d}\mathbf{u} \\ &= \mathbf{B}^d\mathbf{c} + k\end{aligned}$$

with k an unknown integration constant. Using the affine property of B-spline functions,

$$p(b(\mathbf{x})) = \mathbf{B}^d\left(\mathbf{c} + k\cdot\mathbb{1}\right),$$

with $\mathbb{1} = \begin{bmatrix}1 & 1 & ... & 1\end{bmatrix}$. Now $\mathbf{c}$ can be redefined as

$$\mathbf{c} = \begin{bmatrix} c^{t_1}_{d,0,0} + k \\ \tilde{\mathbf{c}} + k\cdot\mathbb{1}\end{bmatrix}.$$

The anchor constraint can than for example be defined as:

$$k = -c^{t_1}_{d,0,0}$$

and given in vector form if $\mathbf{h}_1 = \begin{bmatrix}1 & 0 & ... & 0\end{bmatrix}$ such that

$$\mathbf{h}_1\cdot\mathbf{c} = 0. \tag{4-9}$$

Equation (4-4) and (4-9) can be combined to the constraint matrix

$$\mathbf{F} = \begin{bmatrix}\mathbf{H} \\ \mathbf{h}_1\end{bmatrix} \tag{4-10}$$

Combining equation (4-7) and (4-10) results in the following systems of linear equations:

$$\begin{aligned}\mathbf{s} &= \mathbf{D}\mathbf{c} + \mathbf{n}, \\ \mathbf{0} &= \mathbf{F}\mathbf{c}\end{aligned}. \tag{4-11}$$

Equation (4-11) can be rewritten in the following optimization problem:

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}} ||\mathbf{D}\mathbf{c} - \mathbf{s}||_2^2 \tag{4-12a}$$

$$\text{subject to} \quad \mathbf{F}\mathbf{c} = \mathbf{0}, \tag{4-12b}$$

which finally results in

$$\phi(\mathbf{x}) = \mathbf{B}^d\hat{\mathbf{c}}.$$

4-4-2 SABRE curvature measurements

Anchor constraint

Integrating the second derivative results in

$$\begin{aligned}\int_{\mathcal{T}}\int_{\mathcal{T}} D^2_{\mathbf{u}} p(b(\mathbf{x}))\mathrm{d}^2\mathbf{u} &= \int_{\mathcal{T}}\int_{\mathcal{T}} d(d-1)\mathbf{B}^{d-2}\mathbf{P}^{d,d-2}\mathbf{c}\mathrm{d}^2\mathbf{u} \\ &= \int_{\mathcal{T}} d\mathbf{B}^d\left(\mathbf{P}^{d,d-1}\mathbf{c} + k_1\cdot\mathbb{1}\right)\mathrm{d}\mathbf{u} \\ &= \mathbf{B}^d\left(k_1\mathbf{c} + k_2\cdot\mathbb{1}\right).\end{aligned}$$

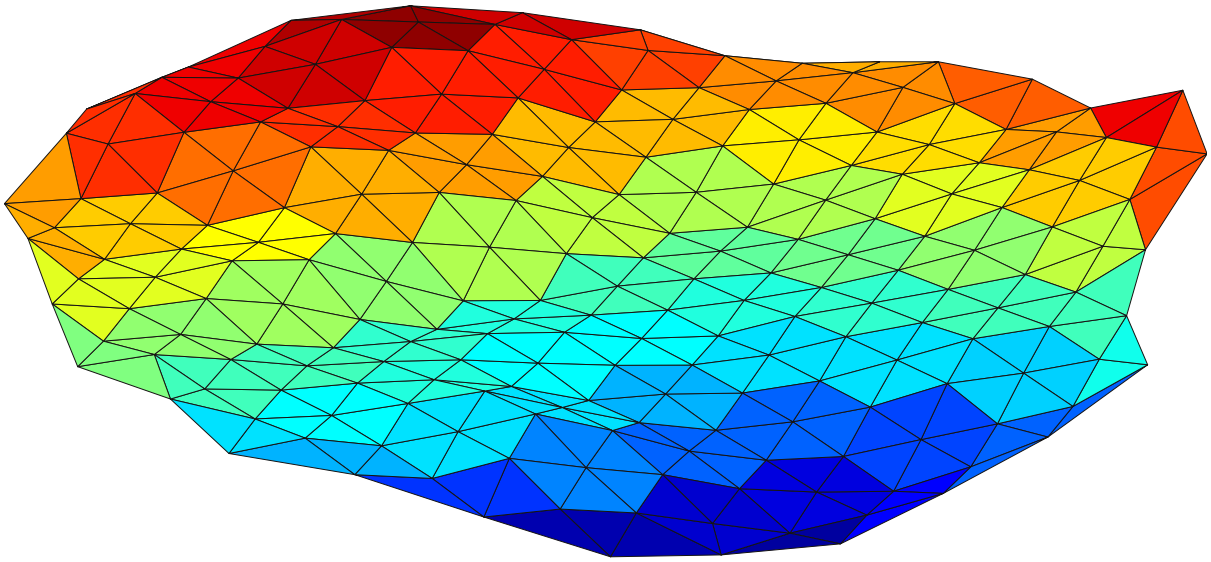


Figure 4-5: Example of using linear Simplex B-splines to approximate a surface

There are two integration constants k_1 and k_2. Now $\mathbf{c}$ can be redefined as

$$\mathbf{c} = \begin{bmatrix} k_1 c^{t_1}_{d,0,0} + k_2 \\ k_1 c^{t_1}_{d-1,1,0} + k_2 \\ k_1 \tilde{\mathbf{c}} + k_2 \cdot \mathbb{1} \end{bmatrix}.$$

The anchor constraints can then be defined as

$$\begin{aligned} k_1 c^{t_1}_{d,0,0} + k_2 &= 0, \\ k_1 c^{t_1}_{d-1,1,0} + k_2 &= 0 \end{aligned}$$

and in vector form

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \end{bmatrix}}_{\mathbf{h}_2} \cdot \mathbf{c} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4-13}$$

Using the equations (4-4),(4-6) and (4-13) the reconstruction problem is given by

$$\begin{aligned} \mathcal{L}(x,y) &= d(d-1)\mathbf{B}^{d-2}(b(x,y)) \left(\mathbf{P}^{d,d-2}(\mathbf{a}_x) + \mathbf{P}^{d,d-2}(\mathbf{a}_y) \right) \mathbf{c} + \mathbf{n} \\ \mathbf{0} &= \mathbf{F}\mathbf{c}, \end{aligned}$$

with the constraint matrix $\mathbf{F}_2$

$$\mathbf{F}_2 = \begin{bmatrix} \mathbf{H} \\ \mathbf{h}_2 \end{bmatrix}.$$

4-5 Solving SABRE

In this section it will be shown how the SABRE can be calculated. Especially finding the B-coefficients will be discussed, since this is the most computational intensive part. See also figure 4-6. If we recall the SABRE problem from equation (4-12):

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}} ||\mathbf{D}\mathbf{c} - \mathbf{s}||_2^2 \tag{4-14a}$$

$$\text{subject to} \quad \mathbf{F}\mathbf{c} = \mathbf{0} \tag{4-14b}$$

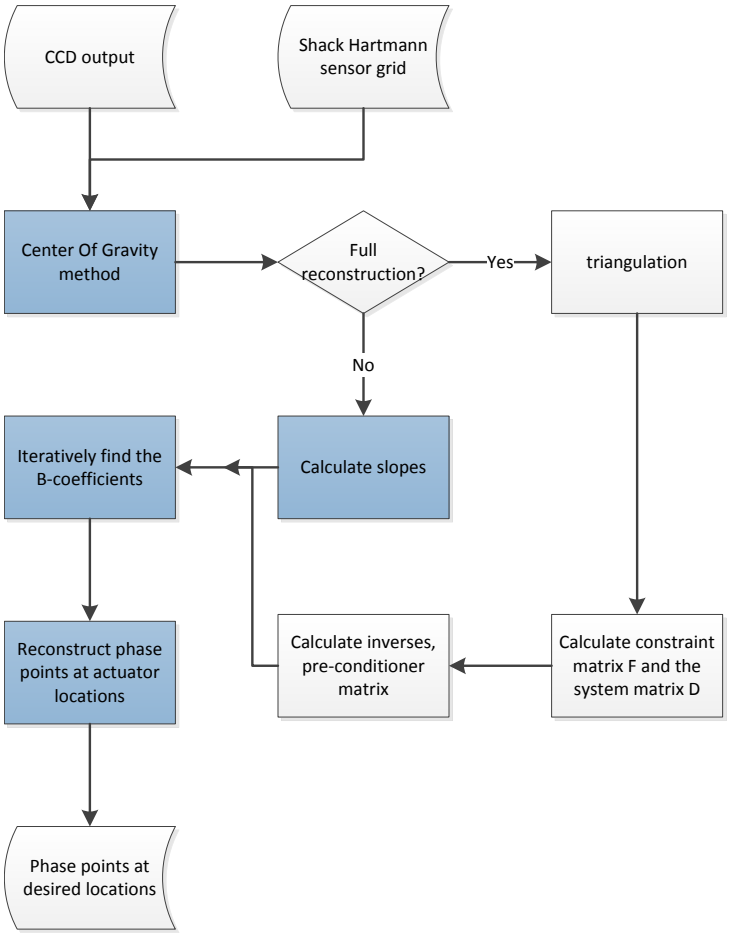


Figure 4-6: Flow diagram of wavefront reconstruction using the SABRE method, in purple are the steps that need to be done on line, full-reconstruction is only required if the locations of the sub-apertures change substantially

Usually such a problem is solved with either the Lagrange multipliers or the Null-space method. Both methods will be discussed in the next two sections.

4-5-1 Solving SABRE using Lagrange multipliers

The Lagrangian is given by

$$L(\mathbf{x}, \mathbf{y}) = ||\mathbf{D}\mathbf{c} - \mathbf{s}||_2^2 + \mathbf{y}^{\mathrm{T}}(\mathbf{F}\mathbf{c}),$$

where $\mathbf{y}$ is the vector with the Lagrange multipliers and the other variables are given in section 4-4. To find the solution the Lagrangian needs to maximized with respect to the Lagrange multipliers and to be minimized with respect to the B-coefficients. This is done by setting

$$\frac{\partial L}{\partial \mathbf{c}} = \frac{\partial L}{\partial \mathbf{y}} = 0.$$

Evaluating the derivatives gives

$$\underbrace{\begin{bmatrix} \mathbf{D}^{\mathrm{T}}\mathbf{D} & \mathbf{F}^{\mathrm{T}} \\ \mathbf{F} & \mathbf{0} \end{bmatrix}}_{\mathbf{A}_{\mathrm{lm}}} \underbrace{\begin{bmatrix} \mathbf{c} \\ \mathbf{y} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{D}^{\mathrm{T}}\mathbf{s} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{b}_{\mathrm{lm}}}.$$

The $\mathbf{A}_{\mathrm{lm}}$ matrix is a saddle-point matrix. There is a lot of literature about solving these saddle point problems. The reader is referred to for example [5]. Note that $\mathbf{D}^{\mathrm{T}}\mathbf{D}$ is semi-positive definite, contrary to the often made assumption that the upper left matrix of the saddle point matrix is positive definite. This makes for example preconditioning of the $\mathbf{A}_{\mathrm{lm}}$ difficult.

4-5-2 Solving SABRE using the Null-space method

For the SABRE problem the null-space method corresponds to fitting the slopes in the null-space of the constraints. This results in the following problem:

$$\mathbf{z} = \arg\min_{\mathbf{z}} ||\mathbf{D}\mathbf{N}_{\mathrm{F}}\mathbf{z} - \mathbf{s}||_2^2 \tag{4-15a}$$

$$\hat{\mathbf{c}} = \mathbf{N}_{\mathrm{F}}\mathbf{z} \tag{4-15b}$$

$\mathbf{N}_{\mathrm{F}}$ is the null-space of the constraints, $\mathbf{F}$ and can be computed using the Householders QR algorithm or can be calculated analytically as will be disussed in section 4-6. Only for zero order continuity constraints the null-space matrix, $\mathbf{N}_{\mathrm{F}}$ is garantueed to be sparse. This is because zero order continuity do not overlap with each other as can be seen in figure 4-4, which makes that constraints can be grouped in sets of constraints that are independent from other groups. For higher order continuity constraints, it will be more difficult to obtain a sparse null-space. An advantage of higher continuity constraints is that the null-space is smaller. From now on it is assumed that linear B-splines with zero order continuity constraints are used, Null-space Spline based ABeration REconstruction zero order constraints (NSABRE0).

Assumption To make the least-squares optimization problem strictly convex, it is assumed that the function, determined by the B-spline functions, for each triangle is fully determined by the slope measurements except for his piston:

$$\text{For } i = 1..\mathrm{N} \qquad \text{rank } \mathbf{D}\left((i-1)\hat{\mathrm{d}}{+}1{:}i\hat{\mathrm{d}}, (i-1)\hat{\mathrm{d}}{+}1{:}i\hat{\mathrm{d}}\right) = \hat{\mathrm{d}} - 1, \tag{4-16}$$

with $\hat{\mathrm{d}}$ the number of B-coefficients for each simplex and N the number of simplices. It is with minus one subtracted because of the unknown piston mode for each triangle.

4-6 Analytical calculation of the null-space matrix of N_F

The analytical description of the null-space matrix of $\mathbf{N}_\mathrm{F}$ can be obtained by using the consensus and sharing method. Assume that our goal is to apply the continuity constraints. If the simplices are seen as local problems then they have to reach consensus about the B-coefficients with their neighbour simplices. As discussed in section 4-3 each B-coefficients has a unique location. Consensus is reached by letting all B-coefficients be the mean value of the B-coefficients that have the same B-coefficient location. Now the variable $\mathbf{z}$ can be introduced, which contains the mean value of these B-coefficients and is also the null-space vector of $\mathbf{N}_\mathrm{F}$. $\mathbf{z}$ are the degree of freedoms left, after applying the continuity constraints. See also figure 4-7. The constraint

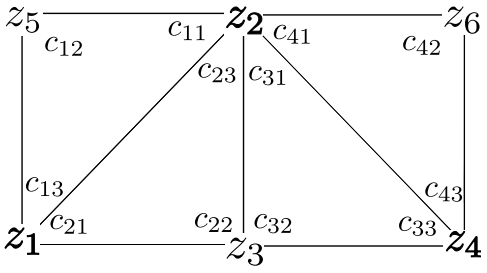


Figure 4-7: Visual representation of the Null-space vector $\mathbf{z}$ of $\mathbf{H}$.

matrix can now be split up, only valid for zero order continuity constraints:

$$\mathbf{Hc} = \mathbf{0} \equiv \mathbf{c} - \mathbf{M}_\mathrm{H}\mathbf{z} = \mathbf{0}, \tag{4-17}$$

with $\mathbf{H}$ the constraint matrix as defined in equation (4-4). This results in the observation that $\mathbf{M}_\mathrm{H}$ is the null-space of the $\mathbf{H}$ matrix. $\mathbf{M}_\mathrm{H}$ links the mean value at each B-coefficients location and can analytically be obtained by

$$\mathbf{M}_\mathrm{H}(i,j) = \begin{cases} 1 & \text{if } c_i \text{ is located at B-spline location } z_j \\ 0 & \text{else} \end{cases}. \tag{4-18}$$

Using the definition of $\mathbf{M}_\mathrm{H}$ given in equation (4-18) the anchor constraint, as discussed in section 4-4-1, is not satisfied. To satisfy the anchor constraint, one removes one variable of the $\mathbf{z}$ vector and removes the corresponding column. By doing this, you force one group of shared B-coefficients to be zero. If $\mathbf{M}_\mathrm{H}$ has n columns then the analytical null-space of $\mathbf{F}$ is given by

$$\mathrm{null}\,(\mathbf{F}) = \mathbf{M} = \mathbf{M}_\mathrm{H}(:, 1{:}n).$$

NSABRE0 can be solved by solving a set of linear equations

$$\underbrace{\mathbf{M}^\mathrm{T}\mathbf{D}^\mathrm{T}\mathbf{D}\mathbf{M}}_{\mathbf{A}_\mathbf{M}}\underbrace{\mathbf{z}}_{} = \underbrace{\mathbf{M}^\mathrm{T}\mathbf{D}^\mathrm{T}\mathbf{s}}_{\mathbf{b}_\mathbf{M}}, \tag{4-19}$$

with the B-coefficients then given by

$$\hat{\mathbf{c}} = \mathbf{M}\mathbf{z}.$$

Finite difference methods There is resemblance between solving finite differences and NSABRE0, since $\mathbf{z}$ represents values of a function on a grid and since linear B-splines are used, the differential equation is solved using finite differences. Well studied is the Poisson equation, although our method uses a different finite difference model, it is expected that the methods studied for the Poisson equation, will lead to the same performance as for solving NSABRE0.

From solving the Poisson equation, it is well known that the multigrid solver scales very well. Using the Algebraic Multigrid (AMG) Preconditioned Conjugate Gradient (PCG) it is expected to solve NSABRE0 with close to linear computational time complexity. For an explanation of AMG PCG and a more thou rough disscussion of how the NSABRE0 can be solved both sequential and distributed the reader is referred to chapter 6.

4-7 SABRE compared to other wavefront reconstruction methods

It was discussed in chapter 3 that the wavefront methods could be divided into two categories zonal methods and global methods. Sub-classes of the global methods are the transformation to a different domain methods, the radial basis functions and polynomial functions. The zonal methods and the transformations methods have the lowest computational complexity in time. The zonal methods because of their local structure and the transformation methods because of the fact that integrating in a different domain can be very straightforward. SABRE is the only method out of those that has not the disadvantage of having restrictions on the wavefront sensor grid. SABRE combines the local property of the zonal methods together with the radial basis function that don't depend on a grid. For convenience, several methods are compared in table 4-1.

Table 4-1: Comparison of Wavefront reconstruction methods

	Zonal methods		Modal methods				
	Finite Differences	SABRE	Polynomial functions		Transforms		Radial basis functions
			Zernike polynomials	Karhunen Loeve functions	Haar wavelets	Fourier domain	
Grid shape	square	any shape	circular	circular	square	circular	any shape
grid	yes	no	no	no	yes	yes	yes
Computational time complexity, for N measurements.	$\mathcal{O}(N)$ using multigrid preconditioned conjugate gradient [21]	$\mathcal{O}(N)$ using Null-space method with algebraic multigrid preconditioned conjugate gradient [a]	$\mathcal{O}(N^2)$ using pseudo inverse	$\mathcal{O}(N^2)$ using pseudo inverse	$\mathcal{O}(N)$ Haar wavelet transform	$\mathcal{O}(N\log N)$ Fourier transform [26]	$\mathcal{O}(N^2)$ using pseudo inverse
Special properties	Fried geometry; waffle mode insensitive, Southwell geometry; harder to calibrate the sensor	estimates local functions	compromise between approximation power and stability	compromise between approximation power and stability, has a diagonal covariance matrix			

[a] expected, not yet proven, see section 4-6. Holds only for zero order continuity constraints

Chapter 5

GPU programming and CUDA

The last decade the number of transistors in a Central Processing Unit (CPU) and in a Graphics Processing Unit (GPU) did follows Moore's Law; every two years the number of transistors used in IC's doubles. Though this was not the case for the clock speed, the clockspeeds started to stagnate. This results in an increasing performance gap between GPU's and CPU's. With the introduction of Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL), GPU's are capable to do scientific calculations in a relatively easy way. Therefore, it is worthwhile to optimize wavefront reconstruction algorithms for the GPU. An example of the use of GPU's is the Hale Telescope at the Palomar Observatory [57], they use several NVIDIA 8800GTX GPU's for the wavefront reconstruction. The Shack Hartmann consists out 64x64 sub apertures. They achieve a wavefront reconstruction of up to 2 kHz.

5-1 Introduction to CUDA

CUDA, let you use the CUDA NVIDIA GPU's to do numerical computations. There are extensions for C/C++ and Fortran. Assuming the C programming language is used, the code can be divided into a part that runs on the host, the CPU, and and a part that runs on the device, GPU. The device code are called kernels, and their associated data, structures. The

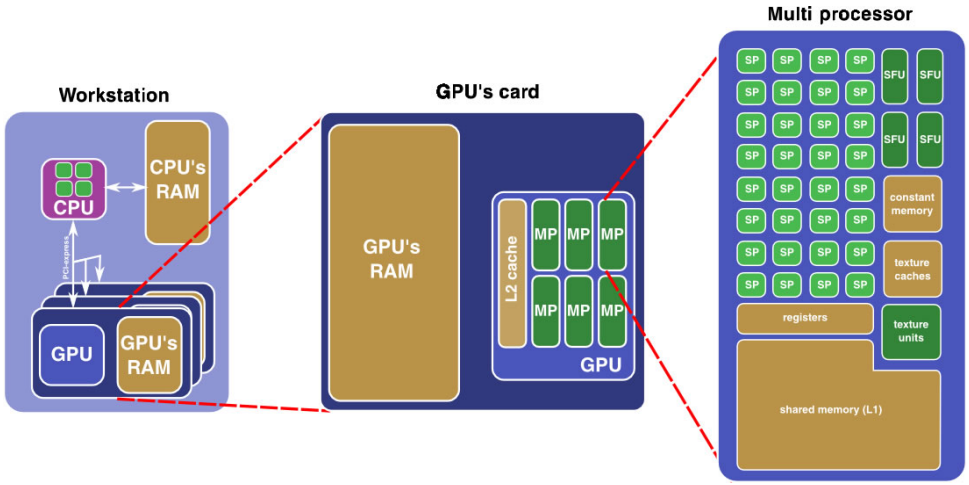


Figure 5-1: Architecture of the GPU in relation to the workstation *Aurélien Plyer* http://www.aurelien.plyer.fr/wp-content/uploads/2012/01/aGPUWorstation.png

GPU is an Single Instruction, Multiple Data (SIMD) processor. Meaning that all processors execute the same instruction for a large number of data. The GPU has a number of streaming multi-processors. Each of these multi-processors has Scalar Processor (SP) also called CUDA cores, usually 8, in figure 5-1.

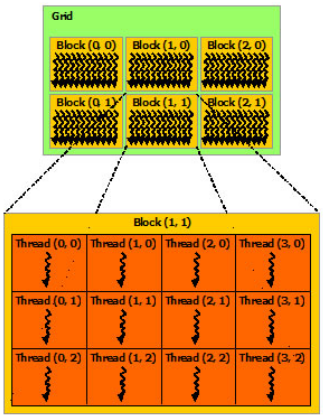


Figure 5-2: Threads and blocks *NVIDIA*
http://docs.nvidia.com/cuda/cuda-c-programming-guide/

5-1-1 Internal organization

A computational problem is split up in sub problems wich can be calculated parallel. Each sub problem is then transformed to a thread, which is a sequence of instructions that can be independently done on a SP. The main problem is then divided into a grid of 3D blocks, each containing the same number of threads, figure 5-2. Each block is divided into warps of 32 threads, if available.

Scheduling of threads

The multi-processor execute the blocks, one by one and then warp for warp. The idea of warps is, that if one thread within a warp is still waiting on data, another warp can be scheduled.

Memory

The GPU has various types of memory at his disposal, each with their own access times and characteristics.

Register memory Registers are exclusive to each thread, they have the lowest access times of all the memory types.

Shared memory After the register memory it has the lowest access times. It is available for all threads within a block. Part of the shared memory is used as a cache.

Texture memory This is read-only memory, available to all blocks within a single multi-processor.

Global memory It is the slowest memory and biggest memory of all the memory types within the GPU. The global memory is available to all multi-processors.

5-2 Coarse grain and fine grain parallelism

In this section it will be discussed what coarse grain and fine grain parallelism is and how it affects the GPU. In parallel computing, granularity is a qualitative measure of the ratio of computation to communication [7], see also figure 5-3. Two things are import here, that is the

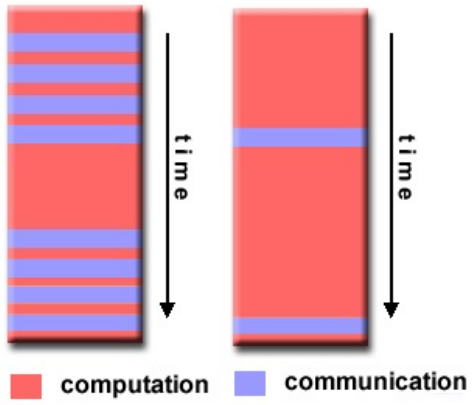


Figure 5-3: Left, granularity for fine grain parallelism and right, granularity for for coarse grain parallelism [7]

load balance and the total time required for communication. In a course grain algorithm, the times of communication is minimized, to reduce overhead in communication. In a fine grain algorithm the communication happens more frequently to optimize the load balance. There is a trade off between load balancing and total communication time. In a cluster of workstations or super computers the communication, is such a bottle neck that coarse grain parallelism is often preferred.

5-2-1 GPU and fine grain parallelism

In the case of the GPU, the communication between the cores is less of a bottleneck, which makes the GPU suitable for fine grain parallelism. By taking a look at the GPU again, one can have an idea of how this works. In section 5-1 the CUDA cores were introduced, these cores are totally different than the ones found in a CPU and can do only scalar operations. The idea is now to divide a computational problem into threads that can be run on the CUDA cores, since these cores can handle only small problems, communication must happen frequently, this corresponds to fine grain parallelism and is the main idea behind CUDA.

Domain decomposition on the GPU The threads corresponds to fine grain parallelism. Remember from section 5-1-1 that the problem is not only divided into threads but also in blocks. These blocks correspond to domain decomposition, course grain parallelism. This will become more clear in the example in the next section.

5-3 Matrix multiplication example

Four example will be shown of how matrix multiplication can be done in CUDA. The first two examples are copied from [33].

Matrix multiplication using a single block In the first example, the matrix multiplication problem is shown. *Md* and *Nd* being the matrices to be multiplied and *Pd* is the result of the multiplication. Assuming all matrices are square matrices with size *Width*. The matrix multiplication can now divided into *Width* times *Width* problems. Each problem calculates one entry of the result matrix *Pd*. The complete problem can now be divided into a 2D dimensional block. The threads are identified by *tx* and *ty*, which gives the location in the block. For convenience, *tx* and *ty* will also be used as location in the *Pd* matrix. Since all computations at every data piece must be the same, only for one subproblem the computations have to described as can be seen in the code 5.1, which gives the algorithm to calculate one element of the to be computed matrix *Pd*. When calling the function, one also gives as input the blocksize and the number of blocks. Current example uses a single block. The blocksize is limited to 512 or 1024, which means that for large matrices the problem must be divided into multiple blocks which will be discussed in the next section.

Listing 5.1: Matrix multiplication example

```
1  //matrix multiplication kernel - thread specification
2  __global__void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width)
3  {
4     //2D Thread ID
5     int tx = threadIdx.x;
6     int ty = threadIdx.y;
7
8     //Pvalue stores the Pd element that is computed by the thread
9     float Pvalue =0;
10
11    for (int k=0;k<Width; ++k)
12    {
13       float Mdelement = Md[ty*Width+k];
14       float Ndelement = Nd[k*Width+tx];
15       Pvalue+= Mdelement*Ndelement;
16    }
17    // Write the matrix to device memory each thread writes one element
18    Pd[ty*Width+tx]=Pvalue;
19 }
```

5-3-1 Matrix multiplication using multiple blocks

In this section matrix multiplication will be discussed using multiple blocks. Each block is a submatrix of the to be multiplied matrices; *Md* and *Nd*. The row and column of *Pd* is now also determined by block location in the grid as can be seen in the code 5.2.

Listing 5.2: Matrix multiplication using multiple blocks example

```
__global__void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width)
{
   // Calculate the row index of the Pd element and M
   int Row=blockIdx.y*TILE_WIDTH+treadIdx.y;
   // Calculate the column index of Pd and N
   int Col = blockIdx.x*TILE_WIDTH+threadIdx.x;

   float Pvalue=0;
   //each thread computes one element of the block submatrix
   for (int k=0;k<Width;++k)
      Pvalue +=Md[Row*Width+k]*Nd[k*Width+Col];

   Pd{Row*Width+Col]=Pvalue;
}
```

5-3-2 Matrix multiplication using libraries

Libraries can be used to do many mathematical operations. For example the NVIDIA cuSPARSE and CUDA Basic Linear Algebra Subroutines (cuBLAS) can be used to do matrix operations, respectively sparse matrix operations. In the CUSP library several Krylov subspace methods can be found. An example is given for a matrix multiplication using the cuBLAS library. Only one line of code is required to do the multiplication.

```
//C=A*B
//additional options can be set, for example to use the transpose of A.
opt8 = cublasSgemm(opt1, opt2, opt3, height(A), width(B),width(C) ,opt4, A,
    opt5, B, opt6,opt4, C, opt7);
```

5-4 Centroid algorithm on the GPU

As a last example, it will be shown how CUDA can be used to calculate the slopes of the wavefront. As discussed in section 2-4-2, light that goes through the sub-apertures creates for every sub-aperture a spot at the CCD sensor. Using the center of gravity method on local square grids of the CCD sensor, the location of each spot can be determined. To calculate spot locations from the CCD output, a center of gravity method is used. Since CCD noise is not zero mean the centers locations will be biased. To limit this effect the Thresholding Center of Gravity (TCoG) is used [56]. It basically extends the center of gravity method by taking CCD output values under a certain threshold not into account.
To obtain the slopes, one is interested in the difference between the locations of the spot in the ideal case of a flat wavefront and the measured wavefront. It is assumed that the locations, $centers_0$, of the spots, in the ideal case are known. The threshold that will be used will be different for each lenslet and is calculated by:

$$T = \beta\left(\max(\mathrm{CCD}_i) - \min(\mathrm{CCD}_i)\right) + \min(\mathrm{CCD}_i)$$

With β a user defined value between 0 and 1.

The GPU is very suitable to do determine the slopes of the wavefront using the Shack Hartmann sensor, since each slope can be calculated in parallel and the problem size is big enough to make the transition to the GPU worthwhile. The TCoG is implemented on a GPU using CUDA. Each thread calculates a slope and does his calculations on a subarray of the CCD given by a grid. The following calculations are done by each thread on a different subarray of the CCD sensor:

1. Calculate minimum and maximum of the subarray.
2. Determine threshold.
3. Calculate moments of the center of gravity method.
4. Calculate the slopes.

The corresponding code is given by code 5.3.

Listing 5.3: TCoG parallel

```
__global__ void Tcog(float* CCD, float* Grid,int N,int widthCCD,float* min,
    float* max,float percent,float* level,float* sumx,float* sumy, float* slopes
    , float minRefX, float  minRefY,float  rangeRefX,float  rangeRefY,float*
    refCenters,float alphax,float alphay,float* mass,float val)
//Important either GRID or both minRefY, minRefX should be added with 1,
    because of different array indexing MATLAB and C
//slopes are normalized according to wfr_ShackHartmannSim2 in the SABREdemo

/*input*/
//CCD,widthCCD   the CCD array with width widthCDD
//Grid   array with size [number of lenslet,4] every row contains the x,y
    coordinate of the lower bottom corner and upper higher corner of each
    CCDasubarray
//N,percent,val   number of lenslets, beta, largest floatingpoint number
//minRefY,minRefX,rangeRefX,rangeRefY,refCenters,slopes   see
    wfr_ShackHartmannSim2 in the SABREdemo
//alphax,alphy   2*pi/lambda*Nlenslets*Dlenslet*rangeRefX(Y)*dx_focus/flenslet
    see  wfr_ShackHartmannSim2 in the SABREdemo

/*output*/
//min,max   vector with size N, minimum and respectively maximum value of each
    subCCDarray
//level   threshold of each CCDsubarray
//sumx,sumy,mass   sum of the center of gravity in the x respectively y
    direction and the sum of total values for  every CCDsubarray

{
  int i;
  int j;
  int lensLetId=blockDim.x*blockIdx.x+threadIdx.x;
  if (lensLetId<N)
  {
    //initialize
    min[lensLetId]=val;
    max[lensLetId]=0;
    sumx[lensLetId]=0;
    sumy[lensLetId]=0;
    mass[lensLetId]=0;
    //find maximum and maximum of each CCD
    for (i=Grid[lensLetId*4];i<Grid[lensLetId*4+1];i++)
    {
      for (j=Grid[lensLetId*4+2];j<Grid[lensLetId*4+3];j++)
      {
        if(CCD[i*widthCCD+j]>max[lensLetId])
          max[lensLetId]=CCD[i*widthCCD+j];
        if(CCD[i*widthCCD+j]<min[lensLetId])
          min[lensLetId]=CCD[i*widthCCD+j];
      }
    }
    //obtain threshold
    level[lensLetId]=(max[lensLetId]-min[lensLetId])*percent+min[lensLetId];
    //calculate spot location
    for (i=Grid[lensLetId*4];i<Grid[lensLetId*4+1];i++)
    {
      for (j=Grid[lensLetId*4+2];j<Grid[lensLetId*4+3];j++)
      {
        if(CCD[i*widthCCD+j]>level[lensLetId])
        {
```

```
49              sumx[lensLetId]=sumx[lensLetId]+CCD[i*widthCCD+j]*(i-Grid[lensLetId
                    *4]);
50              sumy[lensLetId]=sumy[lensLetId]+CCD[i*widthCCD+j]*(j-Grid[lensLetId
                    *4+2]);
51              mass[lensLetId]=mass[lensLetId]+CCD[i*widthCCD+j];
52
53          }
54
55        }
56      }
57      //calculate slopes
58      slopes[2*lensLetId]=alphax*((Grid[lensLetId*4]+sumx[lensLetId]/mass[
            lensLetId]-minRefX)/rangeRefX-refCenters[2*lensLetId]);
59      slopes[2*lensLetId+1]=alphay*((Grid[lensLetId*4+2]+sumy[lensLetId]/mass[
            lensLetId]-minRefY)/rangeRefY-refCenters[2*lensLetId+1]);
60    }
61  }
```

It was decided that each thread does calculate one slope. While this results in enough parallelism, the the number of variables for each thread is very large which decreases the occupancy of the GPU. Therefore it is recommended to increase the number of threads by splitting the calculations of one slope into subproblems.

5-5 Parallel Sparse Matrix vector multiplication on the GPU

In this section it is discussed how one can do sparse matrix multiplication on the GPU. After introducing different storage formats it is discussed how matrix vector multiplication works and what the limitations are.

Storing sparse Matrices

To reduce storage, it is possible to store sparse matrices in a different way than the standard 2D array. A detailled overview can be found in [4].

DIA

In the case that the matrix is banded. The matrix can efficiently stored by storing the diagonals.

COO

COO is a coordinate list of the values, (row indice, column indice, value). It works very well for matrix reconstruction.

CSR and CSC

Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC). 3 one dimensional arrays are used to store the matrix. For CSR this is ([values, row by row],[column indice],[row pointers]). The row pointers are used to specify in the array *values*, where the next row starts. For CSC it is defined as: ([values, column by column], [row indice], [column pointer]). CSR and CSC are very suitable for sparse matrix-vector multiplication. An example for CSR:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 8 & 3 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 5 \end{bmatrix} \rightarrow \begin{array}{lllllll} 0 & 1 & 4 & 6 & 7 & & \text{row offsets} \\ \hline 1 & 1 & 2 & 3 & 2 & 4 & 4 \quad \text{column indices} \\ \hline 2 & 1 & 8 & 3 & 1 & 2 & 5 \quad \text{values} \end{array}$$

ELLPACK

Consists out of two 2D arrays, one with on each row the column indices to the corresponding row and one with the values on each row for the corresponding row. It can be combined with COO into a **hybrid** format.

BCSR

Blocked Compressed Sparse Row (BCSR) format, instead of values, the sparse matrix is divided in blocks.

Matrix free format

In case the matrix is only used to reorder a vector, using matrix vector multiplication, then it can be stored in only one array. Each element of the array, contains the indice, where the corresponding element should be stored.

Sparse matrix vector multiplication and communication

One is interested in to computing the following sparse matrix vector multiplication:

$$\mathbf{y} = \mathbf{A}\mathbf{x},$$

with $\mathbf{A}$ a sparse matrix. The current most widely sparse matrix formats are CSR and ELLPACK. In that case, each core on the GPU calculates one element $\mathbf{y}_i$ by multiplying one row with a vector, $\mathbf{A}_i\mathbf{x}$. If there are many nonzeros on a row, then it might be advantageous to also split the row vectors of $\mathbf{A}$ and to split the $\mathbf{x}$ vector. It is assumed that this is not the case. According to section 5-1, The matrices are split up in blocks of multiple rows, each block contains preferable a multiple of 32 rows, and each block is split up in warps of 32 rows. Then the data required to do matrix vector multiplication for that block, is stored in shared memory. As soon the data for one warp is loaded, that warp is scheduled and calculated. It is important that the elements of $\mathbf{x}$ required for one block are coalesced and preferable, also reused in multiple rows. whether this is achievable, highly depends on the structure of the matrix.

5-6 Latency hiding, calling kernels and the problem size

In this section it will be discussed how latency hiding can be used to fully utilize the GPU. It will also be disscussed what influence of calling a kernel is on the computation time. Both are in a certain way related to the problem size.

5-6-1 Latency hiding

What is latency hiding? As discussed in section 5-1-1, the warp scheduler, schedules the warp for which the data is already available. Which means that one does not need to wait until all data is loaded for all warps. In this way latency because of loading data can be partially hidden. But can one can imagine that the number of threads, meaning more warps, of a problem must be far larger than the amount of scalar processors, to be able to hide the latency.

5-6-2 Calling kernels

The execution of a kernel is an expensive operation. The call of a kernel can even have a duration up to a few micro-seconds. This is because the execution off the calls are done on

the host, which causes unnecessarily datatransfer between host and device. The new generation NVIDIA graphics card support dynamic parallelism, which makes it possible to call kernels from the device. The kernel calls are asynchronous, meaning that that they don't have to wait for the previous kernel(s) to be finished. This means they can run parallel with the computations on the GPU. When threads are synchronized or memory is copied, kernel calls are synchronous, meaning that the next kernel call has to wait for the previous kernel to be finished. If one runs many small problems, problems with low computational time, then the time required for calling the kernels might be higher than the computational time of each kernel itself. Therefore especially on older GPUs one must avoid the computation of multiple kernels in sequence which are in number of threads small and not computational very expensive.

5-7 Using multiple GPUs

For very large problems, 1 GPU might be insufficient for your problem. Using multiple GPUs have some implications. Fine grain parallelism, that is possible to use on single GPU is not possible for multiple GPUs. The communication will be a bottle neck and the times that the GPUs communicate with each other should be minimized. The course grain methods in that case have to be used, for example domain decomposition. The fine grain parallelism has to be combined with coarse grain parallelism. Examples of the conjugate gradient on multiple GPUs can be found in [61, 8].

5-8 Wavefront reconstruction: CPU or GPU?

An interesting question would be whether it is beneficiable to do wavefront reconstruction on the GPU. Assume that for the wavefront reconstruction, a Shack Hartmann sensor is used, combined with either Spline based ABeration REconstruction (SABRE) or the finite difference method. The center of gravity method to calculate the slopes from the CCD sensor, was already discussed in section 5-4. It is very suitable for the GPU due to the large amount of parallelism and having no data dependencies between the different threads. Sending the CCD data to the GPU can be done in parallel with the computations but will still result in an extra delay which must be taken into account. Doing the numerical integration of the slopes on the GPU is a bit more cumbersome. The amount of communication is large in sparse solvers, which decrease the efficiency of the GPU significantly. Also the calling of many kernels in an iterative method can slowdown the GPU. But it still interesting and worthwhile to see the difference between CPU and GPU in real experiments.

5-9 Conclusion

The GPU can be used to speed-up scientific computations. The GPU is very suitable for large problems, due to the large amount of cores. The next two points, summarizes the main differences between using cluster of processors and using the GPU.

- the GPU has a SIMD structure. One not thinks not in tasks but in data, therefore everything is as much vectorized as possible.
- The GPU is suitable for fine grain parallelism. Instead of minimizing the times of communication, the duration of communication is minimized by coalescing memory access as much as possible, for example by optimizing the structure of sparse matrices.

- To efficiently use the GPU, computational problems have to have either to contain many threads or to be computational expensive, this in order to not have to suffer from overhead of callings kernels or to benefit from latency hiding.

Also it was discussed that although the transition from CPU to GPU might not be beneficial. Though, it would be interesting to see this in some real experiments.

Chapter 6

SABRE in a distributed way

6-1 Intuition and distributed SABRE

In this section it will be discussed what our intuition says about how Spline based ABeration REconstruction (SABRE) can be parallel solved. We recall the SABRE problem from equation (4-12):

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}} f(\mathbf{c}) \, ||\mathbf{D}\mathbf{c} - \mathbf{s}||_2^2 \tag{6-1a}$$

$$\text{subject to} \quad \mathbf{F}\mathbf{c} = \mathbf{0} \tag{6-1b}$$

First starting with fitting the slopes to the simplices represented by equation (6-1a). This can be done completely parallel. This can be done for each simplex parallel as can be seen in figure 6-1.

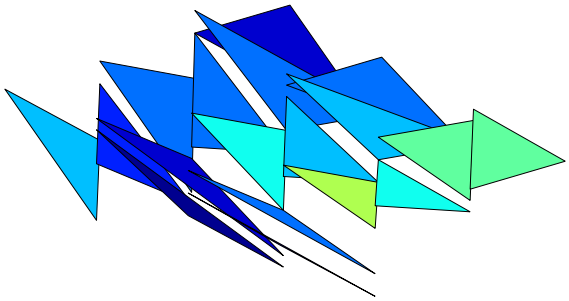


Figure 6-1: Example of reconstructing the slopes without applying the continuity constraints

To investigate the continuity constraints given by equation (6-1b) a look is taken at figure 4-4. It can be observed that for zero order continuity constraints, the set of constraints acting on location can be computed parallel. For higher order of continuity, this is not the case since constraints acting between the same two simplices will not only influence the same set of B-splines but also different B-spline functions. Although the constraints for zero order continuity can be computed parallel, this does not make SABRE parallel since each zero order constraint

uses data from multiple simplices. Based on these observations the following suggestions will be done to develop a parallel SABRE algorithm.

- Depending on the actual algorithm it might be beneficial to restrict ourselves to use only zero order constraints, since without the fitting of slopes they are the only order that can be computed parallel.
- Split SABRE into two parts: fitting of slopes and continuity constraints, since separately, one of the two can be computed parallel.

The logical choice is the null-space method, with zero order constraints. This corresponds to Null-space Spline based ABeration REconstruction zero order constraints (NSABRE0). It will be shown how NSABRE0 can be solved in a distributed way. The NSABRE0 is partially already discussed in section 4-5-2.

6-2 Null-space method

The Null-space is partially already discussed in section 4-5-2. For the SABRE problem the null-space method corresponds to fitting the slopes in the null-space of the constraints. This results for linear B-splines and zero order continuity constraints in solving the following system of linear equations:

$$\underbrace{\mathbf{M}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{D}\mathbf{M}}_{\mathbf{A_M}}\underbrace{\mathbf{z}}_{} = \underbrace{\mathbf{M}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{s}}_{\mathbf{b_M}}, \tag{6-2}$$

with the B-coefficients then given by

$$\hat{\mathbf{c}} = \mathbf{M}\mathbf{z}.$$

6-3 Reordering of $\mathbf{A_M}$

It can be advantageous to reorder the $\mathbf{A_M}$ matrix. For example to obtain a more regular memory access pattern or to have a higher sparsity in the factors of the Cholesky factorization (CHOL). To obtain a matrix with minimum bandwith the Symmetric Reverse Cuthill-McKee (SRCM) ordering [11] can be used. To obtain a higher sparsity in the factors of the CHOL one can use a Minimum Degree (MD) algorithm. Reordering is possible using a transformation matrix $\mathbf{P}$. An example of the sparsity structure of both reorderings can be seen in figure 6-2a. Now equation (6-2) can be rewritten to

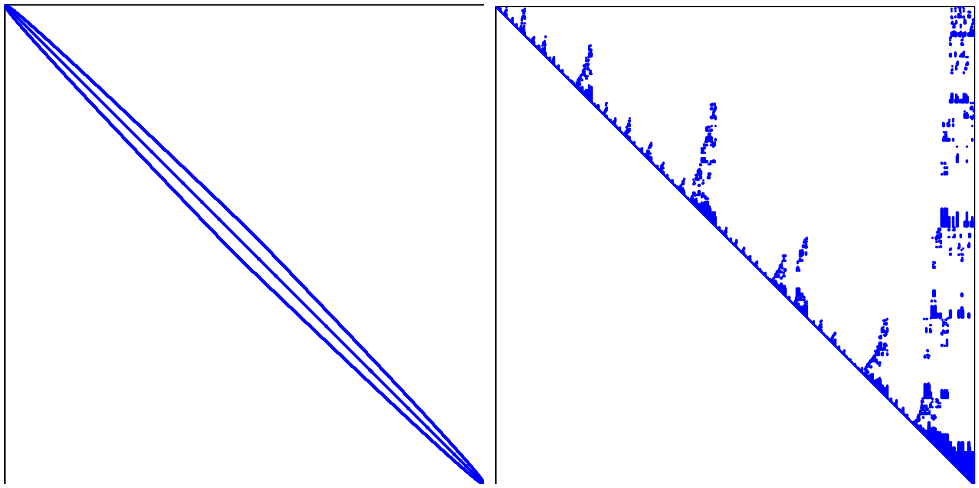
$$\underbrace{\mathbf{P}\mathbf{M}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{D}\mathbf{M}\mathbf{P}^{\mathrm{T}}}_{\tilde{\mathbf{A}}_\mathbf{M}}\underbrace{\tilde{\mathbf{z}}}_{\mathbf{P}\mathbf{z}} = \underbrace{\mathbf{P}\mathbf{M}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{s}}_{\tilde{\mathbf{b}}_\mathbf{M}} \tag{6-3}$$

and the B-coefficients are then given by:

$$\hat{\mathbf{c}} = \mathbf{M}\mathbf{P}^{\mathrm{T}}\tilde{\mathbf{z}}.$$

6-4 Solving NSABRE0 distributed with Cholesky factorization

The first distributed method that is discussed is to solve NSABRE0, equation (6-3), is with CHOL and two triangular solves.
The $\mathbf{A_M}$ matrix can be factorized by a CHOL. The $\mathbf{A_M}$ matrix is factorized by $\mathbf{L}^{\mathrm{T}}\mathbf{L}$. With $\mathbf{L}$ a lower triangular matrix. First $\mathbf{L}\mathbf{v} = \mathbf{b}$ is solved using forward substitution. After that $\mathbf{L}^{\mathrm{T}}\mathbf{z} = \mathbf{v}$ is solved using backwards substitution. It is advantageous to reorder the $\mathbf{A_M}$ matrix which is discussed in section 6-3.



(a) $\mathbf{A_M}$ after reordering with the SRCM ordering, 3053 nonzeros.

(b) Cholesky factorization $\mathbf{L}^{\mathrm{T}}$ after $\mathbf{A_M}$ reordering with MD algorithm, 6624 nonzeros.

Figure 6-2: sparsity structure after reordering of the B-coefficients with both matrix size 624x624, using 1152 triangles and a Shack-Hartmann grid of 25x25.

Table 6-1: required levels to do a triangular solve for different grid and reorderings of $\mathbf{A_M}$

	Shack Hartmann grid 10x10		Shack Hartmann grid 25x25		Shack Hartmann grid 50x50	
	levels	nonzeros	levels	nonzeros	levels	nonzeros
ICHOL(0) SRCM	18	277	54	1836	124	7493
ICHOL(0) MD	8	277	7	1836	10	7493
CHOL SRCM	98	785	622	11319	2498	87113
CHOL MD	30	629	100	6624	217	37761

Parallelism in the sparse triangular solver Two triangular systems must be solved. Can this be done in parallel? Yes this is possible. Finding the solution of triangular system is done in levels. Each level solves the variables that does not dependent on other variables or depend on already calculated variables. Depending on how much rows depend on each other, many levels or only a few levels are required. After a variable is solved, the rows that contain this variable are updated. A triangular matrix with only non-zero entries on the main diagonal can be solved in one level and has therefore the highest amount of parallelism. For more details the reader is referred to [37]. Of course it is interesting to see how many levels are required for NSABRE0 and whether this scales. We did did this for both the SRCM reordering of $\mathbf{A_M}$ and the MD ordering of $\mathbf{A_M}$.For the nested dissection algorithm it is shown that the amount of fill in for finite elemement solvers, the increase of nonzeros scales with $\mathcal{O}\left(N\log\left(N\right)\right)$ [20] where N are the number of nodes, which is proportional in the NSABRE0 case to the number of measurements. The MD algorithm scales similarly as discussed in [36].
For the Shack Hartmann sensor 50 by 50 grid, the number of rows per level are plotted in figure 6-3.
In table 6-1 it can clearly be seen that there is clear advantage in using the MD ordering above the SRCM ordering. The number of levels seems to scale slightly worse than $\sqrt{N}$ with N the number lenslets. In figure 6-3 it can be seen that the number of rows varies wildly per level. Also the amount of rows is low compared to N. Wich means that

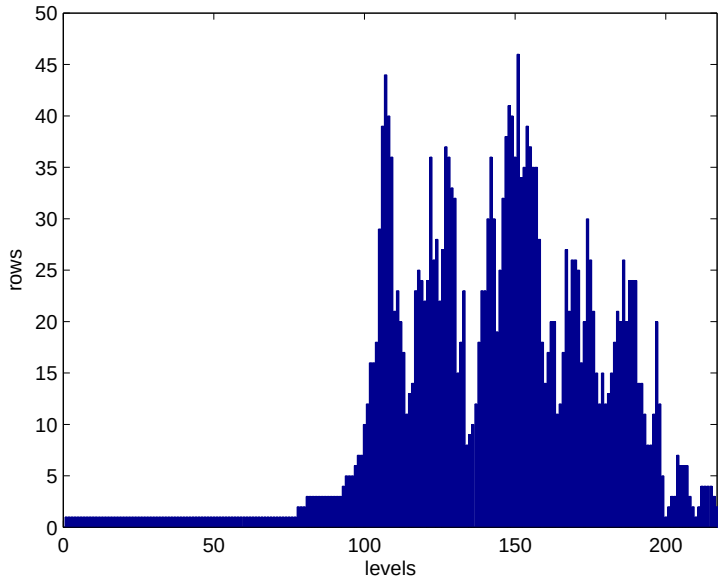


Figure 6-3: Number of rows per level, for CHOL with MD ordering and 50x50 lenslets

the amount of parallelism is limited. Sequential the computational complexity is given by $\mathcal{O}(rows * nonzerosperrow/column)$. For the MD case, the growth of nonzeros per column/row is $\mathcal{O}(\log(N))$ which leads to computational complexity of $\mathcal{O}(N\log(N))$. Parallel the computational complexity is $\mathcal{O}(levels * nonzerosperrow/column)$. For the SRCM case this results in $\mathcal{O}(N\log(N))$. Using the MD ordening a lower computational complexity order seems to be achievable, but unfortunately this could not be quantified. But is close to $\mathcal{O}\left(\sqrt{N}\log(N)\right)$. As a sequential solver the Cholesky solver is a very beautiful algorithm but the very limited parallelism makes it unsuitable as a parallel solver.

6-5 Solving NSABRE0 distributely with the conjugate gradient method

In this section it is discussed how the Conjugate Gradient (CG) can be used to solve the SABRE problem. The CG is introduced and different preconditioners are explained. It is addressed how they can be implemented parallel and how the different preconditioners scale. Furthermore it is explained that is beneficial to leave out the anchor constraint of NSABRE0 to improve convergence.

6-5-1 The conjugate gradient

In this section a short introduction to the CG is given. The CG method belongs to the class of Krylov subspace solvers. In the case that one want to solve the systems of equations $\mathbf{A_M z} = \mathbf{b_M}$, the ith Krylov subspace is given by all combinations of

$$\mathcal{K}_i = \mathbf{b_M}, \mathbf{A_M b_M}, ..., \mathbf{A_M^{i-1} b_M}.$$

The residual $\mathbf{r}_i$ after the ith iteration is orthogonal to the ith Krylov subspace. The CG method can also be seen as a method that recursively solves the energy of $\frac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{A_M z} - \mathbf{z}^{\mathrm{T}}\mathbf{b_M}$. Every iteration it mimimizes the energy in the ith Krylov subspace. In figure 6-4 the CG is compared

to the steepest descent method. It can be shown that in case of solving the Poisson equation

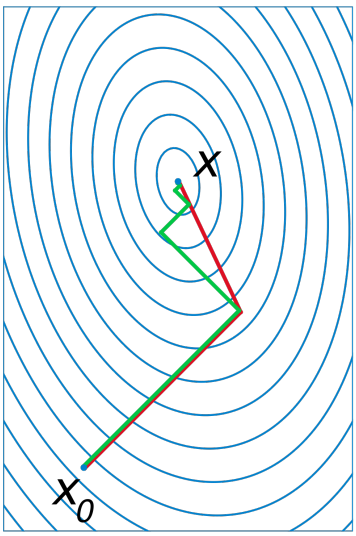


Figure 6-4: Steepest descent (red) compared to CG (green)
http://en.wikipedia.org/wiki/File:Conjugate_gradient_illustration.svg

the number of iterations scale with the square root of the condition number [14]. To improve the condition number, a preconditioner can be used. An algorithm for the Preconditioned Conjugate Gradient (PCG) will be given in the next section, also it will be discussed how it can be implemented parallel.

6-5-2 The preconditioned conjugate gradient

In this section a more thorough look is taken at the PCG and how it can be implemented parallel. Assuming the following system: $\mathbf{Ax} = \mathbf{b}$ with as preconditioner M. With $\mathbf{A}, \mathbf{x}$ and $\mathbf{b}$ in the case of NSABRE0 given by equation (6-4). The preconditioner M is closely related to the $\mathbf{A}$ matrix. The eigenvalues of $\mathbf{I} - M^{-1}\mathbf{A}$ determine how fast the iteration converges. In this section three preconditioners are considered. The Incomplete Cholesky factorization (ICHOL), Approximate INVerse (AINV) and the Algebraic Multigrid (AMG) preconditioner. It will be discussed what the computational complexity of the preconditioner is on the GPU. Assumed is again that there are N processors. Also it will be discussed how the number of iterations scale with the grid size. First the algorithm behind the PCG is shown. The algorithm is shown to solves a system.

function PRECONDITIONED CONJUGATE GRADIENT(M,A,x_0,b,ϵ) ▷ ϵ is used as a stopping criteria
- $k = 0$
- $r = b - Ax_0$
- Solve $Mz_k = r_k$
- $p = z$
- $\rho_0 = ||r||_2^2$
- **while** $\sqrt{\rho_k} > \epsilon \, ||b||_2$ **do**
  - $w = Ap$ ▷ Sparse vector matrix multiply
  - $\alpha_k = \frac{\rho_{k-1}}{p^T w}$ ▷ Inner product
  - $x = x + \alpha_k p$ ▷ vector addition
  - $r = r - \alpha_k w$ ▷ vector addition
  - Solve $Mz_k = r_k$ ▷ Preconditioning step

$\tilde{\rho}_k = r^T z$ ▷ Inner product
$\beta_k = \frac{\tilde{\rho}_k}{\tilde{\rho}_{k-1}}$
$p = z + \beta_k p$ ▷ vector addition
$\rho_k = ||r||_2^2$ ▷ Inner product
$k = k + 1$
end while
return x
end function

The most computational expensive parts are the preconditioning step, the sparse matrix vector multiply and three inner products. Since these steps require communication.
In our case $\mathbf{A}$ is very sparse and relatively well structured. Which mean that the inner product need to be taken into account. If the number of iterations is fixed or there is no preconditioner used then only two inner products are required. If the communication is ignored then the parallel computational complexity for 1 iteration, in case of N processors, is $\mathcal{O}(\log(N))$. With N the number of subapertures. The number of iterations is proportional on the square root of the condition number, which is properportional to the size of the $\mathbf{A}$ matrix. The required number of iterations is then proportional on the square root of the number of subapertures. That gives $\mathcal{O}\left(\sqrt{N}\log(N)\right)$ for the CG.

6-5-3 Cholesky preconditioner

The use of the CHOL to solve the NSABRE0 problem was already discussed in section 6-4. To use the CHOL as a preconditioner, the factorization is approximated with a factorization which is much more sparse than the original one. This is called ICHOL. This is done by dropping values below a certain threshold. The preconditioner matrix is then given by: $M = \mathbf{L}^{\mathrm{T}}\mathbf{L}$. Every iteration one solves the two triangular systems to come closer to the real solution. The reduction in iterations required to converge depends on the threshold the values are dropped. If one drops all values except the ones that are on the same locations as the nonzero entries in $\mathbf{A}$ then the reductions in iterations is reduced by constant, ICHOL(0). One can also use the Modified Incomplete Cholesky factorization (MICHOL), where dropped values are added to the main diagonal. It can also have the same sparsity as $\mathbf{A}$ and now the iterations required to converge drop with the square root of the normally amount of iterations to converge, based on the fact that the condition number of $\mathbf{A}$ drops with one order [15]. This works especially well for partial differential equations problems [51] , which NSABRE0 corresponds to. The computational complexity of the MICHOL(0) and ICHOL(0) PCG is given by the computational complexity of one iterations times the number of iterations to conververge. For the sequential case this is $\mathcal{O}\left(N\sqrt{N}\right)$ for ICHOL(0) and $\mathcal{O}\left(N\sqrt[4]{N}\right)$ for MICHOL(0). For the parallel case this, it is assumed that the number of levels scale logarithmic or better with N using MD reordering . This results in a computational complexity of $\mathcal{O}\left(\sqrt{N}\log(N)\right)$ for ICHOL(0). Unfortunately MD ordering could not be used for the MICHOL(0). It is unclear how the number of levels scale for MICHOL(0), the worst scase scenario is assumed $\mathcal{O}(N)$. The paralllel computational complexity is then $\mathcal{O}\left(N\sqrt[4]{N}\right)$.

6-5-4 Approximate inverse

The inverse of preconditioner matrix can also be calculated by calculating it explicitly. This can be computed in advance and the preconditioning step results in an sparse matrix vector multiplication which will have plenty of parallelism. But is a compromise between sparsity and how close the AINV approximates the real inverse, which is often dense. For larger grids, if one

does not want to increase the number of zeros of M more than with the same factor of which the number of $\mathbf{A}$ increases then the number of iterations drop only with a constant compared to the CG without preconditioner. So it does not improve the scaling of CG which leaves the computational complexity of the PCG to $\mathcal{O}\left(N\sqrt{N}\right)$. There are different approaches to obtain a sparse AINV. Most of the algorithms minimizes the Frobenius norm of

$$||M\mathbf{A} - \mathbf{I}||_{\mathrm{F}}$$

see for example [24]. An AINV that is very simple to compute and has the same sparsity of $\mathbf{A}$ can be found in [28]. It appromixates the inverse of the Symmetric Successive Over-Relaxation (SSOR) matrix.

6-5-5 Multigrid preconditioner

The main idea of multigrid is that: By changing the grid size, the smooth, low frequencies, become rough, high frequencies, winch will lead to much faster convergence. As a preconditioner one could also restrict to coarser grids. If one would only restrict to one coarser grid then the multigrid solver consist out of the following steps.

1. Smooth
2. Restrict to coarser grid
3. Solve the coarse grid
4. Interpolate
5. Postsmooth

The smoothing can be applied with a few Gauss Seidel or Jacobi iterations. To solve the coarse grid, one can also a few Gauss Seidel or Jabobi iterations to approximate a solution, but it also possible to apply a direct method. If it used parallel, Jacobi will be used, since that one is easily parallelizable. Often one uses multiple coarse grids. There are different ways in which order to go from finer grid to coarser grid, see figure 6-5 for details. It can be shown that it is beneficial to

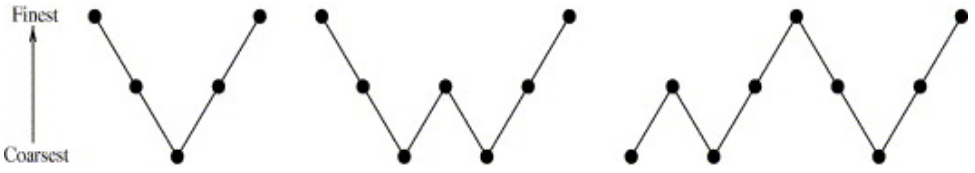


Figure 6-5: V-cycle, W-cycle and full multi-grid

stay on the coarse grid longer. The full multi-grid therefore performs best. Though, sequential, the multi-grid outperforms other methods it does not mean that it is also the fastest method if it is used parallel. Very sparse system are not easily parallelizable with multi-grid methods, which degrades performance [31, p. 606]. Another problem is that on course grid amount of parallelism is much lower than on finer grids, which is especially degrades performance for the W-cycle and the full multigrid cycle [3].
Since the grid of NSABRE0 can be unstructured the geometric multigrid will not work. In this case the coarse grids have to be created artificially by analyzing and graph clustering of the corresponding matrix, which is AMG. To use multigrid as a preconditioner, one goes through one multigrid cycle every CG iteration. The time for one iteration cost more than with standard multi-grid, but fewer iterations are required [62]. For a parallel implementation see [40, p. 94].

Computational complexity of the multi-grid preconditioned conjugate gradient In this paragraph the computational complexity of the AMG PCG is discussed. Each level deeper in the V-cycle is 4 times smaller than the previous level, similar for the Full multigrd cycle. For the full multigrid cycle a constant number of cycles are required to converge. For the V-cycle the number of cycles in the order of $\log(N)$. Using a geometric sum, it can be shown that sequential the computational cost is $\mathcal{O}(N)$, together with the number of required full multi-grid cycles being independent of N, the computational complexity seqequential is $\mathcal{O}(N)$. For the parallel case, the V-cycle is used. The number of levels in V-cycle are of $\mathcal{O}(\log(N))$. Then the computational complexity parallel is $\mathcal{O}\left(\log^2(N)\right)$. Assumed is that the AMG PCG only decreases the number of iterations by a constant factor. For convergence proof see for example [67].

6-5-6 Conjugate gradient and the anchor constraint

In section 4-4-1 the anchor constraint is added to the constraint matrix such that NSABRE0 as defined in equation (6-1) is strictly convex. For the CG this is actually not neccesary. If one can make sure that the solution is in the space that belongs to the strictly convex space of the problem then a convex problem can be solved using the CG. Since this is the case for NSABRE0, NSABRE0 can be solved without the anchor constraint by the CG method. It is actually a good idea not to use anchor constraint. With the anchor constraints, one B-coefficient, or for the null-space method a shared group of B-coefficients, is fixed to zero. If this group of B-coefficients, has to change his value in order to let the function on the connected triangle change to the slopes, all B-coefficients have to be shifted with this change in value to satisfy the anchor constraint. This is undesirable, therefore the anchor constraint is excluded. The $\mathbf{A_M}$ matrix and $\mathbf{b_M}$ in equation 6-2 are given by $\mathbf{M}_{\mathrm{H}}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{D}\mathbf{M}_{\mathrm{H}}$ and $\mathbf{M}_{\mathrm{H}}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{s}$, which changes equation to (4-19) to

$$\mathbf{P}\underbrace{\mathbf{M}_{\mathrm{H}}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{D}\mathbf{M}_{\mathrm{H}}}_{\mathbf{A_H}}\mathbf{P}^{\mathrm{T}}\tilde{\mathbf{z}} = \underbrace{\mathbf{P}\mathbf{M}_{\mathrm{H}}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{s}}_{\mathbf{b_H}}, \tag{6-4}$$

where $\mathbf{M_H}$ is defined in equation 4-18. The B-coefficients are then given by

$$\hat{\mathbf{c}} = \mathbf{M}_{\mathrm{H}}\mathbf{P}^{\mathrm{T}}\tilde{\mathbf{z}}.$$

6-6 Conclusion: solving SABRE parallel

It is discuss how NSABRE0 can be in theory solved in parallel using the CG method. Also preconditioners were discussed. An overview can be seen in table 6-2. Based on the communication and parallelism the AINV PCG is a good choice. Based on how the algorithm scales the AMG PCG method seems to be the most promising. In the next chapter, numerical experiments will be done, too see what is the best algorithm in practice.

Table 6-2: Computational complexity for different algorithms on CPU and GPU for NSABRE0, GPU has N processors with N the number of measurements

	computational complexity CPU	computational complexity one iteration GPU	required number of iterations to converge	computational complexity GPU	paral-lelism	commu-nication
Two triangular solves using CHOL [a]	$\mathcal{O}\left(N\log\left(N\right)\right)$			$\mathcal{O}\left(\sqrt{N}\log\left(N\right)\right)$ [b]	low	high
multigrid	$\mathcal{O}\left(N\right)$	$\mathcal{O}\left(\log\left(N\right)\right)$	$\mathcal{O}\left(\log\left(N\right)\right)$	$\mathcal{O}\left(log^2\left(N\right)\right)$	low	high
dense inverse	$\mathcal{O}\left(N^2\right)$			$\mathcal{O}\left(N\right)$	high	low
CG	$\mathcal{O}\left(N\sqrt{N}\right)$	$\mathcal{O}\left(\log\left(N\right)\right)$	$\mathcal{O}\left(\sqrt{N}\right)$	$\mathcal{O}\left(\sqrt{N}\log\left(N\right)\right)$	medium	medium
PCG with ICHOL(0) preconditioner [a]	$\mathcal{O}\left(N\sqrt{N}\right)$	$\mathcal{O}\left(\log\left(N\right)\right)^b$	$\mathcal{O}\left(\sqrt{N}\right)$	$\mathcal{O}\left(\sqrt{N}\log\left(N\right)\right)$ [b]	medium	high
MICHOL(0) PCG [c]	$\mathcal{O}\left(N\sqrt[4]{N}\right)$	$\mathcal{O}\left(N\right)$ [a]	$\mathcal{O}\left(\sqrt[4]{N}\right)$	$\mathcal{O}\left(N\sqrt[4]{N}\right)$ [a]	low	high
AMG PCG, V-cycle	$\mathcal{O}\left(N\right)$, full multigrid cycle	$\mathcal{O}\left(\log\left(N\right)\right)$	$\mathcal{O}\left(log\left(N\right)\right)$	$\mathcal{O}\left(log^2\left(N\right)\right)$	low	high
AINV PCG [d]	$\mathcal{O}\left(N\sqrt{N}\right)$	$\mathcal{O}\left(\log\left(N\right)\right)$	$\mathcal{O}\left(\sqrt{N}\right)$	$\mathcal{O}\left(\sqrt{N}\log\left(N\right)\right)$	medium	medium

[a] $\mathbf{A_M}$ reordered using MD algorithm
[b] Rough estimate, uncertain how the number of levels scale
[c] $\mathbf{A_M}$ reordered using the SRCM
[d] For different gridsize, nonzeros of approximate inverse scale linear with nonzeros of $\mathbf{A_M}$.

Chapter 7

Distributed numerical experiments of NSABRE0

In this chapter, the performance of Null-space Spline based ABeration REconstruction zero order constraints (NSABRE0) method will be analyzed, both in speed and wavefront reconstruction performance. First the numerical setup is introduced, next it will be disscussed how NSABRE0 is implemented on the GPU and at last the results of numerical experiments will be discussed.

7-1 Numerical setup

This section will be divided into two subsections. In the first subsection the simulation of turbulence will be discussed and in the second subsection the simulation of the Shack Hartmann will be discussed.

7-1-1 Simulation of the turbulence

The athmospheric turbulence is briefly discussed in section 2-1. In section 2-1 it was assumed that the refractive index fluctuations could by reprensented as Gaussian noise with zero mean. In this section a slightly more advanced model will be used, which is called the modified von Kármán model [30, 53]. The power sprectal density of the modified von Kármán model can be used to describe the statistics of the refractive index. The model which here will be used is the modified von Kármán power sprectal density function. The related phase power spectrum is given by [49]

$$\phi(\kappa) = \frac{0.023 r_0^{-5/3}}{\left(\kappa^2 + \kappa_0^2\right)^{11/6}} \mathrm{e}^{-\frac{\kappa^2}{\kappa_m^2}},$$

where r_0 denotes the Fried parameter, κ the spatial frequency, κ_0 is the outer scale frequency, defined by: $\kappa_0 = \frac{2\pi}{L_0}$ and κ_m is the inner scale frequency defined by $\kappa_m = \frac{5.9}{l_0}$. l_0 is the inner scale length and L_0 is the outer scale length. The Fourier coefficients can now be obtained by multiplying the square root of the phase power spectrum with independent Gaussian random numbers which have zero mean and unity variance. The phase screen is then given by the real part of the inverse fourier transform.
To improwe the lower order Fourier functions the subharmonic method described by Lane et al [35] is used. In [53] a MATLAB implementation can be found to simulate the turbulence. The

properties of the simulated wavefront can be found in table 7-1 wich is partially based on the properties used in [13]. A simulated wavefront, based upon everything discussed in this section can be seen in figure 7-1.

Table 7-1: Properties of simulated wavefront

Turbulence model	modified von Kármán
Number of phase screens	1
Fried parameter r_0	0.2 m
Turbulence outer scale length L_0	50 m
Turbulence inner scale length l_0	0.01 m
diameter of aperture L	2 m

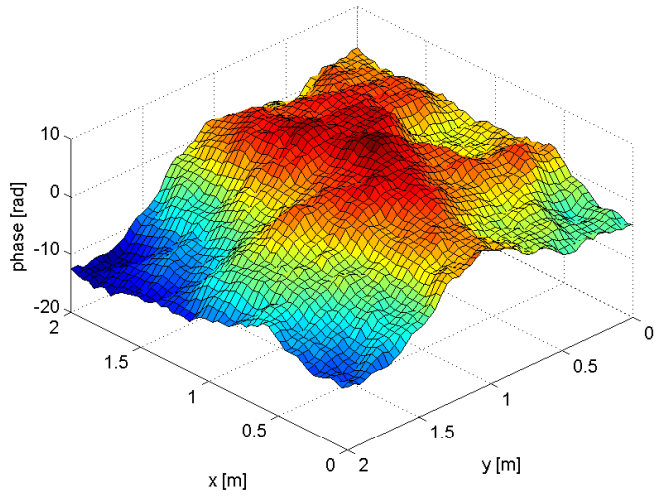


Figure 7-1: Simulated wavefront

7-1-2 Simulation of the Shack Hartmann sensor

In this section it will be discussed how the Shack Hartmann sensor can be simulated. The working principle of the Shack Hartmann sensor has already been addressed in section 2-4-2. Using Fourier optics it is possible to simulate the lenslet array. How to interpret the CCD sensor will be discussed in the next section. MATLAB code from the SABREdemo [1] is used for the simulation of the Shack Hartmann sensor. The properties of the simulated Shack Hartmann sensor can be found in table 7-2. they are partially based on the properties used in [13]. It assumed that the lenslets are perfectly aligned and that there is no electrical noise in the CCD sensor.

7-2 Distributed NSABRE0 implemented on the GPU

In this section it is discussed how NSABRE0 can be implemented on the GPU. In figure 4-6 one find the flow diagram of wavefront reconstruction. The steps that need to applied online will be explained in detail and are given by

1. Calculate the slopes from the CCD output, using a center of gravity method.

Table 7-2: Properties of the simulated Shack Hartmann sensor

Lenslet Array shape	square
noise	no noise
number of lenslets	variable
observation wavelength	$633 \cdot 10^{-9}$
focal distance lenslet	$5 \cdot 10^{-3}$
diameter lenslet	$150 \cdot 10^{-6}$
CCD grid size for each lenslet	16x16

2. Estimate the B-coefficients from slopes.
3. Evaluate the B-spline functions.

These steps will now be discussed individually.

7-2-1 Calculating the slopes

To obtain the slopes from the CCD of the Shack Hartmann sensor the Thresholding Center of Gravity (TCoG) method is used, which has already been discussed in section 5-4.

Triangulation and averaging the slopes

As a triangulation method, the Delaunay triangulation is used. The slope of the B-spline functions at each simplex is given by the average value of the measured slopes at the vertices of the simplex, see section 4-1-1. This can be obtained by a sparse matrix vector multiplication.

$$\mathbf{s}_{\text{avg}} = \mathbf{R}\mathbf{s}, \tag{7-1}$$

where $\mathbf{s}$ are the original slopes measurements, $\mathbf{s}_{\text{avg}}$ are the averaged measurements and $\mathbf{R}$ is a matrix which takes the average slope for every simplex. Equation 6-4 can now be redefined to

$$\underbrace{\mathbf{P}\mathbf{M}_{\text{H}}^{\text{T}}\mathbf{D}^{\text{T}}\mathbf{D}\mathbf{M}_{\text{H}}\mathbf{P}^{\text{T}}}_{\tilde{\mathbf{A}}_{\mathbf{H}}}\tilde{\mathbf{z}} = \mathbf{P}\mathbf{M}_{\text{H}}^{\text{T}}\mathbf{D}^{\text{T}}\mathbf{R}\mathbf{s}. \tag{7-2}$$

Furthermore a square grid is used and is assumed that the lenslets are perfectly aligned in the x and y direction and are of the same size.

7-2-2 Find the B-coefficients

In section 6-5 it is discussed how the B-coefficients could be calculated distributed from the slope measurements using the Preconditioned Conjugate Gradient (PCG) with the Algebraic Multigrid (AMG), Incomplete Cholesky factorization (ICHOL)(0) and the Approximate INVerse (AINV) preconditioner. Since there are different implementations possible for the discussed algorithms, it is shown in table 7-3 what implementation is used.

Table 7-3: Implementation details of the different algorithms to solve NSABRE0

	Reordering of $\mathbf{A_H}$	Algorithm or software used CPU	Algorithm or software used GPU
CG	SRCM	MATLAB 2013	cuSPARSE and cuBLAS
ICHOL(0) PCG section 6-5-3	MD	MATLAB 2013	cuSPARSE and cuBLAS [37]
AMG PCG section 6-5-5	SRCM	MATLAB 2013, code used [66] based on [50]. Number of levels used: $\log_2(N)$. Each level is solved using 3 jacobi iterations.	CUSP [3]
AINV PCG section 6-5-4	SRCM	MATLAB, AINV implementation based on AINV of SSOR precondtioner[28], which has the same sparsity as $\mathbf{A_H}$ with relaxation factor 1	cuSPARSE and cuBLAS

7-2-3 Evaluating the B-spline functions

As a last step the B-spline functions need to be evaluated. This is done by evaluating the B-spline functions at several locations, for example the actuator locations. The evaluation of the B-spline functions can be done off-line and can then be rewritten in the following sparse matrix vector multiplication

$$\phi = \mathbf{Bc}. \tag{7-3}$$

The $\mathbf{B}$ matrix contains the B-spline functions calculated at desired locations. ϕ is a vector with the phase of the wavefront at the desired locations. In this case the evalution grid wil have the same resolution as the number lenslet grid.

7-2-4 Pseudo-code Distributed NSABRE0 on the GPU

In this section an algorithm will be presented to do NSABRE0 in a distributed way on the GPU. The variables are declared in table 7-4. The sparse matrices will be stored in the hybrid

which was discussed in section 5-5. The algorithm will follow the same structure as declared at

Table 7-4: Variables used in the Distributed SABRE algorithm

name	original matrix	explanation
$\mathbf{E}$	$\mathbf{PM}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{R}$	equation (7-1,7-2)
$\tilde{\mathbf{A}}$	$\mathbf{PM}_{\mathrm{H}}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{DM}_{\mathrm{H}}\mathbf{P}^{\mathrm{T}}$	equation 7-2
$\mathbf{G}$	$\mathbf{BMP}^{\mathrm{T}}$	equation (6-3,7-3)
CCD		section 5-4
SHgrid		section 5-4
$centers_0$		section 5-4
ϕ		equation 7-3
$\tilde{\mathbf{z}}$		equation 7-2

the beginning of this section

1. Calculate the slopes from the CCD output, using a center of gravity method.
2. Estimate the B-coefficients from slopes.
3. Evaluate the B-spline functions.

Pseudo-code for Distributed NSABRE0

Here is the Pseudo-code for Distributed NSABRE0. Depending on the solver used, it will require a previous estimate of the solution $\tilde{z}_0$.

function D-NSABRE0($\tilde{A}$,G,CCD,$SHgrid$,$\tilde{z}_0$,E,$centers_0$)
  s= ParallelTCog($CCD, SHgrid$,$centers_0$)
  $b = Es$
  $\tilde{z}$ =solver(solves $\tilde{A}\tilde{z} = b$)
  $\phi = G\tilde{z}$
  return $\phi, \tilde{z}$
end function

In the next section results of numerical experiments are shown based on the implementation shown in this section.

7-3 Numerical experiments on the GPU

In this section, using the numerical experiments, the performance of distributed NSABRE0 will be shown. The first experiment is to show how NSABRE0 converges for different algorithms and gridsizes. After that the speed of the algorithm will be tested. Also the scalability of the solvers will be discussed. Before that it is given which hardware and software is used and there is a discussion about whether to using single floating points or double floating points on the GPU.

7-3-1 Single floating or double floating point on the GPU?

In this section it will be discussed in what precision all data will be stored and the calculations will be done. The advantage of using single floating points are clear; only half of the amount of memory is required and the amount of computations done per time unit are at least twice. Since wavefront reconstruction is very time critical and NSABRE0, single floating point precision will be used. An interesting discussion about this subject can be found in [2].

Table 7-5: Hardware setup

	Hardware
Graphics Card	NVIDIA geforce GTX 480
Memory	8 gigabyte DDR2
Processor	Intel core 2 quad Q9550

7-3-2 Hardware and Software setup

In this section the Hardware and Software is discussed.

Software

Since the available code for SABRE, is only the SABREdemo written in MATLAB, the basis will be MATLAB. The SABREdemo is extended to support the Distributed NSABRE0 algorithm as discussed in section 7-2-4. The algorithm itself is written in C/C++ and in CUDA C. To do the linear algebra operations, the NVIDIA cuSPARSE and NVIDIA cuBLAS libraries are used. Both libraries are free available, regularly updated and good documented. Other libraries were considered, such as the CUSP library used for the AMG preconditioner, but these cannot compete on those three criteria. The code, which consists of both CUDA C and C/C++ syntax is then compiled with the CUDA compiler to an object file. In MATLAB the object files are then linked to the CUDA and MATLAB libraries using MEX, which results in an MEX function that can be called in MATLAB. This both works in Windows and in Linux. See for an introduction to MEX [19]. For the numerical experiments Linux 64 bit is used.

Hardware

The hardware that is used is not the most up to date hardware as can bee seen in table 7-5. The NVIDIA geforce GTX 480 came at the market in march 2010, has 448 cores and has a cuda compute capability 2.0, which means unfortunately that it does not support dynamic parallelism, which was discussed in section 5-6-2. Due to the age of the GPU and the lack of support of dynamic parallelism the numerical experiments might not represent the full potential of applying SABRE on the GPU. Unfortunately this was the best GPU available.

7-3-3 Convergence and scalibility on the CPU

In this section it is disscussed how many iterations different iterative algorithms require to converge. The different algorithms can be found in table 7-3. Equation 7-2,

$$\tilde{\mathbf{A}}_{\mathbf{H}}\tilde{\mathbf{z}} = \tilde{\mathbf{b}}$$

is to be solved. An example of reconstructed wavefront calculated with the GPU can be found in figure 7-2. First the convergence of NSABRE0 solved with the Conjugate Gradient (CG) and different preconditioners is analyzed by computing the Strehl ratio and the residual for the first 50 iterations. This can be seen in figure 7-3. The strehl ratio is is approximated using equation 2-3. The first thing that is noticed in figure 7-3 is that the AMG PCG has the fastest convergence, the standard CG has the slowest convergence, AINV PCG and ICHOL(0) PCG perform in between. This corresponds to the theory covered in chapter 6. The second thing can be noticed in figure 7-3b, is that the convergence of AMG PCG stagnates. The final residual is limited by the performance of the solver of each level. To improve the results, more iterations per level are required or a different solver can be used. The last thing that can be

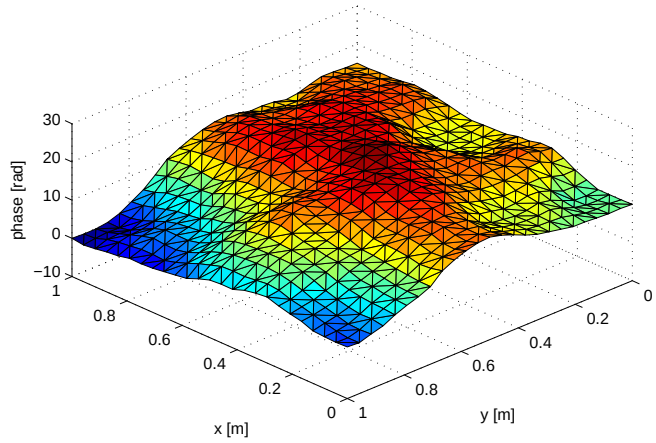
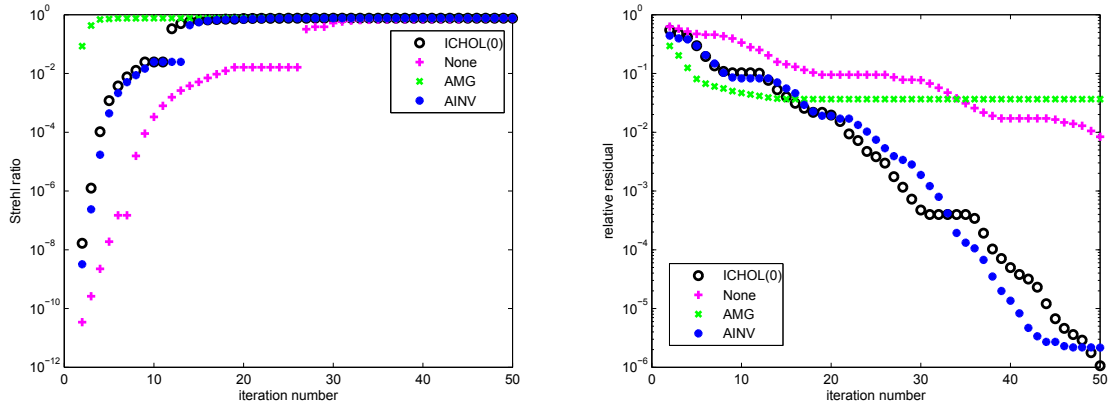


Figure 7-2: Reconstructed wavefront on the GPU using Cholesky to solve NSABRE0, compare with figure 7-1



(a) Strehl ratio

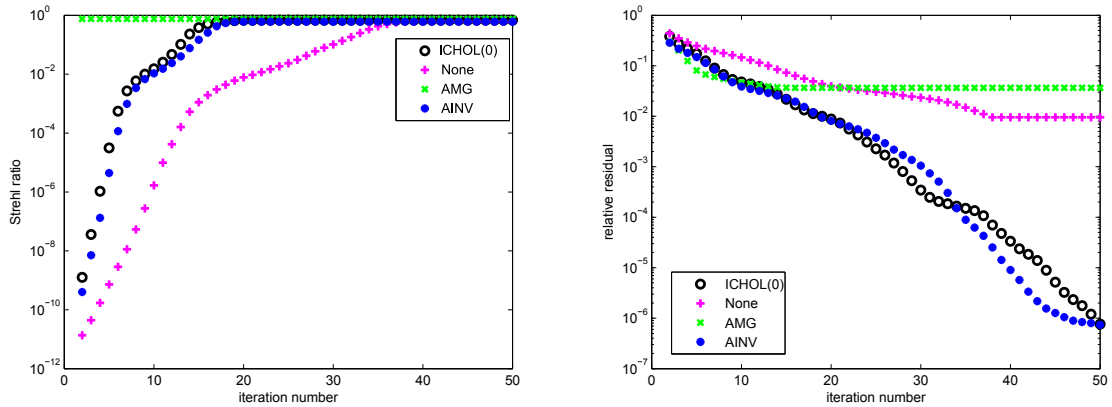
(b) Residual $||\tilde{\mathbf{A}}\tilde{\mathbf{z}} = \tilde{\mathbf{b}}|| \,/\, ||\tilde{\mathbf{b}}||$

Figure 7-3: Convergence for NSABRE0 solved with the CG method for different preconditioners

noticed is that the other algorithms stagnates shortly and even cause a jump in the Sthrehl ratio, which might indicate that $\mathbf{A_H}$ is ill-conditioned. NSABRE0 does lead to a well-organized matrix, the $\mathbf{A_H}$ matrix, but it is not as beautifully structured as the matrix coming from the Poisson equation where every row has the same non-zero values only shifted from each other. In table 7-6 it can be seen that for the case without anchor constraint the Poisson equation has a condition number twice as low as the $\mathbf{A}$ matrix from the NSABRE0 method, derived using the Null-Space. So clearly, some performance is handed in when using the generality of NSABRE0. For ill-conditioned systems, it is better to use the minimal residual method instead of the CG method. The minimal residual method is also a Krylov subspace method, but instead of that the residuals after the ith iteration are orthogonal with respect to the ith Krylov subspace the residual has minimum norm for $\tilde{\mathbf{z}}$ in the ith Krylov subspace. The algorithm is more complex than the CG method, which is a disadvantage. Now the minimal residual method is tested with the same preconditioners as earlier. Although none of the preconditioners stagnates anymore see figure 7-4b, the overall Strehl ratio is not improved for the minimal residual method as can be seen in figure 7-4a. Unfortunately, there was no time left to apply the minimal residual method on the GPU, so we will continue to use the CG method.

Table 7-6: Condition number for several cases of $\mathbf{A}$

	condition number for a 25x25 grid	condition number for a 50x50 grid
Poisson equation	273.3	1053
Householders QR Null-Space with anchor constraint	3328	16137
Householders QR Null-Space without anchor constraint, second smallest eigenvalue used	509.2	2129
Analytical Null-Space with anchor constraint	13321	62532
Analytical Null-Space without anchor constraint, second smallest eigenvalue used	527.0	2067



(a) Strehl ratio **(b)** Residual $||\tilde{\mathbf{A}}\tilde{\mathbf{z}} - \tilde{\mathbf{b}}|| \,/\, ||\tilde{\mathbf{b}}||$

Figure 7-4: Convergence for NSABRE0 solved using the minimal residual method for different preconditioners.

Scalability

Does the number of iterations scale as expected in section 6-5-2? As can be seen in figure 7-5 the answer is yes. Also the number of iterations required to converge can be seen in figure 6-5-2. It must be stated that the multigrid did not or only after many iterations converge for the two smallest grids.

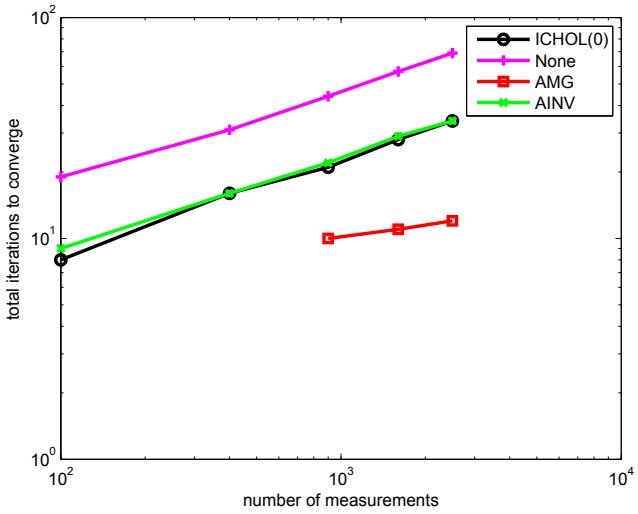


Figure 7-5: Number of iterations required to converge to a relative residual of $2 \cdot 10^{-2}$ for NSABRE0 solved with the PCG method with different preconditioners for different gridsizes.

7-3-4 Experiments on the GPU

In this section the results are shown of the experiments that were done on the GPU. Using the Visual profiler, the time was measured of the functions used in NSABRE0. The experiments were done for a gridsize of 10x10, 25x25 and 50x50. The results are compared to the Cholesky Solver from MATLAB. This was done on a computer with an INTEL i5 2400 processor and 8 GB memory. For the TCoG code, covered in section 5-4, on the CPU the only code available was the MATLAB code containing for loops. These are very inefficiently handled by MATLAB and there was no time to build a C version, therefore the TCoG is left out for the CPU case. As discussed in section 5-6-2 there is a chance that the calling of kernels takes more time than the computation itself, especially in the case of using a GPU that didn't support dynamic parallelism. As discussed in section 7-3-2 we use a GPU that does not support dynamic parallelism. In other words the GPU cannot prepare the next kernel itself, it has to commmunicate with the CPU. From the Visual profiler it becomes clear that in almost all cases the time for calling a kernel, takes more time than the computation of the kernel itself. Therefore the algorithm which uses the least amount of kernels is the fastest. In this case that is the Direct Cholesky Solver, which is indeed the fastest algorithm on the GPU as can be seen in figure 7-6. In figure 7-6 there is, time spend in idle. This is the time between kernels that the GPU is waiting on the CPU for the next kernel. It is obvious that there is a lot of optimization possible here by using a newer GPU. This also the reason that the time spent in idle is left out in figure 7-7.

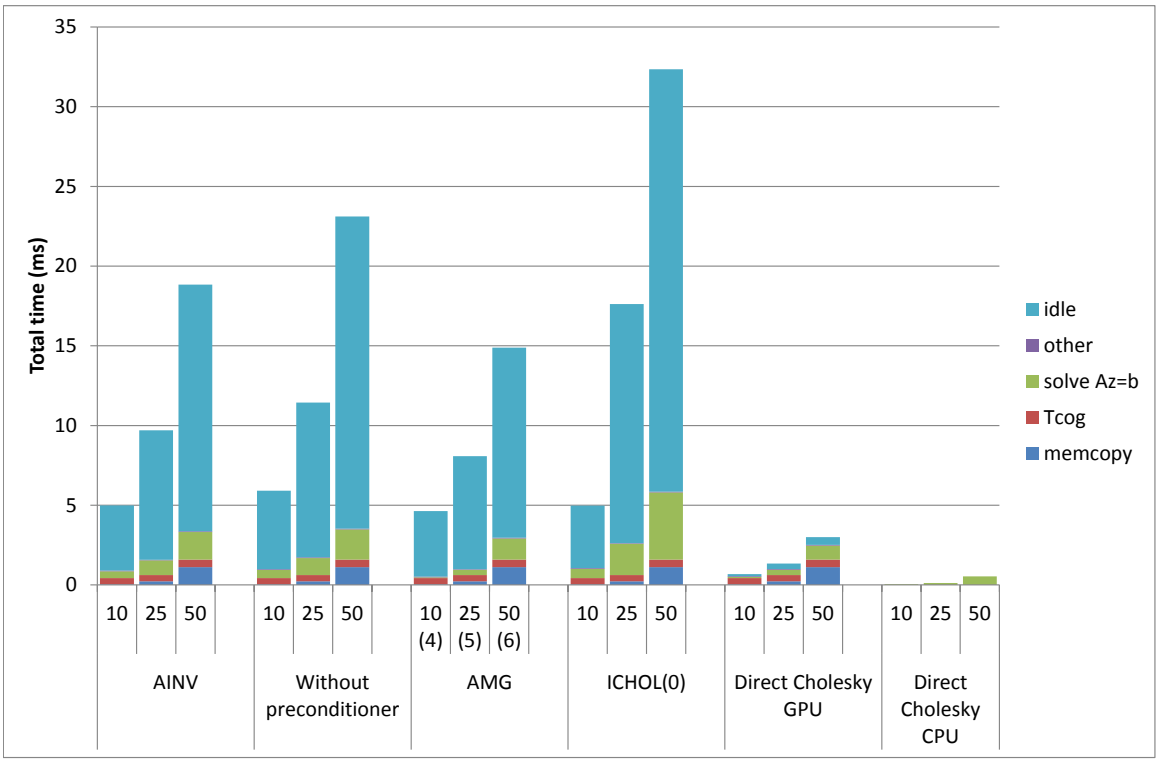


Figure 7-6: Required time on GPU for NSABRE0, for gridsize 10x10,25x25 and 50x50. For the MG preconditioner in brackets the number of levels used. Additionally also the time for the Direct Cholesky solver on the CPU is shown, the TCoG is in that case not taken into account.

Conjugate gradient and preconditioners

Do the different algorithms scale as expected and on what part is most of the time spend on? The required number of iterations is already discussed in section 7-3-3, in this section the focus will be on a single iteration, which is visualized in figure 7-7. For the CG and the AINV PCG, the additional time required for higher grids is almost negligible, which corresponds to the expectations shown in section 6-5-2. The most time in each iterations is spent on computation off the inner products. This makes both the CG and AINV PCG inefficient. The AINV PCG can be improved using a less sparse approximate inverse. The ICHOL(0) preconditioner is a bit dissapointing. The required number of iterations to converge for ICHOL(0) is similar to that of the AINV preconditioner but the cost of each preconditioning step is much higer. The increased computational time for higher grids for ICHOL(0) has to do with the fact that small levels are combined into one kernel call, which reduces overhead in communication. For the AMG preconditioner, based on these three observations and the observations from figure 7-6, it is clear that it does not scale as $\mathcal{O}\left(\log^2(N)\right)$ as discussed in section 6-5-2. Unfortunately due to the very generic code of the CUSP library, the AMG PCG algorithm is a black box which makes it virtually impossible to see what is happening inside the algorithm.

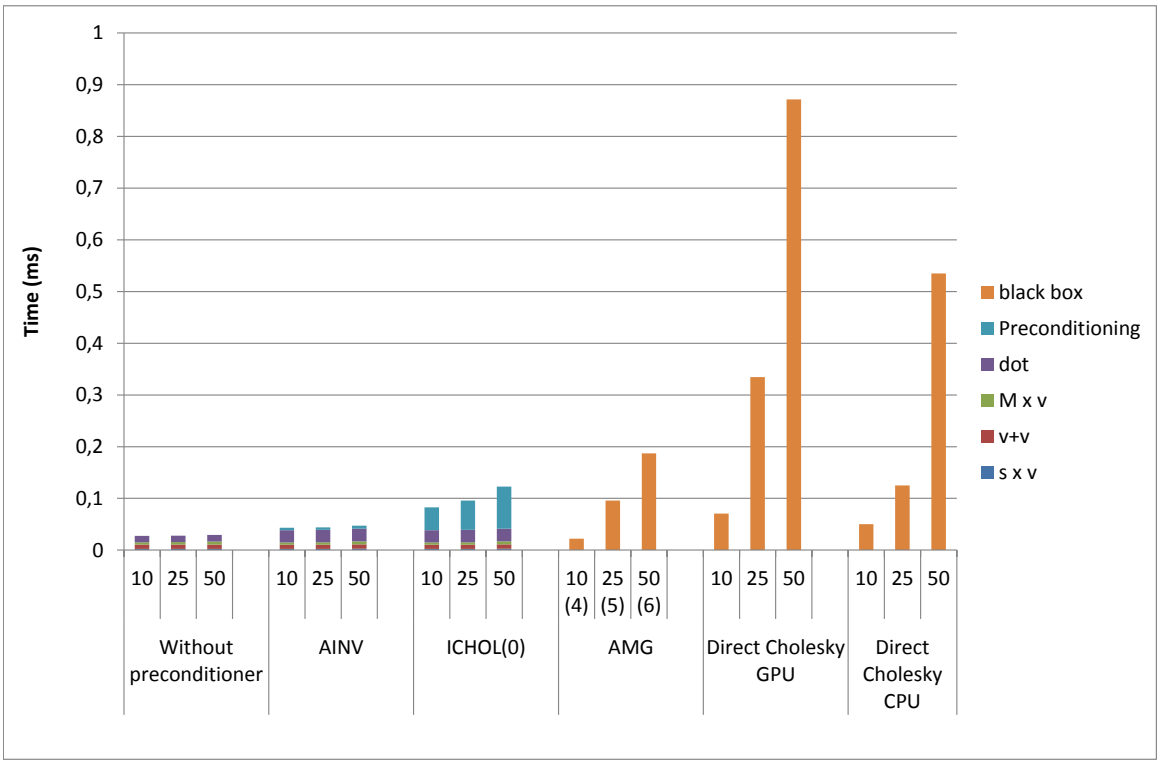


Figure 7-7: For NSABRE0 solved with the PCG with different preconditioners the time for a single iteration on the GPU is shown, for gridsize 10x10,25x25 and 50x50. For the MG preconditioner in brackets the number of levels used is shown. The time that the GPU is idle is left out. Additionally also the time required for the Cholesky solver both on CPU and GPU is shown.

Sparse Cholesky solver on CPU and GPU

At last the results of the Sparse Cholesky solver on both CPU and GPU are discussed. In theory the Sparse Cholesky scales as $\mathcal{O}\left(\sqrt{N}\right)$ as treated in section 6-4. The triangular solver on the GPU scales, slightly worse than $\mathcal{O}\left(\sqrt{N}\right)$ as can be seen in figure 7-7. This is again due to the case that small levels can be grouped into one kernel call, which reduces communication overhead for smaller grids. It can be seen that on the CPU the computational time scales as linearly with the number of measurements. It can also be noticed that in this experiment the CPU Cholesky solver is faster than the GPU Cholesky solver.

Center of Gravity method on the GPU

Also the TCoG method is implemented on the GPU. As can be seen in figure 7-6, the TCoG algorithm scales perfectly. There is some optimization possible by splitting the calculation of a spot location in multiple problems, threads. This reduces the required amount of registers. To use the TCoG method, the CCD data has to be sent to the GPU. Using GPUdirect[39] it is possible to by pass the CPU memory. There is no communication required between the each problem that calculates a slopes, this makes the GPU a good candidate to use for the

calculation of the spot locations.

7-4 Conclusion

Using numerical experiments it is shown how the PCG with different preconditioners and the sparse Cholesky solver behave in practice for solving NSABRE0. The first thing that was noticed was the stagnation of the CG, which revealed that NSABRE0 method results in an ill-conditioned system. At first sight the results on the GPU do not seem very promising because it is outperformed by the Cholesky solver on the CPU, but there is a lot of optimization possible, namely:

- Using newer hardware.
- If a fixed number of iterations is used, one inner product of the PCG can be left out.
- Depending on the situation less iterations might be required.
- Computational time of PCG is mainly determined by the inner products which uses the GPU very inefficiently. By using a less sparse AINV the efficiency of the CG method can be increased.

The AMG PCG has a lower computational complexity in theory, but this does not hold in practice and the AMG PCG uses the GPU very inefficiently. The AINV preconditioner can relatively be efficiently computed on the GPU and by using a less sparse AINV the overall efficiency of the CG method can be increased, which makes the AINV PCG the most promising algorithm.

Chapter 8

Conclusions and future work

In this thesis, the goal is to improve the resolution of a astronomical telescopes sited on earth by paralellysing the the wavefront reconstructor used in adaptive optics. First an introduction was given to the wavefront reconstruction and wavefront sensing. This revealed that the wavefront reconstruction problem is a numerical integration problem. Several wavefront reconstruction methods were addressed and it was shown that Spline based ABeration REconstruction (SABRE) is unique in the sense that it combines a sparse structure with the support of scattered data points and nonlinear basis functions. Using the null-space method it is possible to rewrite SABRE to solving a system of linear equations. By restricting the order of continuity to zero, the null-space of the constraints is garantueed to be sparse and the corresponding equation shows simillarities with the Poisson equation. It also possible to calculate the null-space of the zero order continuity constraints analyticallly. Despite the simillarities with the Poisson equation, Null-space Spline based ABeration REconstruction zero order constraints (NSABRE0) can be two times as ill-conditioned as the Poisson equation. The Conjugate Gradient (CG) with different preconditioners was investigated and applied on the GPU. The lowest computational complexity can be achieved with the Algebraic Multigrid (AMG) preconditioner $\mathcal{O}\left(\log^2(N)\right)$. Though since it has limited in amount of parallelism and also requires much communication the Approximate INVerse (AINV) preconditioner could be a better choice. The result on the GPU were of limited use, since it is known that a newer GPU could dramatically improve the performance. But it was still possible to see how the different algorithms scale and what part of the algorithm contribute the most to the computational time. Since in current AINV PCG case the computational time is mainly determined by the calculations of inner products, the AINV PCG computational time can be decreased by using a less sparse inverse. In the setting in this thesis the CPU is always faster. But since there is room for lots of improvement we believe that it possible that for larger grids the GPU will be faster than CPU. The future work can be divided into two parts, possible improvements of distributed NSABRE0 itself and general suggested areas of improvement of distributed SABRE. With respect to NSABRE0 the following areas of future work are formulated:

- What will the convergence of the CG be for scattered data?
- Main advantage of the AINV PCG is that the AINV preconditioner requires little amount of communication and has much amount of parallelism. It is already suggested that a less sparse inverse could improve the performance, since it makes the PCG method more depend on the efficient AINV preconditioning step. There are different approaches to find an approximate inverse, it should be investigated which approach gives the best results.

- Is there an explanation of the fact that NSABRE0 is ill-conditioned?
- how does distributed NSABRE0 perform against other distributed wavefront reconstructed methods?

For distributed SABRE in general, areas for future work are

- Is there a way to find a sparse basis of the null-space of first and higher order constraints? If there is a sparse basis, what will be the convergence of the null-space method?
- can distributed SABRE be extended to the temporal domain with for example a Kalman filter?

Bibliography

[1] Sabredemo. Package of MATLAB scripts related to wavefront reconstruction and multivariate B-splines.

[2] Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180(12):2526 – 2533, 2009.

[3] Nathan Bell, Steven Dalton, and Luke N. Olson. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM J. Scientific Computing*, 34(4), 2012.

[4] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, December 2008.

[5] Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *ACTA NUMERICA*, 14:1–137, 2005.

[6] Anna Burvall, Elizabeth Daly, Stéphane R. Chamot, and Christopher Dainty. Linearity of the pyramid wavefront sensor. *Opt. Express*, 14(25):11925–11934, Dec 2006.

[7] Livermore Computing Center. Introduction to parallel computing. https://computing.llnl.gov/tutorials/parallel_comp/#DesignGranularity. accesed: June 2013.

[8] Ali Cevahir, Akira Nukada, and Satoshi Matsuoka. Fast conjugate gradients with multiple gpus. In *Computational Science ICCS 2009*, volume 5544 of *Lecture Notes in Computer Science*, pages 893–903. Springer Berlin Heidelberg, 2009.

[9] Joana B. Costa. Modulation effect of the atmosphere in a pyramid wave-frontsensor. *Appl. Opt.*, 44(1):60–66, Jan 2005.

[10] Joana B. Costa, Roberto Ragazzoni, Adriano Ghedina, Marcel Carbillet, Christophe Verinaud, Markus Feldt, Simone Esposito, Elena Puga, and Jacopo Farinato. Is there need of any modulation in the pyramid wavefront sensor? *Proc. SPIE*, 4839:288–298, 2003.

[11] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[12] C.C. De Visser. *Global Nonlinear Model Identification with Multivariate Splines*. PhD thesis, TU delft, 2011.

[13] Cornelis C. de Visser and Michel Verhaegen. Wavefront reconstruction in adaptive optics systems using nonlinear multivariate splines. *J. Opt. Soc. Am. A*, 30(1):82–95, Jan 2013.

[14] J.W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[15] Todd Dupont, Richard P. Kendall, and Jr. Rachford, H. H. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM Journal on Numerical Analysis*, 5(3):pp. 559–573, 1968.

[16] Brent L. Ellerbroek. Efficient computation of minimum-variance wave-front reconstructors with sparse matrix techniques. *J. Opt. Soc. Am. A*, 19(9):1803–1816, Sep 2002.

[17] S. Esposito and A. Riccardi. Pyramid Wavefront Sensor behavior in partial correction Adaptive Optic systems. *Astronomy and Astrophysics*, 369:L9–L12, April 2001.

[18] David L. Fried. Least-square fitting a wave-front distortion estimate to an array of phase-difference measurements. *J. Opt. Soc. Am.*, 67(3):370–375, Mar 1977.

[19] Pascal Getreuer. Writing matlab c/mex code, 2010.

[20] John R. Gilbert and Robert Endre Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50(4):377–404, 1986.

[21] L. Gilles. Order-n sparse minimum-variance open-loop reconstructor for extreme adaptive optics. *Opt. Lett.*, 28(20):1927–1929, Oct 2003.

[22] Andreas Glindemann, Stefan Hippler, Thomas Berkefeld, and Wolfgang Hackenberg. Adaptive optics for large telescopes. *Experimental Astronomy*, 10(1):5–47, 2000.

[23] J.W. Goodman. *Introduction To Fourier Optics*. McGraw-Hill physical and quantum electronics series. Roberts & Co., 2005.

[24] Marcus J Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

[25] Olivier Guyon. Limits of adaptive optics for high-contrast imaging. *The Astrophysical Journal*, 629(1):592, 2005.

[26] P. J. Hampton, P. Agathoklis, and C. Bradley. A New Wave-Front Reconstruction Method for Adaptive Optics Systems Using Wavelets. *IEEE Journal of Selected Topics in Signal Processing*, 2:781–792, November 2008.

[27] J.W. Hardy. *Adaptive Optics for Astronomical Telescopes*. Oxford Series in Optical & Imaging Sciences. Oxford University Press, 1998.

[28] Rudi Helfenstein and Jonas Koko. Parallel preconditioned conjugate gradient algorithm on gpu. *Journal of Computational and Applied Mathematics*, 236(15):3584 – 3590, 2012.

[29] Richard H. Hudgin. Wave-front reconstruction for compensated imaging. *J. Opt. Soc. Am.*, 67(3):375–378, Mar 1977.

[30] A. Ishimaru. *Wave Propagation and Scattering in Random Media*. IEEE/OUP series on electromagnetic wave theory. Academic Press, 1978.

[31] G.E. Karniadakis and R.M. Kirby. *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press, 2003.

[32] A. N. Kellerer and A. M. Kellerer. Error propagation: a comparison of Shack-Hartmann and curvature sensors. *Journal of the Optical Society of America A*, 28:801, May 2011.

[33] D.B. Kirk and W.W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Applications of GPU Computing Series. Elsevier Science, 2010.

[34] R. Klees and R.P. Dwight. Applied numerical analysis (ae2212 part 1), 2012.

[35] R. G. Lane, A. Glindemann, and J. C. Dainty. Simulation of a Kolmogorov phase screen. *Waves in Random Media*, 2:209–224, July 1992.

[36] MILAMIN. Reordering. http://milamin.sourceforge.net/technical-notes/reordering. accessed: February 2014.

[37] M. Naumov. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the gpu. June 2011.

[38] Robert J. Noll. Zernike polynomials and atmospheric turbulence. *J. Opt. Soc. Am.*, 66(3):207–211, Mar 1976.

[39] NVIDIA. Gpudirect. https://developer.nvidia.com/gpudirect. accessed: February 2014.

[40] M.A. Olshanskii. Lecture notes on multigrid methods, 2012.

[41] F. Pérennès, M. Ghigo, and S. Cabrini. Characterisation of adaptive optic pyramid wavefront sensors fabricated by deep x-ray lithography. *Microelectron. Eng.*, 67-68(1):566–573, June 2003.

[42] J. Porter, H. Queener, J. Lin, K. Thorn, and A.A.S. Awwal. *Adaptive Optics for Vision Science: Principles, Practices, Design and Applications*. Wiley, 2006.

[43] Lisa A. Poyneer. Advanced techniques for fourier transform wavefront reconstruction. *Proc. SPIE*, 4839:1023–1034, 2003.

[44] Roberto Ragazzoni, Emiliano Diolaiti, and Elise Vernet. A pyramid wavefront sensor with no dynamic modulation. *Optics Communications*, 208(1–3):51 – 60, 2002.

[45] F. Roddier. The effects of atmospheric turbulence in optical astronomy. In E. Wolf, editor, *Progress in optics*, volume 19, chapter 5, pages 281–376. North-Holland Publishing Co., Amsterdam, 1981.

[46] F. Roddier. *Adaptive Optics in Astronomy*. University Press, 1999.

[47] F. Roddier, M. Northcott, and J. E. Graves. A simple low-order adaptive optics system for near-infrared applications. *pasp*, 103:131–149, January 1991.

[48] Francois J. Roddier. Wavefront curvature sensing and compensation methods in adaptive optics. *Proc. SPIE*, 1487:123–128, 1991.

[49] M.C. Roggemann and B.R. Hunt. *Imaging Through Turbulence*. Taylor & Francis Group, 2014.

[50] JW Ruge and Klaus Stüben. Algebraic multigrid. *Multigrid methods*, 3:73–130, 1987.

[51] Y. Saad. *Iterative Methods for Sparse Linear Systems: Second Edition*. Society for Industrial and Applied Mathematics, 2003.

[52] S.K. Saha. *Diffraction-Limited Imaging With Large and Moderate Telescopes*. World Scientific, 2007.

[53] J.D. Schmidt. *Numerical Simulation of Optical Wave Propagation With Examples in MATLAB*. Press Monograph. SPIE Press, 2010.

[54] W. H. Southwell. Wave-front estimation from wave-front slope measurements. *J. Opt. Soc. Am.*, 70(8):998–1006, Aug 1980.

[55] Eric Thiébaut and Michel Tallon. Fast minimum variance wavefront reconstruction for extremely large telescopes. *J. Opt. Soc. Am. A*, 27(5):1046–1059, May 2010.

[56] S. Thomas, T. Fusco, A. Tokovinin, M. Nicolle, V. Michau, and G. Rousset. Comparison of centroid computation algorithms in a shack-hartmann sensor. *Monthly Notices of the Royal Astronomical Society*, 371(1):323–336, 2006.

[57] T. N. Truong, A. H. Bouchez, R. G. Dekany, J. C. Shelton, M. Troy, J. R. Angione, R. S. Burruss, J. L. Cromer, S. R. Guiwits, and J. E. Roberts. Real-time wavefront control for the PALM-3000 high order adaptive optics system. 7015, July 2008.

[58] R.K. Tyson. *Principles of Adaptive Optics*. Series in Optics and Optoelectronics Series. CRC Press, 2010.

[59] V. V. Voitsekhovich and L. J. Sánchez. Effect of scintillations in curvature sensing. *A&A*, 399(3):1177–1181, 2003.

[60] Michel Verhaegen. Control for high resolution imaging, 2012.

[61] Mickeal Verschoor and Andrei C. Jalba. Analysis and performance estimation of the conjugate gradient method on multiple {GPUs}. *Parallel Computing*, 38(10):552 – 575, 2012.

[62] C. R. Vogel and Q. Yang. Multigrid algorithm for least-squares wavefront reconstruction. *Appl. Opt.*, 45(4):705–715, Feb 2006.

[63] Curtis R. Vogel. Sparse matrix methods for wavefront reconstruction, revisited. *Proc. SPIE*, 5490:1327–1335, 2004.

[64] Christophe Vérinaud. On the nature of the measurements provided by a pyramid wave-front sensor. *Optics Communications*, 233(1–3):27 – 38, 2004.

[65] Byron M. Welsh, Brent L. Ellerbroek, Michael C. Roggemann, and Timothy L. Pennington. Fundamental performance comparison of a hartmann and a shearing interferometer wavefront sensor. *Appl. Opt.*, 34(21):4186–4195, Jul 1995.

[66] Chin-Tien Wu. Algebraic multigrid linear solver. http://www.mathworks.com/matlabcentral/fileexchange/35866-algebraic-multigrid-linear-solver/content/AMG_NCTU_Taiwan/amg.m. accessed: February 2014.

[67] Harry Yserentant. Old and new convergence proofs for multigrid methods. *Acta Numerica*, 2:285–326, 1 1993.

[68] W. Zou and University of Central Florida. *Optimization of Zonal Wavefront Estimation and Curvature Measurements*. University of Central Florida, 2007.

Glossary

List of Acronyms

AINV	Approximate INVerse
AMG	Algebraic Multigrid
BCSR	Blocked Compressed Sparse Row
CCD	Charged Coupled Device
CG	Conjugate Gradient
CHOL	Cholesky factorization
CPU	Central Processing Unit
CSC	Compressed Sparse Column
CSR	Compressed Sparse Row
cuBLAS	CUDA Basic Linear Algebra Subroutines
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
E-ELT	European Extremely Large Telescope
ESO	European Southern Observatory
ICHOL	Incomplete Cholesky factorization
MD	Minimum Degree
MICHOL	Modified Incomplete Cholesky factorization
NSABRE0	Null-space Spline based ABeration REconstruction zero order constraints
OpenCL	Open Computing Language
PCG	Preconditioned Conjugate Gradient
SABRE	Spline based ABeration REconstruction
SIMD	Single Instruction, Multiple Data

SP	Scalar Processor
SRCM	Symmetric Reverse Cuthill-McKee
SSOR	Symmetric Successive Over-Relaxation
TCoG	Thresholding Center of Gravity

List of Symbols

$\hat{\phi}$	Vector with the estimate of the phase of the wavefront for different spatial locations
ϕ	Vector with phase of wavefront for different spatial locations
$\mathbf{c}$	Vector with B-coefficients
$\mathbf{D}$	Matrix with the derivative of the B-splines
$\mathbf{F}$	Matrix describing continuity constraints including the anchor constraint
$\mathbf{H}$	Matrix describing continuity constraints excluding the anchor constraint
$\mathbf{M}$	Analytical obtained null-space matrix of $\mathbf{F}$
$\mathbf{M_H}$	Analytical obtained null-space matrix of $\mathbf{H}$
$\mathbf{s}$	Vector with slopes
$\mathbf{z}$	Null-space vector
$\mathcal{O}$	Big O notation
$\phi()$	Phase of wavefront
$\psi()$	Wavefront
N, N	Number of sub-apertures