Performance Measure and Optimisation of a Computer Tomography Code for High Performance Computing

Stefan Wysocki

MSc in High Performance Computing
EPCC center
The University of Edinburgh
Year of Presentation: 2010

24th August 2010
Abstract

Time resolved three dimensional measurements of turbulent flames are required to further understand the combustion process and to support advanced simulation techniques. The understanding also helps to develop more sophisticated and efficient burners. A sensor technique applied by Floyd [17] uses computed tomography (CT) of chemiluminescence to reconstruct the structure of the turbulent flame brush, where a set of projection measurements are used to estimate the scalar field. CT allows a flame’s 3D chemiluminescence profile to be obtained by inverting a series of integral measurements. However, the method is often limited by the computational efforts required to reconstruct the field. Nowadays, modern computers allow the exploitation of parallel computing by using multi-core chips to reduce the time to solution. Therefore two CT algorithms (Algebraic Reconstruction Technique and Multiplicative Algebraic Reconstruction Technique) were parallelised for different memory systems. Open-MP was applied for the shared memory model, whereas a message-passing interface (MPI) was chosen for the distributed memory system. In addition to exploit modern multi-core chips, a mixed mode was also developed to maximise performance improvements. The nature and implementation of the algorithms led to a great deal of variance in the results. The Open-MP implementations performed better, but also suffered from result averaging, especially at higher number of CPUs. Good speed-ups were achieved with the mixed-mode version, even though the performance was still influenced by the communication overheads, especially at higher numbers of CPUs. Therefore the implementations are better suited to local clusters with unlimited CPU usage instead of super computers.
# Contents

1 Introduction 6
   1.1 Computed Tomography of Chemiluminescence 7
   1.2 Project Focus 10
   1.3 Outline 11

2 Algorithms 12
   2.1 Algebraic Reconstruction Technique (ART) 13
   2.2 Multiplicative Algebraic Reconstruction Technique 15

3 Hardware, Software and Compilers 16

4 Sequential code of ART 17
   4.1 Requirements 17
   4.2 Original Code 17
   4.3 Convergence Criterion 19
   4.4 Optimisation 19

5 Parallelisation 23
   5.1 Message Passing 24
   5.2 Shared Memory 27
   5.3 Mixed Mode 29
   5.4 Advanced Approaches 31
   5.5 Alternative Approaches 31

6 Test Cases 33

7 Results 35
   7.1 Correctness and Performance Results 35
   7.2 Message Passing 37
      7.2.1 ART 37
      7.2.2 MART 43
   7.3 Shared Memory 47
      7.3.1 ART 47
      7.3.2 MART 52
   7.4 Mixed Mode 56
      7.4.1 ART 57
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.2 MART</td>
<td>59</td>
</tr>
<tr>
<td>7.5 General Comments</td>
<td>61</td>
</tr>
<tr>
<td>8 Conclusion</td>
<td>63</td>
</tr>
<tr>
<td>9 Future work</td>
<td>65</td>
</tr>
<tr>
<td>Bibliography</td>
<td>67</td>
</tr>
<tr>
<td>A Additional Results</td>
<td>71</td>
</tr>
<tr>
<td>B Experiments</td>
<td>74</td>
</tr>
<tr>
<td>C Script</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Figures

1.1 The generic CTC set-up. $N_q$ cameras are located around a burner at radius $r$. The angle of each camera is $\theta$. [16] .................................. 10

2.1 Example projections at two different angles for a two cylinder objects [16] .... 12
2.2 Example projections based on line integrals. Note how multiple projections at a common angle comprise a view [16] ................................. 13
2.3 A pixel’s contribution $w_{q,pv}$ to projection $q_p$ is the intersection of the projection geometry and the pixel. The object domain has been discretised with $N_v$ pixels [16]. ........................................ 14

4.1 Convergence criterion against iteration number until solution is converged for a test case of 142x40x10 .................................................. 20

6.1 Matrix burner, its dimensions and a view from position indicated by the arrow. 33

7.1 Phantom (a) and ART (b) and MART (c) reconstruction ............................ 36
7.2 MPI outer loop ART algorithm - Speed up ............................................. 38
7.3 MPI outer loop ART algorithm - Efficiency ............................................ 39
7.4 MPI outer loop ART algorithm with varying relaxation factors- Run time (s) . . 40
7.5 MPI inner loop ART algorithm - Speed up ............................................. 42
7.6 MPI inner loop ART algorithm - Efficiency ............................................ 42
7.7 Example of communication pattern for ART algorithm with MPI ................. 43
7.8 MPI outer loop MART algorithm - Speed up ............................................. 44
7.9 MPI outer loop MART algorithm - Efficiency graphs .................................. 45
7.10 MPI inner loop MART algorithm - Speed up ........................................... 46
7.11 MPI inner loop MART algorithm - Efficiency ........................................... 47
7.12 Open-MP outer loop ART algorithm - Speed up ..................................... 48
7.13 Open-MP outer loop ART algorithm - Efficiency ..................................... 49
7.14 Open-MP inner loop ART algorithm no restrictions - Speed up .................. 50
7.15 Open-MP inner loop ART algorithm - Speed up ..................................... 51
7.16 Open-MP inner loop ART algorithm - Efficiency ..................................... 51
7.17 Example of CPU usage time line for the medium test case on 4 processors .... 52
7.18 Open-MP outer loop MART algorithm - Speed up ................................... 53
7.19 Open-MP outer loop MART algorithm - Efficiency ................................... 54
7.20 Open-MP inner loop MART algorithm - Speed up ................................... 55

1
List of Tables

3.1 Hardware specifications of the cluster 'Ness' ............................................. 16
4.1 Profiling results for the original ART algorithm on a 142x40 pixel, 10 view test case 19
4.2 Initial performance of ART algorithm for a convergence factor of $\Delta_c = 1e - 6$ .. 22
6.1 Test case parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
7.1 MPI outer loop ART algorithm - Run Time (s) ............................................. 37
7.2 Number of iterations required for a different number of processors ................. 38
7.3 MPI outer loop ART algorithm - non zero elements only ............................. 40
7.4 MPI inner loop ART algorithm - Run Time (s) ............................................. 41
7.5 MPI outer loop MART algorithm - Run Time (s) ......................................... 44
7.6 MPI inner loop MART algorithm - Run Time (s) ......................................... 46
7.7 Open-MP outer loop ART algorithm - Run Time (s) ..................................... 48
7.8 Open-MP inner loop ART algorithm update in place .................................... 49
7.9 Open-MP inner loop ART algorithm - Run Time (s) ..................................... 50
7.10 Open-MP outer loop MART algorithm - Run Time (s) .................................. 53
7.11 Open-MP inner loop MART algorithm - Run Time (s) .................................. 54
7.12 Mixed-mode processor decomposition ....................................................... 56
7.13 Mixed-mode ART algorithm - Run Time (s) .............................................. 57
7.14 Mixed-mode MART algorithm - Run Time (s) ............................................ 59
A.1 ART - Number of iterations required for number of processors using MPI ........ 73
A.2 MART - Number of iterations required for number of processors using MPI ...... 73
A.3 Number of iterations and run times for an alternative approach .................... 73
List of Algorithms

4.1 Pseudo code for the ART algorithm .................. 18
5.1 Pseudo Code - MPI implementation of ART algorithm for the outer loop .... 25
5.2 Pseudo Code - MPI implementation of ART algorithm for the inner loop .... 26
5.3 Pseudo Code - Open-MP implementation of ART algorithm for the outer loop
using a critical section ........................................... 28
5.4 Pseudo Code - Open-MP implementation of ART algorithm for the inner loop . 29
5.5 Pseudo Code - Mixed-mode implementation of ART algorithm .............. 30
Acknowledgment

The present work has been carried out at the EPCC center at the University of Edinburgh, where I undertook a part time Masters’ study in High Performance Computing during my Ph.D. research project. The Masters project was carried out under the supervision of Dr. Stephen Booth. I would like to acknowledge the financial support from the center throughout my part time studies.

I would also like to thank Dr. Booth for the opportunities he has given me and his continuous support and supervision throughout the project. He was very supportive at the beginning of the project when a lot of difficulties arouse and the code portability was limited.

I am also grateful for the support from my Ph.D. supervisor Dr. Andreas Kempf at Imperial College. He deserves gratitude for ensuring my financial support and the opportunity to undertake the MSc in High Performance Computing at the University of Edinburgh.

Furthermore I would like to thank my colleague Micheal Pettit for the discussions we had. The breaks and conversations helped me to gain a different perspective at problems. Last but not least, I would like to thank Eleanor Plant who put up with me in stressful times.
Chapter 1

Introduction

Scientists and engineers have been interested in the characteristics and efficiency of combustion for a long time. To name two examples, efficient combustion allows cars to travel further with the same amount of fuel and power stations to run more efficient. Improving combustion is also important to reduce CO₂ emissions and prevent in-reversible long term effects such as climate change. In addition rising fuel prices and diminishing fossil fuel supplies are another reason to improve the combustion process as much as possible. Alternative methods exist, but they are not fully developed yet or are not economically liable for the majority of energy users. Until these methods have been improved, fossil fuels will remain the main energy source and this requires measures to make the most of the remaining supplies.

Further investigations are required to fully understand and improve the process of combustions. Understanding the combustion process allows the development of more sophisticated burners, which require less fuel and burn more efficient. Scientists and engineers use experimental and computational studies to investigate turbulent flame structures. The main employed techniques to gain two dimensional experimental data are Particle Image Velocimetry (PIV) [18, 31] and Planar Laser Induced Fluorescence (PLIF) [14, 41]. However combustion is often turbulent and the process is characterised by unsteady three dimensional fluctuations occurring over a wide range of scales. Therefore the interest lies with three-dimensional data and the instantaneous 3D structure of a flame is also important for the fundamental understanding and modelling of turbulent combustion.

Although three-dimensional data is required, it is not easy to measure. Filling a flow volume with intrusive thermocouples for instantaneous temperature measurements is not desirable. A few three-dimensional experiments have been conducted using cross-plane LIF. For example, Bingham et al. [5] use a cross-plane laser sheet for tomographic measurements of a turbulent V-flame. Nonetheless the equipment for these measurements is very expensive, because lasers and very fast cameras are required. In addition measuring three-dimensional data can be difficult and therefore alternative techniques are required to improve combustion.

One of the techniques is the simulation of turbulence using one of three potential methods: Direct Numerical Simulation (DNS), Large-Eddy Simulation (LES) or Reynolds-Averaged
Navier-Stokes (RANS). Direct numerical simulation allows to simulate great detail, including all the fluctuations. DNS takes a set of equations in space and time to resolve the smallest turbulent eddies and simulate all fluctuations. However this approach is computationally very expensive and limited to small domains because it requires a grid that is fine enough to resolve the smallest eddies. These limitations can be overcome by time-averaging the governing equations and solving for mean quantities and fluctuations on coarser grids. This approach is called RANS, however it is not suited for complex flows due to the time-averaging. Large-Eddy simulation can be seen as an intermediate approach between DNS and RANS. The large scale eddies are directly represented by LES, whereas small eddies are modelled to increase accuracy. Flow quantities are spatially solved by applying a low-pass filter to the governing equation and LES represents the large scales of turbulent flow. LES might be the most suitable application for the modelling of turbulence. Although as noted by Pitch [40], LES is better in predicting the scalar mixing in comparison to RANS. LES is a very young approach and the development in all three approaches is ongoing.

LES and RANS approaches need to be verified with experimental data to aid their development. In the long term fewer experiments should be required and the industry can rely more on simulations. Until then it is important to verify the simulation results by comparing them with experimental data. These can be done by comparing instantaneous turbulence measurements with computational data. Plane data was often used for the validation of the simulation results, for example it has been done by in the Sandia National Laboratory [11]. However more complex flames and combustion processes require the time dependent nature of the flow to be validated correctly. As a result two-dimensional measurements are not ideal for the validation of numerical results of complex reacting flows that feature swirl. The three dimensional nature of the flames is important and 3D diagnostics are required to provide a greater insight. Therefore many different techniques have been suggested to obtain 3D data of flames. For example, Ng and Zhang [37] have also used a crossed Planar Laser Induced Fluorescence (PLIF) method to locate the instantaneous 3D flame normal to a single point, but as mentioned above, it has disadvantages. Abdel transforms are another method to obtain 3D data from single line-of-sight measurements of axis-symmetric flows. These approaches only provide information over a limited volume of the flame or require time averaging. For more detailed information two other methods are more suited: fast Laser Sheet Scanning (LSS) and Computed Tomography (CT). In the first method a laser is rapidly scanning through a volume of interest. These scans can than be stacked together in order to recover the three dimensional information, if the time frame between measurements is sufficiently short [6]. At the same time LSS requires very fast cameras with high resolutions, which are very expensive. The second method, computed tomography is the basis of this work, and therefore will be explained in more detail.

1.1 Computed Tomography of Chemiluminescence

Computed Tomography (CT) is essentially a mathematical inversion technique which is used to reconstruct images from one or two dimensional projections. Three dimensional data is returned from integral (line-of-sight) measurements taken at different angles around a chosen object. The method was first introduced by Radon in 1917, nonetheless it was not broadly applied until the 1970s when the EMI scanner was invented. The basic technique of computed tomography has
not changed since its early developments, but it remains an active field of research. Today CT is employed in many different disciplines such as medicine, seismology and electron microscopy, to name the most common areas of application.

Early attempts of applying computed tomography in the field of combustion were made in the early 1980s, but success has often been limited. The resolution of the reconstruction depends on the number of views, resolution of each view and signal to noise ratio. In order to achieve spacial resolution, measurements need to be taken simultaneously. Therefore many detectors and cameras are required. To get around the problem, some studies [3, 14, 41, 42] used complex set ups by using large number of mirrors, which have never been fully implemented. The main reason was the limitation of the equipment cost and availability.

A high number of angles (views) and low noise levels are required for optimum reconstructions, as well as simultaneous measurements for spacial resolution. With improving technology, one advantage of computed tomography is that the method is moderate in cost and it enables the use of standard cameras with flexible camera positions. In addition, high speed cameras would allow rendering of images for detailed film analysis of flames in the three dimensions from different angles. With computed tomography the reconstruction is always discrete and will always be an approximation of the continuous original field, where the error diminishes with increasing resolution and number of views. More information on computed tomography can be found in the book *Principles of Computerized Tomographic Imaging* by Kak and Slaney [29]. The book covers the mathematical principles and theory of tomography starting at one-dimensional signal processing and discussing the fourier representation. Later on the principles of the Algebraic Reconstruction Technique are covered.

Unlike in medical applications, if the right algorithm is selected, computed tomography allows for flexible camera positions. Therefore it has been a preferred method in combustion research in recent years. Examples of applying computed tomography to the field of combustion include the work by Feng et al. [15] and Gillet et al. [21], who used the method to localise integral measurements. Due to the flexibility in sensor location and number of sensors, studies focused on lower resolution, 2D reconstructions in industrial control applications such as pressure vessels [7, 12, 20]. Higher 3D resolutions are often considered in relation to flow visualisation applications [1, 15], though more complex set ups are required for these systems. 3D flame chemiluminescence has been investigated by Ishino and Ohiwa [27] and focused on turbulent flame flickering and expensive custom built cameras were used. Floyd [16] measured the chemiluminescence of a turbulent opposed jet flame to reconstruct an three-dimensional instantaneous scalar field of the flame.

Other natural flame emissions coupled with CT have included infra red (IR) soot emissions as applied by Correia et al. [8]. However chemiluminescence is preferred in the development of computed tomography in combustion because chemiluminescence gives a measure of the flame structure.

Chemiluminescence is the result of a chemical reaction that emits light which is not result-
ing from heat [17]. The reaction occurs in a narrow region of the flame and therefore gives information on its structure [46]. The reaction produces an electron in a more exited state, which makes it unstable. Therefore the electron will decay to a more stable ground state by the emission of a photon of energy $hv$, where $h$ is Plank’s constant and $v$ is the frequency of the emitted light. The photon energy is equivalent to the difference between the two states. The frequency $v$ of the emitted light depends on the specific transition that occurs for a particular molecule. Therefore, the concentration of a particular excited molecule can be determined from measurements of the emitted light intensity if the transition probability is known. For further information on the chemical reaction refer to the book *The Spectroscopy of Flames* by Gordon [19].

Measuring chemiluminescence is non-intrusive and therefore commonly employed in the study of turbulent combustion - especially in premixed combustion with no or little soot [2, 9, 39]. These studies looked at the local equivalence ratios and heat release. However results can be affected by absorption and scattering of emitted light by the flame before it reaches the detector. Measuring reliable concentrations of a particular species can become very difficult when a flame is optically thick. For this reason lean premixed flames are more suitable for chemiluminescence measurements. Optical measurements can also be conducted on non-premixed flames by partially premixing the fuel stream as demonstrated by Wang and Barlow [44]. Non-sooting flames can also be limited by self-absorption of the emitted light which reduces the measured signal. Chemiluminescence has also been used in the study of transient phenomena such as flash back [6] and to detect auto-ignition [33]. Recently, studies have been focusing on the modelling of chemiluminescence and different chemical reaction mechanisms [4, 32, 38] with a good agreement between the modelled values and calibrated concentrations.

Computed Tomography of Chemiluminescence (CTC) is the name given to a sensor technique that was developed by Floyd [16] and forms the basis of this work. The technique uses computed tomography to reconstruct 3D chemiluminescence profile of a flame from multiple measurements. Floyd also obtained the measurements simultaneously by using multiple cameras at different angles (fig. 1.1) and applied an algorithm to reconstruct the flame profiles. The aim of his work was to gain a better understanding of combustion and to contribute towards the development of turbulent combustion modelling by validating Large-Eddy Simulation results with his experimental data.
1.2 Project Focus

Floyd applied an Algebraic Reconstruction Technique (ART) algorithm, in an iterative manner, to reconstruct the three dimensional field. The algorithm was implemented as a sequential Fortran95 code, which had high computational cost of many hours. The aim of this project is to reduce the time to convergence by analysing, optimising and parallelising the sequential code that was written by Floyd [16]. This problem should be ideal for parallelisation, considered that the voxel\(^1\) reconstruction is independent of neighbouring voxels. Furthermore parallelising the code allows for a greater number of views\(^2\) (\(\sim 60\) view setup) and improves convergence speed of greater domains.

The code is to parallelise for a distributed memory system, a shared memory system and a mixed mode version using Open-MP and MPI in order to achieve maximum speed up. Initial tests were conducted on smaller test cases, which were generated based on a phantom study. Knowing the results in advance allows to evaluate optimised and parallelised reconstructions.

An alternative algorithm MART (Multiplicative Algebraic Reconstruction Technique) will also be tested. Floyd experienced problems with the implementation of this algorithm during his research studies as presented in greater detail in [16]. Therefore the algorithm will be re-implemented into the sequential code before parallelising. More details on both algorithms and their background are given in chapter 2.

---

\(^1\) A voxel is a 3D pixel

\(^2\) A view is the set of projections with a common viewing angle that together span the whole of the object domain at that angle.
1.3 Outline

Chapter 2 will outline the basics of the algorithms that have been parallelised as part of this project. The background theory of the algorithms and terminology are introduced. Finally the ART and MART algorithms are described based on the theoretical derivation. A brief overview of the hardware and software specifications is given in Chapter 3. The parallel code was developed and tested on a local cluster at the University of Edinburgh. In chapter 4 the performance and optimisation of the sequential code are discussed. The starting point is the sequential code as it was handed over by Floyd [16] at the beginning of the project. The optimisation focuses on source code and compiler optimisation. The convergence criteria is described at the end of this chapter. The different implementation approaches are outlined in Chapter 5. This chapter also contains a section about alternative approaches that were tested but did not lead to efficient results in terms of portability and convergence speed. The different test cases are briefly described in Chapter 6. The performances of the shared memory, distributed memory and mixed-mode implementations are discussed in chapter 7. The results are compared the the run time of the sequential algorithm in order to evaluate the parallel speed ups and efficiencies. A conclusion is given in chapter 8, before outlining potential future work in chapter 9.
Chapter 2

Algorithms

The aim of the CT algorithms is to solve the reconstruction problem by using a given set of projection measurements of a field in order to estimate the field. Formally, computed tomography can be defined as a general numerical technique that can reconstruct a discrete estimate of a scalar field from any set of integral measurements. The projection measurement \( I_p \) is defined as any spatial summation of a scalar field \( (f(x, y)) \), as shown in eq. 2.1. The integral measurements must be known and different viewing angles are required to reconstruct the problem. For simplicity, the analysis of the algorithms is reduced to two dimensions.

\[
I_{qp} = \int_{qp} f(x, y)dA
\]  

(2.1)

In the current context a projection refers to a single integral measurement of index \( p \). Fig. 2.1 shows an example of projections at two different angles for a two cylinder object. The projections are line of sight measurements of the objects.

Figure 2.1: Example projections at two different angles for a two cylinder objects [16]

A view refers to a set of \( N_p \) projections with a common viewing angle \( \theta \) (fig. 2.2). Views are indexed \( q \) and a total of \( N_q \) views are used to reconstruct an object. A view can be one or two dimensional.
In the literature a discrete estimate of the reconstructed object is often termed Image. This does not make sense in three dimensional terms as it can also be confused with two dimensional projections which themselves are images. Therefore it will be referred to as the reconstruction.

Numerous CT algorithms exist and they can be divided into two types: analytical and iterative. Most analytical types are based on the concept of back projection and require many low noise views for successful reconstruction [29]. These algorithms, in particular Filtered Back Projection (FBP), are applied most commonly to medical application where good access to objects is guaranteed. The iterative algorithms are generally preferred in physical science because they are more robust in the presence of noise. Furthermore an increase in modern computing power reduces the difference in processing time, especially with the opportunity of using multi-core systems.

Floyd et al. [17] used an iterative algorithm called ART (Algebraic Reconstruction Technique) in his work. It has been successfully applied in other studies [7, 13, 26]. The algorithm is robust and places no restriction on the geometry of the projections nor their relative orientations.

### 2.1 Algebraic Reconstruction Technique (ART)

The original ART algorithm was developed by Gordon [22] and has been modified into many different forms since. In this section an overview of the algorithm is given that was used by Floyd [16], more details about the algorithms are available in references [17, 25, 29]. As an iterative algorithm, it is generally based on the re-expression of the reconstruction problem as a system of linear equations. The system is obtained from the discretisation of the projection transformation by dividing the object domain into a grid of pixels of indeces $i$ and $j$. The projection transformation can be written in discrete form as
\[ I_{qp} = \sum_{v=1}^{N_v} w_{qpv} f_v \]  

(2.2)

where the pixel indices have been vectorized in order to simplify the notation. In equation 2.2 the contribution of the pixel \( v \) to the \( p^{th} \) projection is represented by \( w_{qp} \), as shown in figure 2.3.

Figure 2.3: A pixel’s contribution \( w_{qp} \) to projection \( qp \) is the intersection of the projection geometry and the pixel. The object domain has been discretised with \( N_v \) pixels [16].

Furthermore eq. 2.2 implies no self-absorption of the emitted light. Therefore the flame is regarded as optically thin, which is a reasonable assumption for the wavelength considered [17]. Therefore the reconstruction problem can be re-expressed as a system of linear equations, as shown below.

\[ I = Wf \]  

(2.3)

where \( W \) is referred to as the projection matrix, \( I \) is a column vector containing the integral values of all projection measurements and \( f \) is the vectorized object. Once the inverse of the projection matrix \( W \) is known, the vectorized object \( f \) can be found. The projection matrix has got a row for each projection and a column for each object pixel, which can make it very large. However, a projection often only covers a small portion and \( W \) can be very sparse.

In most situations it is not possible to find the inverse of the matrix due to high level of noise, limitations in the number of views available and a non-square projection matrix \( W \). In this case the system is under-determined because the number of unknown pixel values in the reconstruction domain exceeds the number of measured projection values. Therefore multiple solutions are possible. As a result, the ART algorithm can only provide a solution to eq. 2.3 based on some optimisation criteria.

In order to overcome this problem, the ART algorithms iteratively back project a diminishing error into the reconstruction domain until a solution criteria is met. In ART, the error is generated by comparing each measured projection value with an equivalent projection taken through a current iteration’s estimate of the object \( f_v^{(h)} \) [17]. The error is then normalised and back projected into the reconstruction domain using the contribution factor \( w_{qp} \) before the next projection is addressed. After all projections in all views have been addressed, the next iteration
will start.

Floyd [16] used a relaxed, additive version of the ART algorithm in his work, which is shown below in vector notation,

$$f^{(h+1)} = f^{(h)} + \beta w_{qp} I_{qp} - w_{qp} \ast f^{(h)} \over w_{qp} \ast w_{qp}$$

(2.4)

where $\beta$ is a relaxation factor, $f$ the reconstruction object at iteration $h$, $I$ is the column vector and $w$ is the weighting factor. The solution is converged if the difference of the object in two successive iterations is below some proportion of the field.

According to equation 2.4, each view contributes individually to the reconstruction. Regardless the mathematical description of the algorithm, the convergence speed can be improved by updating pixel values in place. Therefore the correction of a pixel depends on a previous estimate of a previous angle in the same iteration. For example, the value of a pixel $P_{q+1}$ at angle $\theta_q$ of becomes $P_q$ in the calculation of $P_{q+1}$ for the following angle $\theta_{q+1}$. Therefore the correction of the estimate is progressive within an iteration as well as between iterations.

### 2.2 Multiplicative Algebraic Reconstruction Technique

The Multiplicative Algebraic Reconstruction Technique (MART) is based on the basic ART algorithm, described above, and often considered as a successor of it. The disadvantage of the ART algorithm is that negative corrections are possible and reconstructions often show artefacts or tracers in the reconstruction [13]. To avoid this problem, the multiplicative ART algorithm was suggested [43], and used in the work as shown in eq. 2.5. The algorithm involves a multiplicative correction to the pixel based on the ratio of the recorded projection intensity and the reconstruction intensity from the previous iteration. The MART algorithm can diminish the influence of noise in the projections data and guarantees a meaningful solution when the initialisation is a constant.

$$f^{(h+1)} = f^{(h)} \ast (1 - \frac{w_{qp} I_{qp}}{w_{qp} \ast w_{qp}} (1 - I_{qp} / w_{qp} \ast f^{(h)}))$$

(2.5)

Similar to the ART algorithm, MART considers the correction of a pixel at a time, with an iteration being completed after all projections have been considered. In contrast to the ART algorithm, once a reconstruction pixel is set to zero with MART, it remains at zero because the correction is applied multiplicatively. As a result an initial estimate of $f^{(0)} = 0$ is not suitable. The mean value of derived from the projections or $f^{(0)} = 1.0$ are acceptable initialisations for the MART algorithm.
Chapter 3

Hardware, Software and Compilers

The following chapter will give a brief overview on the hardware, software and compilers that were used during the development and testing phases of the parallelisation. All of the development and testing was done on ‘Ness’ a small cluster at the University of Edinburgh. The hardware specifications are listed in the table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>front-end</th>
<th>back-end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>2.6 GHz AMD Opteron</td>
<td>2.6 GHz AMD Opteron</td>
</tr>
<tr>
<td>Number of processors</td>
<td>2</td>
<td>32 (divided into 2 boxes)</td>
</tr>
<tr>
<td>Memory per processor</td>
<td>2 GB</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

The front-end is used to log onto the system. The user can also edit and compile code at the front-end before submitting it to the back-end. The HPC jobs are run at the back-end, which is accessed via a SGE batch system. The operating system is Linux and the distribution is ‘Scientific Linux’. Two different compilers were used during the development work. The first compiler was the Sun Studio Compiler (Version 12) from the Oracle Solaris Studio, which offers good performance on multi-core applications as well as an in-build debugger and analyzer to evaluate the code during the development stages. Furthermore the compiler supports compiler auto-parallelisation. The second compiler is a Portland group compiler, PGF90. PGF90 is the interface to the Portland Group Fortran 90 compiler for IA-32 processors. The compiler was used to test the code’s functionality with different compilers in conjunction with the libraries. In addition the second compiler enables the evaluation of the optimisation of the Sun compiler. Different compilers might have a different effect on the optimisation of the code. The Portland group compiler was also used in connection with the analysing tool Vampir. Vampir (Visualisation and Analysis of MPI Resources) permits the analysis of the message events where data is transmitted between processors. Further information on the compilers and analyser tools can be found in the user guides.
Chapter 4

Sequential code of ART

The following chapter describes the sequential code in more detail. The code was developed by Floyd [16] as part of his post-graduate research. The main computational functions, based on the ART algorithm, will be discussed; including any prerequisites that are required for the code. Furthermore the original code performance will be evaluated, before suggesting optimisations to the sequential code. The convergence criteria, mentioned in Chapter 2, will also be explained in more detail.

4.1 Requirements

Two libraries were written by Floyd [16] to read the images and to write the reconstruction. The input is based on pgm-files and the output is written as a HDF5 data file. Floyd chose to use HDF5 files because they support data types and efficient and flexible I/O as well as high volume data. These characteristics make it useful for storing the three dimensional reconstruction. The pgm library was written for a phantom study and is kept separate from the reconstruction program to increase readability and reduce code complexity. These libraries will not be optimised or parallelised as part of this project because the effort does not justify the potential performance gain. Images are only read at the beginning (up to 64 files) and the output (a single file) is the estimated field once the solution is regarded as converged. Therefore the computational effort of the input and output is minimal compared to the reconstruction. Alternative file types might be implemented in the future to avoid the use of these libraries. If this is the case, the optimisation of the I/O should be re-evaluated at a later point.

4.2 Original Code

The code analysis is important before parallelising it to ensure that data dependencies are not broken. Breaking dependencies might result in the incorrect behaviour of the program, which might not be noticable from the visual output. In addition, a good sequential code might not be suited for parallelisation because the effort of parallelising does not justify the potential performance gain. Therefore the potential speed up should be investigated to see if the code benefits from parallelisation. A pseudo code will be used throughout the report to simplify the discussion of the parallelisation and to highlight the key areas that will be focused on as shown in alg.
4.1. The code is structured with subroutines and relevant subroutine names will be introduced throughout the report. The two important routines are the main program itself and the subroutine containing the reconstruction algorithm.

As mentioned before, the code was written in Fortran95 and uses subroutines to maximise readability. The projection matrix and projection weights are stored as sparse matrices to minimise the memory requirements by only storing non-zero values. Temporare data types are used to store the information of all views, for example angles about axis, distance to object, focal length, camera pixel size and view width. Furthermore the code uses floating point numbers with single precision, but can also handle double precision.

Algorithm 4.1 shows pseudo code outlining the main computational steps that are necessary for the reconstruction.

**Algorithm 4.1** Pseudo code for the ART algorithm

```plaintext
do k = 1, number of images
    read in data
end do
recon = 0.0  // initial guess
do while ≠ converged  // loop until converged
    do j = 1, number of images
        do i = 1, number of pixels per image
            error = P(i, j)−...  // calculate the correction
            recon_{q+1} = recon_{q} + error * relaxFac
        end do
    end do
    converged?  // check for convergence
end do
write recon to file
```

where $P(i, j)$ is the projection value and recon the reconstruction of the field. The j-loop will be referred to as the outer loop and the i-loop will be referred to as the inner loop to simplify the discussion of the parallelisation in Chapter 5 and 7.

The original program was profiled with a tool called ’Analyzer’ from Sun Studio to get a better understanding of timings required by each function and subroutine. The analyzer collects and arranges statistical data, as well as timing information. Table 4.1 shows the results of the initial analyzer output. A large percentage of the computational time (∼ 98%) is the actual calculation of the reconstruction. Therefore the algorithm will be optimised before developing a parallel algorithm. The routine is only called once, so the actual routine needs to be optimised instead of trying to reduce the number of calls. The second most costly routine is the convergence function. This is not surprising because the convergence needs to be checked after every completed iteration. Reducing the number of calls will affect the results and the effect needs to be evaluated carefully. The initial analysis was based on a high resolution (142x40 pixel) test case with a low number of views (10 views). The test cases are described in more detail in Chapter 6.
<table>
<thead>
<tr>
<th>Functions</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1014.12</td>
</tr>
<tr>
<td>ART algorithm</td>
<td>1000.74</td>
</tr>
<tr>
<td>Convergence</td>
<td>7.90</td>
</tr>
<tr>
<td>Others</td>
<td>5.36</td>
</tr>
</tbody>
</table>

Table 4.1: Profiling results for the original ART algorithm on a 142x40 pixel, 10 view test case

### 4.3 Convergence Criterion

A convergence criterion has been briefly mentioned in Chapter 2, which determines when the reconstruction is sufficient. Reconstruction algorithms are known to converge if sufficient projections and views are available to reconstruct the object. However, as shown by Floyd [16], the stopping point is often non-trivial due to noise in the measured data. The simplest approach is often suggested which converges when the difference between two successive iterations is less than some value $\Delta_c$. Floyd chose a variance of this approach. His reconstruction converges if three successive iterations are below some proportion of the field.

$$f^{(h+1)} - f^{(h)} < f^{(h+1)} \ast \Delta_c$$  \hspace{1cm} (4.1)

In this case $f$ is equivalent to `recon` in the pseudo code. This approach works well for the relaxed ART algorithm and is more stable in the presence of noise. Although, he suggests that there is no outstanding choice for the convergence criterion. An example of the convergence behaviour of the ART algorithm over the number of iterations is shown in fig. 4.2. The medium test case (142x40 pixels with 10 views) was used with a convergence factor of $\Delta_c = 1e^{-6}$ and 0.0..SP initialization. The difference between two successive iterations falls below the convergence criterion at 91 iterations before rising again until the solution is regarded as converged after 135 iterations.

### 4.4 Optimisation

Optimisation does not introduce new features to a program. The aim is to improve the run efficiency and to ensure minimum resources are used. The balance between complexity of the program to ensure good performance and readability is important. A very complex code is often difficult to develop further, however the end product should be as efficient as possible. Optimisation can be split into different levels:

- **Design level**: to ensure the best use of the available resources
- **Source code level**: to avoid poor quality code that executes slowly and contains a lot of unnecessary bottlenecks
- **Compile level**: to use different compiler flags to optimise the program’s performance
- **Assembly level**: to use an assembly language which is designed for a particular hardware platform
In this work the optimisation will focus on the source code and compile level. The assembly level will be neglected because the code should be as portable as possible. Therefore specialising the code for a specific platform would limit its portability. In addition optimising at assembly level is very time consuming and the potential gain might not justify the efforts required in this project.

Choosing the correct compiler flags is the fastest and easiest way of improving a programs’ performance. Different compiler flags were tested to gain a maximum performance improvement. The flags are often compiler dependent and during this study the Sun Studio compiler was used. In general all debugging flags (for example: -g) should only be used during the development phase. Optimisation flags (eg -fast -xtarget=native) should be used to achieve good performance. Modern compilers are usually good at optimizing source code without introducing potential bugs. Although care should be taken, as aggressive optimisation might break the code or change the result. Overall the biggest performance improvement can be achieved by avoiding debugging flags, which introduce additional overheads at the compilation stage.

The algorithm routine contains three nested loops, therefore the right loop order is important to exploit spatial locality in caches and to avoid cache misses. The loop order (in Fortran) should be arranged that the most outer loop corresponds to the array index furthest to the right, as shown in alg. 4.1. This was not the case in the original program. The cache can also be used for fast re-access of data and therefore cache misses should be minimized. Fetching data from cache is faster than getting it from the main memory. Although to guarantee the fast access, cache memory is a lot smaller than the main memory. Figure 4.3 shows the memory hierarchy and access times. Loop unrolling should be done by the compiler during the optimisation.
Figure 4.2: Example of memory hierarchy and access speeds

Optimisation flags and loop order were the main changes made to the code. An alternative method to improve the convergence speed was suggested by Guan and Gordon [24]. They investigated the convergence speed by changing the order in which projections are considered in the reconstruction. The multilevel access scheme, that is suggested, attempts to reorder projections so that consecutive projections are 90 degrees apart. With such a scheme they showed a drastrical increase in convergence speed of the ART algorithm. Furthermore they suggest that a random access scheme performs almost as well as the multilevel access scheme. Nevertheless it needs to be questioned if the parallelisation will benefit from the multilevel access scheme. Parallelising over the number of images (outer loop), images are not accessed in order. Instead images will be accessed at the same time before results are synchronised. Hence the multilevel access scheme does not contribute to the reduction in run time in parallel. In addition, Verhoeven [43] and Floyd [16] both experienced that the multilevel access scheme is only beneficial for low noise data that requires very few iterations.

Overall a good improvement in performance was recorded by choosing the right compiler flags, ensure cache coherent data and the correct loop order. Unlike stated by Floyd, the multilevel access scheme had a small impact on the overall performance in sequential. The key performance results after the optimisation are summarized in table 4.1. The initial analysis was carried out on the ART algorithm with a convergence factor of $\Delta_c = 1e^{-6}$. Alternative convergence factors were also tested, but $\Delta_c = 1e^{-6}$ will be used for all the performance measures to limit the number of variables.

The $-g$ flag introduces additional overheads for the debugger and analyser and the $-C$ enables array bound checking. Code for the 64 bit memory model is created with $-m64$ and
Table 4.2: Initial performance of ART algorithm for a convergence factor of $\Delta_c = 1e^{-6}$

$-xtarget = native$ optimises the code for the host system.

Applying all optimisations and optimum compiler flag options, the overall run time was reduced to 250 seconds for the original test case (142x40 pixel with 10 views). The biggest improvement was achieved by turning off the debugging flags and using optimisation flags. All parallelisation results will be compared to the fastest sequential implementation to gain a realistic idea of the potential speed up gained, as well as comparing it to the parallel implementation run on a single processor.
Chapter 5

Parallelisation

The ART algorithm family and the Fourier Back Projection (FBP) family both have advantages and disadvantages. The FBP requires many different views with low noise, which is not suitable for combustion. The disadvantage of the ART family is the long processing time for the reconstruction. Therefore the aim of this project is to speed up the ART algorithm run time by processing the data in parallel. Initially the problem appeared to be ideal for parallelisation because voxel values are independent of neighbouring voxel values. In the sections below, the parallelisation of the ART algorithm will be discussed. Three different versions have been created, one for a shared memory system using Open-MP, a distributed memory system using message passing (MPI) and a mixed-mode version for multi-core workstations. The same approaches are taken for the parallelisation of the MART algorithm.

As mentioned before, it is important to determined data dependencies in a sequential algorithm. However, the reconstruction problem is usually defined by a set of under-determined equations and the reconstruction is an approximation to the solution, as opposed to the actual solution. Therefore violating the data dependencies may change the particular solution, but does not necessarily reduce the quality of the reconstruction itself. Regardless, data dependencies need to be identified before it can be decided if they can be broken.

The data dependency between iterations for the reconstruction cannot be broken. It is not possible to estimate the solution of $f^{(h+1)}$ until a solution for $f^{(h)}$ has been found. Therefore the most outer loop cannot be parallelised.

The next loop to consider is the outer do-loop, which are the projections at different angles. There is a dependency. In the current implementation, the new estimate of the voxel value depends on a previous estimate of a previous angle in the same iteration. For example, the value of a voxel $V_{q+1}$ at angle $\theta_q$ becomes $V_q$ in the calculation of $V_{q+1}$ for the following angle $\theta_{q+1}$. It should be noted that Guan and Gordon [24] found that it is this dependency which slows down the convergence of the algorithm. So they suggested the multilevel access scheme, as described in section 4.4. As a result the loop can be parallelised under the condition of estimating a reconstruction of similar quality in comparison to the sequential reconstruction.
Finally, dependencies arise as each ray in a particular projection is considered. Usually each ray width only considers one voxel value of a particular row or column. Hence the calculation of each voxel, based on one ray in a particular projection, is independent of the calculation of other voxel.

As a result, the code is parallelised over the outer and inner loop separately for the shared and distributed memory versions, which results in 4 different versions. Parallelising over the outer loop means that each processor gets one or more images to estimate the reconstruction. The inner loop parallelisation is slightly different. One image is considered at a time and each processor considers a set of rays in each image. The mixed-mode version uses message passing for the outer loop, i.e. whole images per node, and shared memory for the inner loop, i.e. multiple processors per image.

Overall it is very important to maximise the work-to-communication ratio to ensure maximum efficiency. Therefore different test cases benefit from different data decompositions. In real reconstruction problems the work load for each view is not known in advance. Therefore static decompositions are used at the beginning of the project and alternative options will be investigated later on. More details are given in chapter 6 and 7.

5.1 Message Passing

The current implementation of the algorithms estimates the reconstruction in order of the projections, starting with the projection at angle $\theta_0$ and ending with the projection at angle $\theta_p$. The estimate for each projection is based on the estimate of the projection at the previous angle due to the immediate back projection of the error calculation, as discussed above. In addition, it has been mentioned that it is possible to consider the projections at random. However the reconstruction of the projection at angle $\theta_{x+1}$ is still based on the estimate of projection at angle $\theta_x$. This data dependency would limit the potential speed up of a parallelisation. Each estimate would need to be communicated and estimates would have to be evaluated sequentially.

These data dependencies will be ignored for a moment to look at the reconstruction from a different point of view. In the first iteration, based on the first projection, the reconstruction will be a 'smearred' image of the projection across all the rays. All voxels of a particular ray have the same values. If the estimate of the reconstruction of the second projection is based on the initial guess, instead of the previous projection, the reconstruction will similarly be the projection smeared back across all rays. If all projections are considered this way, the reconstruction is estimated by adding all estimates together. However this will overestimate the field. Therefore it is suggested to average the summed up reconstruction to achieve voxel values within the proper range and successive iterations could use the averaged estimate instead of the incrementally modified reconstruction. This alteration is easily parallelised by dividing projections amongst processors collecting the averaged results at the end of each iteration.

A pseudo code representation is shown in algorithm 5.1, using communication routines based on the message passing interface (MPI). A block distribution of projections was chosen. Alternat-
tive distributions, such as cyclic can easily be implemented. It is critical to check that there are a greater or equal number of projections to avoid idle processors. This becomes essential when using the code on a super computer such as Hector, where users are charged for the CPU time used. Furthermore each processor can only see image data in its own memory, which makes the redistribution of data to balance the work load during a reconstruction difficult. This redistribution introduces additional communication overheads and potential gains might be lost in the redistribution.

**Algorithm 5.1** Pseudo Code - MPI implementation of ART algorithm for the outer loop

```plaintext
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
if (size .GT. number of projections) STOP

do k = first image on processor, last image on processor
    read in data
end do

recon = 0.0 // initial guess

do while \( \neg converged \) // loop until converged
    do j = 1, number of images per processor
        do i = 1, number of pixels per image
            error = \( P(i, j) - ... \) // calculate the correction
            recon\(_q+1\) = recon\(_q\) + error * relaxFac
        end do
    end do

MPI_ALLREDUCE(recon, delta_image, MPI_SUM)
recon = delta_image / size
if (root .EQ. 0) converged? // check for convergence
    MPI_BCAST(converged)
end do

write recon to file

calculate correlation coefficient
```

It is also important to notice that averaging the results will lead to slower convergence because more iterations are required, which results in sub-optimal parallelisation. For example, dividing work onto 8 processors would result in twice as many iterations. Ignoring the communication overheads, there is still a potential speed up of 4 that can be achieved. This is still a significant improvement, especially considered that the code is most likely to be run on a small multi-core cluster (up to 16 CPU) where the usage is free (see section 7.2).

The inner loop considers the rays within each projection. For both algorithms, ART and MART, dependencies do not exist and so unrolling the loop across multiple processors is straightforward. The pseudo code is shown below; a block distribution has been used again with a message passing interface.
Algorithm 5.2 Pseudo Code - MPI implementation of ART algorithm for the inner loop

call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
do k = 1, number of processor
  read in data required by each processor
end do
starting_pixel = (i * rank) + 1
final_pixel = (i * (rank + 1))
recon = 0.0 // initial guess
do while ≠ converged // loop until converged
do j = 1, number of images
  do i = starting_pixel, final_pixel
    error = P(i, j) - ... // calculate the correction
    recon_{q+1} = recon_{q} + error * relaxFac
  end do
end do
MPI_ALLREDUCE(recon, delta_image, MPI_SUM)
recon = delta_image / size // check for convergence
if (root .EQ. 0) converged?
  MPI_BCAST(converged)
end do
write recon to file

calculate correlation coefficient

Melvin [36] suggests that the parallelisation of the inner loop is more complex because communication is required after each projection has been considered. This leads to high communication overheads and limits the potential speed up of the parallelisation. The frequent communication will impact on the performance in all three test cases. An alternative approach will also be investigated in which results will be communicated once all views have been considered. This approach is similar to the communication pattern of the outer loop parallelisation (see algorithm 5.2). Advantage is taken of the fact that the data dependencies between the projections can be neglected. The work/communication ratio is therefore maximised and better speed ups are expected.

Furthermore the amount of data communicated in both versions needs to be considered. Theoretically only updated voxels need to be communicated between processors, which reduces the message sizes and communication overhead. However a significant amount of processing is necessary to determine the voxels that need to be sent by each processor. The potential gain of sending smaller messages is lost by the additional calculations and messages required. Sending only updated voxels results in varying message sizes. So the message sizes need to be communicated before the actual message containing the data can be sent. Initial results showed a longer run time in comparison to the simple allreduce-statement. So for simplicity, the entire reconstruction is communicated between all processors using an allreduce-statement. This method keeps the code simple, it is easy to implement and it is currently faster than the alternative. Optimising the 'individual' message approach might result in better run times, but due to time limitations, this approach will not be developed.

The input routine also needs to be evaluated in the parallel version. The data required by each processor is read in individually. This solution is not very elegant, but it is easy to imple-
ment. No data types are required to communicate the sparse matrices containing the projection values, positions and weights. However, reading in the data individually might result in overloading the file system. The overload can happen if too many processors try to read or write data to disk at the same time. In this work, test cases were limited to 64 views on a maximum of 16 processors and no problems were encountered. The input and output are a small proportion of the run time, so the focus was set on parallelising the different algorithms. With continuing code development and increasing number of projections the risk of overloading the file system should be eliminated. In an alternative implementation, a master processor communicates the data to each processor to avoid overloading the file system.

In this work, a static work load decomposition was used for both MPI cases. It is often difficult to know the work load in advance and different decompositions result in different performances. Some cases work better with cyclic or random decomposition due to an uneven load balance. This often affects low or high ranked processors because the number of rays per projection is not constant. Therefore it is suggested to give extra work to these processors. For example, if the number of projections cannot be split evenly amongst the number of processors, the extra projections should be assigned to the lower and higher ranked processors.

The parallelisation of the algorithm for a distributed memory machine allows for greater scalability because memory requirements per processor are reduced. A greater number of views can be used in the reconstruction, resulting in faster and more accurate reconstructions. Furthermore higher resolution images can be used in the reconstruction process. Processors would only hold part of an image in their memory to estimate a very small part of the reconstruction. In addition, the code is also portable onto shared memory machines because MPI implementations can be executed on a shared memory machine.

The multiplicative algorithmic reconstruction technique was parallelised in the same way as the ART algorithm. Results for both algorithms are discussed in section 7.2.

5.2 Shared Memory

The MPI version is limited by the communication overhead and the extra iterations required due to the averaging of the estimate. Shared memory systems offer an attractive alternative which can eliminate the inefficiency of communication. Both algorithms (ART and MART) can also be parallelised for a shared memory system using the Open-MP directives. A similar approach to the parallelisation above has been carried out. Two different versions were created; the first version considers the parallelisation of the outer loop, the second version considers the inner loop. Although the main focus of this work was the parallelisation for a distributed memory machine and a mixed mode because the code will be used on a local 48 multi-core cluster at Imperial College London. The shared memory implementation was primarily used as a test basis for the mixed mode option.

In the shared memory implementation, all data is stored in global memory and threads work on parts of the global arrays. Although it is not necessary in this work, threads could access
Algorithm 5.3 Pseudo Code - Open-MP implementation of ART algorithm for the outer loop using a critical section

do k = 1, number of processors
    read in data required by each processor
end do

OMP PARALLEL, DEFAULT (NONE), PRIVATE(...), SHARED(...)
recon = 0.0 // initial guess

do while \( \not= \) converged // loop until converged
    OMP DO SCHEDULE(clause, chunk)
    do j = 1, number of images
        do i = 1, number of pixel per image
            error = \( P(i, j) \)−... // calculate the correction
            OMP CRITICAL
                recon\( q+1 \) = recon\( q \) + error * relaxFac
            OMP END CRITICAL
        end do
    end do
    OMP END DO
    if (root .EQ. 0) converged? // check for convergence
end do

write recon to file
calculate correlation coefficient

data of other threads through memory without specific communication.

For the outer loop, the estimates were calculated in private arrays before results were averaged into a global memory address space. The advantage of using Open-MP is that no communication is required but synchronisation takes place in memory. This implementation requires private arrays because each processor is working on its private estimate before results are globally averaged. These private arrays require additional memory.

To avoid the memory problem, updates could be done in place in the global array. Therefore data does not need to be averaged over the number of processors which reduces the number of iterations. This worked well for one of the three test cases (see section 7.3), but collisions occurred in the other test cases. The collisions lead to race conditions and the reconstruction would not converge. It is therefore not an appropriate solution to the problem. In order to avoid the memory problems and race condition, a critical statement could be placed around the reconstruction update to avoid these problems, as shown in Algorithm 5.3. The disadvantage of a critical section is that the code within it becomes sequential. This is a bottleneck when it comes to parallelisation. The pseudo code for the critical section is shown below. The correction calculation does not need to be in the critical section because the reconstruction is an estimate. So collisions, where one processor updates a voxel based on an old estimate, can be neglected.

Ideally each processor locks the voxel it is updating, so it can only be accessed by one processor at the time. Limiting the voxel access allows other processors to update other data points in the array without being limited by the sequential section. As mentioned before the main focus of this work was the parallelisation for a distributed memory system. The Open-MP version was implemented to test the different options and schedule clauses before implementing a
mixed-mode version. The results for the different schedule clauses are shown in Chapter 7 and Appendix A.

The inner loop controls the consideration within a single projection and so no data dependencies exist for that loop and the implementation should be straightforward. Melvin [36] suggests that the update can be done in place without any restrictions, as shown in algorithm 5.4. He assumes that only one ray per projection updates a voxel at a time. However it should be strongly emphasized that this ignores all the data dependencies and Open-MP guidelines. Therefore it is not regarded as a sophisticated solution and alternatives are investigated. A critical section or local delta image can be used, which is similar to the implementation of the outer loop. This limits the convergence speed but guarantees correct estimates. As work load varies with the problems, it is difficult to set an optimum schedule and chunk size. Different schedule clauses and chunk sizes were tested to find the best combination and to gain maximum speed up. These should be re-evaluated with different work loads and problem sizes.

\textbf{Algorithm 5.4} Pseudo Code - Open-MP implementation of ART algorithm for the inner loop

\begin{verbatim}
do k = 1, number of processor
   read in data required by each processor
end do

OMP PARALLEL, DEFAULT (NONE), PRIVATE(...), SHARED(...)
recon = 0.0  // initial guess

do while \$\neg\$ converged  // loop until converged
   do j = 1, number of images
      OMP DO SCHEDULE(clause, chunk)
      do i = 1, number of pixel per image
         error = P(i, j) - ...  // calculate the correction
         recon_{q+1} = recon_q + error \ast relaxFac
      end do
   OMP END DO
end do

if (root .EQ. 0) converged?  // check for convergence
   write recon to file
  calculate correlation coefficient
\end{verbatim}

The results of the performance analysis and efficiency are discussed in section 7.3.

\section{5.3 Mixed Mode}

Nowadays computers have multi-core chips consisting of 2 or more CPUs on a single chip and workstations with up to 16 quad-core chips are no rarity in research. Alternative languages are available, for example High Performance Fortran and POSIX. However, this work uses the message-passing interface (MPI) and shared memory directives (Open-MP) because they are widely known and can be considered as the industry’s standard.

Message passing often suffers from load imbalance and communication overheads, but it offers good scalability and portability. The shared memory scalability is often limited by the available
memory, although synchronisation overheads are often negligible. The aim of the mixed-mode version of an ART algorithm is to combine the advantages of both methods. The mixed-mode might be more efficient than MPI alone, without being limited by the memory (Open-MP).

MPI will be used for the communication between the nodes and Open-MP to synchronise threads through local memory on the chip. In the case of the ART and MART algorithm, the outer loop will be parallelised using MPI where each multi-core chip has access to the data of one or more images. Open-MP will be used for the inner loop to estimate the reconstruction per projection in parallel in order to gain maximum speed up. This should be beneficial to high resolution test images and reconstructions. Open-MP is usually better suited for fine grain problems. Decomposing the problem this way offers the best flexibility in terms of load balancing and it is straightforward to implement. Potential load imbalance per thread should also be reduced in the mixed-mode version, which will lead to a better performance. In addition it is important to ensure thread-safety and MPI calls within an Open-MP region should be placed inside a single or master directive.

The pseudo code, shown in algorithm 5.5, illustrates the concept of the mixed mode implementation. To be consistent with previous examples the mixed-mode version is shown with a static block decomposition and performance results of each algorithm are discussed in section 7.3.

Algorithm 5.5 Pseudo Code - Mixed-mode implementation of ART algorithm

```c
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
if (size .GT. number of projections) STOP
do k = first image on processor, last image on processor
    read in data
end do
recon = 0.0 // initial guess
do while not converged // loop until converged
do j = 1, number of images per processor
    OMP DO SCHEDULE(clause, chunk)
do i = 1, number of pixels per image
    error = P(i, j) - ...
    recon_{q+1} = recon_{q} + error * relaxFac
    OMP END DO
end do
MPI_ALLREDUCE(recon, delta_image, MPI_SUM)
recon = delta_image / size
if (root .EQ. 0) converged? // check for convergence
    MPI_BCAST(converged)
end do
write recon to file
calculate correlation coefficient
```
5.4 Advanced Approaches

The ART algorithm, chosen by Floyd [16], uses a relaxation factor to improve the stability of convergence and reduce the chance of over-prediction by one view due to noise, see section 2.1. The parallel version averages results which has a similar effect to applying a relaxation factor. Averaging and relaxing the estimate each iteration leads to over-relaxation. So the actual relaxation factor in the parallel ART can be removed or its influence can be minimized to improve convergence speed.

Furthermore, the MART algorithm does not change a voxel value in the reconstruction once it has been set to zero by one of the projections. Unlike the ART algorithm, the MART algorithm does not allow zero values to be changed once set by a view. Noise in the data could mean that a projection changes a zero to a non-zero value continuously. This can lead to race conditions and slow the convergence speeds. Therefore the ART algorithm will be altered to ignore voxels that have been set to zero to evaluate and compare results to the original algorithm. The change requires a non-zero initialisation like the MART algorithm.

5.5 Alternative Approaches

Alternative approaches to the averaging, as described above, have been evaluated before parallelising both algorithms. The alternatives were investigated because the averaging approach deviates from the original algorithm and performances can be improved with alternative methods.

In the first alternative approach, the maximum value of all reconstructions is used for each voxel as the new estimate instead of averaging voxels. The aim is to maximise the correction in each iteration. Having said that using the maximum value of each voxel lead to an over-prediction of the reconstruction and resulted in a higher number of iterations and slower convergence speed compared to the averaged version. In the ART algorithm the maximum value also lead to a more erratic behaviour of the corrections which also influenced the convergence behaviour. Negative corrections and noise in the data are the cause of the erratic behaviour when using the maximum value. In addition to the problems with the ART algorithm, some of the test cases would not converge with the MART algorithm and the maximum value approach. The convergence criteria of three successive iterations would not be met due to the erratic corrections. Therefore this approach was neglected.

The current implementation independently applies different views to the local reconstructions. The final reconstruction is gained by globally averaging the local reconstructions after each iteration. As mentioned above, the averaging of results deviates from the original algebraic reconstruction technique as described in equation 2.4. The method also requires additional iterations, which limits the performance of the speed up, and each processor has got a different reconstruction. Instead of merging the reconstructions, it is suggested to merge the changes of each processor. At the beginning of each iteration a copy of the current reconstruction is stored in a separate array. The corrections are then evaluated as normal, but the changes are recorded by each processor. Once all views have been considered within an iteration, the recorded changes
are applied to back-up copy of the reconstruction and the updated back-up becomes the new reconstruction. First of all the sum of all changes (for ART) and product of all changes (for MART) were applied to the back-up. This lead to a massive over-prediction of the results and additional iterations were required to correct for the over-prediction. Rather than summing up the changes, the approach was adjusted and the average of the changes were taken before added to the back-up of the reconstruction. Averaging the changes lead to a reduction in iterations (ART only) compared to the averaging of the reconstructions, but actual time to convergence was slower. The extra calculations necessary to store the back-up at the beginning of each iterations, storing the changes and reapplying it to the back-up are the main reasons for the slower time to convergence. On the down side, more memory is required to temporarily store the changes and the back-up. This approach also requires more views than processors, otherwise the approach is similar to the averaging of the reconstructions. Although in the case where each view has its own processor, it becomes the progressive algorithm of eq. 2.4. The key results from the investigations are summarised in Appendix A. The MART algorithm showed no reduction in iterations, but an increase in run time. Therefore results were neglected and work was focused on the averaging of the reconstructions.
Chapter 6

Test Cases

A matrix burner set up was used for the phantom study to create three test cases of different sizes and resolutions. The matrix burner plate, its dimensions and a sample view are shown in fig. 6.1. The arrow indicates a camera position from which a view was taken. The experimental work was carried out by Floyd [16] during his post-graduate research.

The three test cases (small, medium and large) were created to test the different implementations of the parallelisation and to measure the performance. The small test case should perform well in the MPI parallelisation of the outer loop because the memory requirements are relatively low. Most data should be stored in a higher level memory if run with a high number of processors. On the other hand the test is expected to perform badly for the MPI parallelisation of the inner loop because the processing time should be dominated by the synchronisation of the processors. The opposite is expected from the medium test case. A relatively low number of views limits the number of processors in the parallelisation of the outer loop. Therefore better results are expected of the inner loop parallelisation, especially with regards to the Open-MP
The large test case was created to test the behaviour under heavy loads for both loops. This test case is very interesting for the mixed-mode version, where good speed ups are expected due to the high resolution and high number of views. The key parameters for each test case are summarized in table 6.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Resolution (number of pixel)</th>
<th>Number of views</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowres_highview</td>
<td>64x18</td>
<td>64</td>
</tr>
<tr>
<td>highres_lowview</td>
<td>142x40</td>
<td>10</td>
</tr>
<tr>
<td>highres_highview</td>
<td>142x40</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 6.1: Test case parameters

*lowres* stands for *low resolution* and *highview* indicates a high number of views, and vice versa. It is very important to keep all processors equally busy while minimizing the communication/synchronisation overheads. The overall speed up will be determined by the slowest processor and as mentioned before, data is stored in sparse matrices. Therefore load balancing will be an important factor in the performance.
Chapter 7

Results

The quality of the reconstruction and performance measurements will be discussed in this chapter. The performance and speed up of the different parallel implementations (shared memory, distributed memory and mixed-mode) will also be evaluated. Analyser tools have been used to get a better understanding of the performance results and their output will also be discussed in the appropriate sections. The analysis of each method focuses on the aspect of strong scaling for each test case. In strong scaling, the problem size is kept constant and the number of processors changes. Weak scaling will be neglected because the main interest is the optimum performance at reasonable CPU cost.

7.1 Correctness and Performance Results

The correctness of the reconstruction was assessed in two ways: visually and numerically.

All test cases are based on high and low resolution phantoms of the matrix burner, which means the reconstruction is known. This allows for visual inspection and comparison of the results, which is a first indication to the correctness of the results. However the user should not rely on the visual comparison. Results can look correct but still be wrong or be of low quality.

The second method of testing the correctness of the reconstruction is by calculating the correlation coefficient of the phantoms and their respective reconstructions on single and multiple processors. The coefficient was calculated for all $N_q$ to confirm the validity of the visual process. Before parallelising the code the correlation coefficient was implemented according to

$$r_{xy} = \frac{\sigma_{xy}^2}{\sigma_{xx}^2 \sigma_{yy}^2}$$  \hspace{1cm} (7.1)

where

$$\sigma_{xy}^2 = \frac{\sum (x - \bar{x})(y - \bar{y})}{N}$$  \hspace{1cm} (7.2)

When the correlation coefficient approaches unity, the phantom and reconstruction are identical. Therefore it can be assumed that the reconstruction was successful. During the project, all results were in the range of $0.96 < r_{xy} < 1.00$. 
Figure 7.1 shows the phantom, ART reconstruction and MART reconstruction, respectively. The images were reconstructed on 10 processors with a convergence factor of \( \Delta_c = 1e^{-6} \). The reconstructions are almost identical to the phantom. Small differences appear on the peak of the flame on the right hand side in both reconstructions. The correlation coefficients were 0.999105 and 0.983214.

![Figure 7.1: Phantom (a) and ART (b) and MART (c) reconstruction](image)

In parallel computing, Amdahl’s law can be used to predict the maximum theoretical speed up using multiple processors. The speed up of a program using multiple processors is limited by the time needed for the sequential part of the code [10]. In this work, speed up refers to how much faster the parallel algorithm is compared to the sequential algorithm. It can be expressed as

\[
S_p = \frac{T_1}{T_p}
\]

(7.3)

where \( p \) is the number of processors, \( T_1 \) is the execution time on one processor and \( T_p \) is the execution time on \( p \) processors. Using the execution time of the parallel algorithm on one processor gives the relative speed up, instead of the absolute speed up where the execution time of the sequential algorithm is used. The aim of any parallelisation should be linear speed up, i.e., doubling the number of processors, doubles the speed up. However, communication and synchronisation reduces the efficiency and limits the speed up.

Estimating the utilisation of processors in solving the problem is another performance measure in high performance computing. The efficiency is defined as

\[
E = \frac{S_p}{p} = \frac{T_1}{pT_p}
\]

(7.4)

The values are between 0 and 1, where one indicates linear speed up. For high performance computers, where the users pay for the CPU time used, it is important to have a high efficiency in order to minimise computational costs.
7.2 Message Passing

The following section gives an overview of the key results from the performance analysis of the parallel ART and MART algorithms using the message passing interface for communication. Each algorithm has been evaluated for the three different test cases with parallelisation of the inner and outer loop. Time measurements were averaged over 3 runs to reduce the error in the measurements. The ART algorithm tests were initialised with \( \text{recon} = 0.0 \), whereas MART algorithm tests were initialised with \( \text{recon} = 1.0 \); unless it is otherwise stated.

7.2.1 ART

This subsection of the report discusses the results for the parallel ART algorithm using messages for communication. The discussion has been split into three sections; outer loop, inner loop and communication pattern.

Outer loop

The run time of the algorithm with parallelisation in the outer loop is shown in table 7.1. The run times show a great deal of variance. The poor performance is caused by communication latencies, work load imbalances and additional iterations required due to the averaging of the reconstruction.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64</td>
</tr>
<tr>
<td>1</td>
<td>23.7</td>
</tr>
<tr>
<td>2</td>
<td>153.3</td>
</tr>
<tr>
<td>6</td>
<td>148.2</td>
</tr>
<tr>
<td>8</td>
<td>146.6</td>
</tr>
<tr>
<td>10</td>
<td>142.2</td>
</tr>
</tbody>
</table>

Table 7.1: MPI outer loop ART algorithm - Run Time (s)

The performance gained from using multiple processors is lost in the communication latency and work load imbalance. The work load imbalance can be improved by using different domain decompositions. In this case a cyclic decomposition might perform better than the chosen static decomposition. The addition iterations required due to the averaging are another cause of poor performance. The extra iterations require additional computational time and communication, which reduces efficiency. Table 7.2 illustrates the number of iterations required for convergence based on the highres_highview test case. The complete list of iterations for the other test cases is shown in the Appendix. Ten processors require more than three times as many iterations as the sequential code, which means they have three times as much work to do. Despite the extra iterations, the parallelisation resulted in a speed up of factor 2 on 8 processors.
<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Run time (s)</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>171</td>
<td>192</td>
</tr>
<tr>
<td>2</td>
<td>137</td>
<td>221</td>
</tr>
<tr>
<td>4</td>
<td>152</td>
<td>382</td>
</tr>
<tr>
<td>6</td>
<td>155</td>
<td>551</td>
</tr>
<tr>
<td>8</td>
<td>89</td>
<td>347</td>
</tr>
<tr>
<td>10</td>
<td>133</td>
<td>677</td>
</tr>
<tr>
<td>16</td>
<td>119</td>
<td>565</td>
</tr>
</tbody>
</table>

Table 7.2: Number of iterations required for a different number of processors

The theoretical speed up is clearly not supported by the experimental results as shown in Fig. 7.2. The parallel algorithm scales well on 2 processors with a reasonable efficiency. Considering the nature of the algorithm and the necessary communication at the end of every iteration, an efficiency between $0.4 < \text{efficiency} < 1.0$ is acceptable, fig. 7.3 shows the efficiency results. A better speed up could be achieved by calculating the reconstruction on all elements and not just non-zero (sparse matrices). Processors would be load balanced and more computations would be required before communicating results. As a result the algorithm would scale better. Having said that it is not the purpose of parallel computing to slow down an algorithm to achieve better looking results. Furthermore storing zero elements has got a much higher impact on the memory and high resolution test cases with a high number of views might not be possible to store.

![Figure 7.2: MPI outer loop ART algorithm - Speed up](image-url)
The influence of the relaxation factor was also investigated for the parallelisation of the outer loop. The run times for the different test cases with different relaxation factors are shown in fig. 7.4. Yet again, the results show some variance and no correlation between the number of processors and the relaxation factor can be established. In general, all results are close to together for each test case. However the higher relaxation factors generally perform better, which proves that the averaging of results has some effect on the convergence. The effect of changing the relaxation factor is very case dependent; for example it had no effect on the small test case, but a big effect on the medium test case.
Figure 7.4: MPI outer loop ART algorithm with varying relaxation factors- Run time (s)

In the MART algorithm, values are not changed during the reconstruction once they have been set to zero. A similar implementation was tested on the ART algorithm to improve its convergence speed. The tests had to be initialised with $\text{recon} = 1.0$ and results are compared to the original ART algorithm with $\text{recon} = 1.0$ initialisation. The results are based on the highres_lowview test case and summarised in table 7.3.

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>ART initialised with 1.0 Run Time (s)</th>
<th>Speed up</th>
<th>ART only non-zero Run Time (s)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>1</td>
<td>220</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>133</td>
<td>1.50</td>
<td>138</td>
<td>1.58</td>
</tr>
<tr>
<td>4</td>
<td>114</td>
<td>1.74</td>
<td>123</td>
<td>1.78</td>
</tr>
<tr>
<td>6</td>
<td>116</td>
<td>1.72</td>
<td>128</td>
<td>1.71</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>1.55</td>
<td>135</td>
<td>1.62</td>
</tr>
<tr>
<td>10</td>
<td>148</td>
<td>1.34</td>
<td>136</td>
<td>1.61</td>
</tr>
</tbody>
</table>

Table 7.3: MPI outer loop ART algorithm - non zero elements only

The initial speed up measurements of the non-zero implementation look promising. Especially higher numbers of CPUs showed better performance improvements. However the algorithm converges slower on one processor compared to the standard ART algorithm, which was initialised with 1.0. The slower convergence helps improving the speed up ratios. Although the algorithm is approximately 8% faster on ten processors, where each processor holds the data from one view.

Considering the small impact of the relaxation factor and the change to non-zero values, all the other tests are performed with the standard implementation of the ART algorithm.
Inner loop

The performance of the inner loop with regular communication after each view was poor. No improvement in convergence speed was achieved in any of the three test cases. The computational time was dominated by communication, so any gain was lost. The results presented below consider the approach of communicating reconstructions once all views have been considered. The time to convergence is constant in all three test cases independent of the number of processors. The performance gain by using multiple processors for the reconstruction is lost in communication latency. As mentioned before, the ART algorithm also suffers from the additional iterations required due to the averaging of the reconstruction. Table 7.4 shows that the communication becomes even more dominant when the work load decreases on each processor with an increasing number of processors, in the medium test case as few as 6 processors. The best overall speed up was achieved in the smallest test case for ten processors. The algorithm only ran twice as fast, which means the efficiency was as low as 20%.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>34.7</td>
</tr>
<tr>
<td>2</td>
<td>24.1</td>
</tr>
<tr>
<td>4</td>
<td>26.5</td>
</tr>
<tr>
<td>6</td>
<td>29.7</td>
</tr>
<tr>
<td>8</td>
<td>17.8</td>
</tr>
<tr>
<td>10</td>
<td>16.7</td>
</tr>
<tr>
<td>16</td>
<td>20.6</td>
</tr>
</tbody>
</table>

Table 7.4: MPI inner loop ART algorithm - Run Time (s)

In some test cases the algorithm was slower on multiple processors compared to the sequential algorithm. The results of the poor speed up and low efficiency are not surprising, because the algorithm is not suited for this type of parallelisation using message passing. The work load on each processor is insufficient to justify the communication latency. Although the small test case seems to perform a little bit better because the data fits into a higher level cache if more than 8 processors are used. The small performance gain is quickly lost to communication overheads with an increasing number of processors again. Figures 7.5 and 7.6 show the speed up and efficiencies up to 16 processors for the three test cases.
Overall, the ART algorithm is not suited for this type of parallelisation due to the overheads and additional iterations required. Communicating after each view results in negative speed ups and alternative methods did not improve the time to convergence significantly. A shared memory
implementation is better suited for a small grain parallelisation because synchronisation takes place in the memory and inefficiencies should mainly arise from work load imbalances.

**Communication pattern**

The performances of the MPI implementations are dominated by the communication latencies. Hence the communication pattern needs to be analysed in greater detail to understand where the latencies are coming from. Figure 7.7 shows the communication pattern for the entire reconstruction (left hand side) and an enlarged section (right hand side). The computation is clearly dominated by the communication which limits the speed up. Furthermore the right hand side indicates that the processors are slightly load imbalance, which results in idle processors around the global communication. Processors are idle until all processors enter the MPI call for global communication. To minimise the load imbalance, alternative decompositions should be used. The work load is very case dependent and no single decomposition is best suited.

![Figure 7.7: Example of communication pattern for ART algorithm with MPI](image)

### 7.2.2 MART

The multiplicative algorithm is based on the ART algorithm that was invented by Gorden [23]. The correction is applied multiplicatively instead of additively, as discussed in section 2.3.

**Outer loop**

The MART algorithm scales better than the ART algorithm. The run time does not show great variance in the measurements. The time to convergence is still influenced by the communication overheads, as shown in table 7.5. The medium test case clearly suffers from load imbalance due to the static decomposition. 10 views are split over 6 processors and the slowest processors determines the overall speed. Therefore a cyclic decomposition should be used to improve work load balance amongst the processors. The small test case is dominated by the communication overheads again and hence the performance suffers from a bad work load / communication ratio. The small and medium test cases require less iterations in parallel, as shown in table A.2 in the Appendix. Further investigations are required to understand the behaviour of the large test case. Table 7.5 shows the run times for the three test cases.
<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64</td>
</tr>
<tr>
<td>1</td>
<td>12.5</td>
</tr>
<tr>
<td>2</td>
<td>6.7</td>
</tr>
<tr>
<td>4</td>
<td>5.5</td>
</tr>
<tr>
<td>6</td>
<td>2.6</td>
</tr>
<tr>
<td>8</td>
<td>1.9</td>
</tr>
<tr>
<td>10</td>
<td>5.2</td>
</tr>
<tr>
<td>16</td>
<td>4.7</td>
</tr>
</tbody>
</table>

Table 7.5: MPI outer loop MART algorithm - Run Time (s)

The speed up scales well up to 8 processors, especially taking the communication overhead and work load imbalance into account. The experimental data is close to the linear speed up for the small test case. The efficiency of the parallel MART algorithm (fig. 7.9) is satisfactory especially in comparison to the parallel ART algorithm. Nonetheless the efficiency drops rapidly with higher number of processors.

![Speed up MART Algorithm Outer Loop](image)

Figure 7.8: MPI outer loop MART algorithm - Speed up
The results of the MART algorithm for the MPI implementation of the outer loop are very promising. Even though the parallel algorithm requires the same amount of communication, the time to convergence is not influenced by additional number of iterations. The small test case shows a reduction of iterations up to 8 processors, which leads to the speed up results. The medium test case also shows a reduction in speed up, especially on 6 and 8 processors. The performance is still limited by the communication overhead and work load imbalance. The large test case requires more iterations in parallel as a result of the averaging. So the additional iterations as well as the communication overheads limit the performance for this test case. The advantage of fewer iterations is also noticed in the Open-MP and mixed-mode implementation, as discussed in the following sections.

**Inner loop**

The run times of the parallel MART algorithm for the inner loop are shown in table 7.6. The results look promising for test cases with low number of processors. The time to convergence decreases with an increasing number of processors. On more than 6 processors the communication latency dominates the computational time and performance gains are lost.
<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64</td>
</tr>
<tr>
<td>1</td>
<td>12.5</td>
</tr>
<tr>
<td>2</td>
<td>11.6</td>
</tr>
<tr>
<td>4</td>
<td>6.4</td>
</tr>
<tr>
<td>6</td>
<td>3.3</td>
</tr>
<tr>
<td>8</td>
<td>3.8</td>
</tr>
<tr>
<td>10</td>
<td>4.6</td>
</tr>
<tr>
<td>16</td>
<td>5.8</td>
</tr>
</tbody>
</table>

Table 7.6: MPI inner loop MART algorithm - Run Time (s)

The maximum reduction in run time for the small test case was achieved on 6 processors with a speed up of 3.8 and an efficiency of 63%. The medium test case performed best on two processors with an overall efficiency of 73%. Fig. 7.11 indicates that the efficiency continuously drops with an increasing number of processors. The large test case and the small test case peak again with a higher speed up on 6 processors. In general the efficiency drops with an increasing number of processors for these cases too.

Figure 7.10: MPI inner loop MART algorithm - Speed up
In general the MART algorithm is superior in the correction calculation. No relaxation factor is applied in the correction step and higher corrections are allowed within an iteration, due to the nature of the algorithm. Furthermore the algorithm does not change voxels once they have been set to zero, so the reconstruction is less affected by noise in a single view. The algorithm usually suffers from instability problems with noisy data. Overall the performance of the MART algorithm is better than the performance of the ART algorithm, especially the results of the small test case. The inner loop performance suffers strongly from the large amount of communication required within each iteration and does not match the results from the outer loop parallelisation.

7.3 Shared Memory

The following section will outline the key results from the performance analysis of the parallel ART and MART algorithm using a shared memory system. The performance of each algorithm has been evaluated with the same test cases that were used in the MPI versions. The algorithms were parallelised for a shared memory system as a test basis for the mixed-mode implementation. The key focus of the study was different schedule clauses and chunk sizes to gain maximum performance improvements. An analyser was also used to get a better understanding of the CPU occupancy.

7.3.1 ART

Outer loop

As described in chapter 5, the outer loop was parallelised in a similar way to the MPI implementation where delta images were used on each thread and results were accumulated at the end of
each iteration. The graphs only show the performance results for a static decomposition, for a direct comparison to the MPI performances. Further results of different data decompositions are shown in the appendix. Table 7.7 shows the time to convergence for a static decomposition. The results are similar to the MPI results. Time to convergence reduces lower number of threads. The best performance in all three cases was achieved on 6 threads, afterwards run time increases again. One of the reasons for the increase in convergence time is the additional iterations required due to averaging. Time for synchronisation also increases with an increasing number of processors due to an increasing possibility of load imbalance with a static decomposition.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>64_18_64</th>
<th>142_40_10</th>
<th>142_40_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22.0</td>
<td>251.6</td>
<td>171.2</td>
</tr>
<tr>
<td>2</td>
<td>17.9</td>
<td>143.4</td>
<td>115.3</td>
</tr>
<tr>
<td>4</td>
<td>12.0</td>
<td>109.1</td>
<td>102.3</td>
</tr>
<tr>
<td>6</td>
<td>11.7</td>
<td>90.1</td>
<td>106.0</td>
</tr>
<tr>
<td>8</td>
<td>13.8</td>
<td>142.2</td>
<td>124.3</td>
</tr>
<tr>
<td>10</td>
<td>16.3</td>
<td>154.7</td>
<td>160.1</td>
</tr>
<tr>
<td>16</td>
<td>16.8</td>
<td>N/A</td>
<td>163.1</td>
</tr>
</tbody>
</table>

Table 7.7: Open-MP outer loop ART algorithm - Run Time (s)

A dynamic decomposition performs better for the given test cases and results are shown in the appendix. The speed up results are promising for a parallelisation of up to 6 processors. As shown in fig. 7.3, the efficiency decreases with an increasing number of processors. The use of higher resolution views might improve the performance for higher number of processors because reconstruction would be less dominated by synchronisation.

![Speed up ART Algorithm Outer Loop](image)

Figure 7.12: Open-MP outer loop ART algorithm - Speed up
Overall the performance is slightly better than the MPI results because synchronisation happens in memory and no additional overheads due to sending and receiving messages are introduced.

**Inner loop**

The inner loop was parallelised with the correction done in place, as suggested by Melvin [36]. The results are very promising for different schedule clauses and different chunk sizes. Figure 7.14 shows the results for the different schedule clauses as indicated in the legends. Surprisingly the different schedule clauses have no major effect on the overall run times, which are summarized in table 7.8. The small difference in run times suggests that the medium test case is fairly load balanced.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run time (s)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static</td>
<td>Dynamic</td>
<td>Guided</td>
</tr>
<tr>
<td>1</td>
<td>249.1</td>
<td>249.1</td>
<td>244.3</td>
</tr>
<tr>
<td>2</td>
<td>157.4</td>
<td>146.5</td>
<td>146.8</td>
</tr>
<tr>
<td>4</td>
<td>94.9</td>
<td>91.2</td>
<td>91.5</td>
</tr>
<tr>
<td>8</td>
<td>71.1</td>
<td>73.5</td>
<td>71.4</td>
</tr>
<tr>
<td>10</td>
<td>65.4</td>
<td>67.8</td>
<td>68.5</td>
</tr>
</tbody>
</table>

Table 7.8: Open-MP inner loop ART algorithm update in place

The time to convergence reduces continuously with an increasing number of processors and the implementation seems to scale well. The speed ups look promising even at higher number of processors.
The correction in place only worked for the medium test case. The other test cases did not converge due to race conditions during the reconstruction. Processors tried to update the same voxel at the same time which lead to non convergence with the given convergence conditions. If the convergence conditions would not require three successive iterations, results would also converge for the other test cases. As the aim of this project is the parallelisation of the algorithm and not the implementation of new convergence criteria, it was decided to use the alternative implementation of applying a critical section around the correction step.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64</td>
</tr>
<tr>
<td>1</td>
<td>22.2</td>
</tr>
<tr>
<td>2</td>
<td>17.9</td>
</tr>
<tr>
<td>4</td>
<td>13.2</td>
</tr>
<tr>
<td>6</td>
<td>11.4</td>
</tr>
<tr>
<td>8</td>
<td>15.1</td>
</tr>
<tr>
<td>10</td>
<td>16.2</td>
</tr>
<tr>
<td>16</td>
<td>18.1</td>
</tr>
</tbody>
</table>

Table 7.9: Open-MP inner loop ART algorithm - Run Time (s)

The critical section limits the potential speed up of the code, especially with a higher number of processors. The limitation of the critical section is reflected in the results shown below. The time to convergence is limited by the critical section and therefore increasing the number of processors does not improve the speed up.
Overall the critical results worked for all test cases; however the performance is limited by the critical section in the main body of the algorithm. Therefore the implementation needs to be re-evaluated. To improve the performance a similar approach to MPI can be used where results...
are stored in a delta image on each processor and results are averaged at the end of an iteration. However it might not be a suitable implementation for test cases with a large number of views at high resolution because a lot of memory would be required.

**CPU occupancy**

It is important that all CPU’s are equally busy and threads are not idle to gain maximum performance. In this case the time to convergence is determined by the slowest processor. Therefore the CPU usage was analysed to get a better understanding of idle threads. Figure 7.17 shows a graphical representation of the CPU usage for the first few iterations based on the medium test case run on 4 threads. The clock profiling interval of the analyzer was 11000 microseconds with periodic sampling every second.

![Figure 7.17: Example of CPU usage time line for the medium test case on 4 processors](image)

Thread 1.1 uses CPU time at the beginning when all the data is read in, while the other threads are idle. The figure also shows that the 2nd thread is the most loaded thread, while thread 4 uses the least CPU time. The gaps mean idle time and green indicates CPU usage. The CPU usage of different decompositions can be tested and analysed to get a better understanding of the work load balance. The end of the program should look similar with all threads being idle apart from thread 1 that writes the results to file and the correlation coefficient is calculated.

### 7.3.2 MART

**Outer loop**

The time to convergence results for the Open-MP parallelisation of the outer loop is summarized in table 7.10. Good performances were achieved for a small number of processors. Especially the results of the small test case look very promising with near linear speed up for less than 6 processors. On more than 6 processors the performance seems to be limited by the required synchronisation. Only the large test case shows a reduction in time to convergence on more than 6 processors and the performance drops dramatically on 16 processors. At that point the work load / synchronisation ratio is insufficient. Larger problem sizes should be investigated to achieve better scaling.
<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64</td>
</tr>
<tr>
<td>1</td>
<td>11.1</td>
</tr>
<tr>
<td>2</td>
<td>6.1</td>
</tr>
<tr>
<td>4</td>
<td>2.9</td>
</tr>
<tr>
<td>6</td>
<td>2.3</td>
</tr>
<tr>
<td>8</td>
<td>4.2</td>
</tr>
<tr>
<td>10</td>
<td>4.7</td>
</tr>
<tr>
<td>16</td>
<td>6.8</td>
</tr>
</tbody>
</table>

Table 7.10: Open-MP outer loop MART algorithm - Run Time (s)

The speed up and efficiency results are acceptable for the small and medium test case (fig. 7.18 and fig. 7.19). The large test case does not perform as well as expected, even though it scales to a larger number of processors. The efficiency for all three test cases dropped drastically with an increasing number of processors.

Figure 7.18: Open-MP outer loop MART algorithm - Speed up
Overall the performance suffers from the synchronisation overheads after every iteration and the additional iterations required due to averaging of estimates in the large test case.

**Inner loop**

In the inner loop, the correction of the MART algorithm was done in a similar way to the MPI version where each thread has a private copy of the reconstruction and results are averaged at the end of an iteration. The update in place lead to race conditions which is not an acceptable standard for an algorithm to be used. Furthermore applying a critical section around the correction calculation limits the parallel performance. The time to convergence results are summarized in table 7.11. The overall reduction in run time is a lot better than the alternative method showed with the ART algorithm.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18</td>
</tr>
<tr>
<td>1</td>
<td>12.4</td>
</tr>
<tr>
<td>2</td>
<td>7.8</td>
</tr>
<tr>
<td>4</td>
<td>3.1</td>
</tr>
<tr>
<td>6</td>
<td>2.1</td>
</tr>
<tr>
<td>8</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>2.9</td>
</tr>
</tbody>
</table>

Table 7.11: Open-MP inner loop MART algorithm - Run Time (s)

All three test cases scale to a higher number of processors; and exceptional results were achieved with the smallest test case. The speed up is almost linear for up to 8 processors. The
problem becomes too small on more than 8 processors and the performance does not improve any further. The medium and large test cases also show reasonable results with speed ups close to 4 times faster on 10 processors.

Figure 7.20: Open-MP inner loop MART algorithm - Speed up

Efficiencies as high as 95% were achieved on 4 and 8 processors for the small test case. The medium and large test case show efficiencies of $\sim 60\%$ on 4 processors.
Again, the MART algorithm scales better than the ART algorithm because less additional iterations or even fewer iterations are required due to the averaging of the results. Performances can be improved further by choosing different schedule clause and chunk sizes to find the best data decomposition.

### 7.4 Mixed Mode

The analysis of the mixed mode implementation varies a little bit from the previous two analysis. The outer and inner loop are both parallelised at the same time, as described in chapter 5. Therefore different MPI node and Open-MP decompositions were tested. Table 7.12 shows an overview of the decompositions that were tested.

<table>
<thead>
<tr>
<th>Number of processors (MPI)</th>
<th>Number of threads (Open-MP)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 7.12: Mixed-mode processor decomposition

The tests were limited to 16 nodes due to hardware limitations. Further decompositions should be tested on a computer with more CPUs, up to 64 CPUs in total for the two test cases.
with a large number of views.

### 7.4.1 ART

Run times are shown in table 7.13. The run time drops continuously with an increasing number of processors, however the efficiency drops below 40% for all test cases run on eight or more processors, independent of the decomposition. The inefficient jobs can be justified in situations where very fast results are needed. Although running these jobs on a big supercomputer like Hector would cost expensive resources for a minimal convergence advantage.

<table>
<thead>
<tr>
<th>Num. of Proc.</th>
<th>Num. of Threads</th>
<th>64_18_64</th>
<th>142_40_10</th>
<th>142_40_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>22.5</td>
<td>248.4</td>
<td>176.8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12.1</td>
<td>125.6</td>
<td>95.9</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>9.5</td>
<td>113.0</td>
<td>90.2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6.7</td>
<td>78.2</td>
<td>87.3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>5.9</td>
<td>94.8</td>
<td>61.9</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6.4</td>
<td>80.2</td>
<td>52.4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8.1</td>
<td>126.1</td>
<td>101.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.3</td>
<td>107.8</td>
<td>84.6</td>
</tr>
</tbody>
</table>

Table 7.13: Mixed-mode ART algorithm - Run Time (s)

The efficiency of the two processor jobs should be considered similar to the Open-MP parallelisation of the inner loop, because now communication via MPI takes place between nodes. It is interesting to note that the 2x8 decomposition performs better than the 4x4 decomposition in all three test cases. The highres_highview test case runs significantly better in the 2x8 decomposition. The limited communication via the message passing interface might be one of the reasons for the good performance. The results of the speed up and efficiency are shown in the figures below.
The algorithm performs a lot better compared to the message passing implementation. However the relaxation factor, communication overheads and extra iterations still hinder faster convergence times.
7.4.2 MART

The time to convergence measurements for the mixed-mode parallel MART algorithm are shown in Table 7.14. The results look very promising, especially the speed up, and therefore resulting efficiency plots for the small test case. The data in memory is very small especially for decompositions with a high number of processors. Therefore data can be stored in a higher cache level, which leads to faster data access. In addition the MART algorithm allows for bigger corrections, it requires less additional iterations due to averaging results and zero-values are not updated. The maximum speed up was achieved with a 2x4 decomposition and resulted in an efficiency of 97%.

<table>
<thead>
<tr>
<th>Num. of Proc</th>
<th>Num. of Threads</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>15.1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7.7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3.9</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2.7</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1.9</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>3.3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.9</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 7.14: Mixed-mode MART algorithm - Run Time (s)

The results of the high resolution test case with a high number of views do not look promising. The additional iterations are the reason for the bad performance, unlike the other two test cases, which scale well. If the algorithm is run on a local cluster, the results would be regarded as satisfactory because the run time was reduced by a factor of 2.5 with an efficiency as low as 0.15.
The results of the medium test case are satisfying. A maximum speed up of 3.5 was achieved on a 2x4 decomposition, which results in an efficiency of over 40%. Due to the averaging of the results, no collisions occurred in any of the three test cases which allowed for the correction to
be done in place.

### 7.5 General Comments

Overall the parallel MART algorithm scales better than the parallel ART algorithm. The parallel ART algorithm requires more additional iterations than the equivalent MART algorithm. The additional iterations are a result from averaging the reconstruction over the number of processors. Furthermore the ART algorithm uses a relaxation factor for stability in convergence. The relaxation factor limits the correction in each time step, whereas the nature of the MART algorithm allows for greater correction which limits the impact of the averaging. The MART algorithm benefits from its multiplicative nature with regards to 'zero' values. If one voxel has been set to zero, its value will not be changed any longer. This limitation does not exist in the ART algorithm; the algorithm actually allows for negative corrections. In parallel this might lead to collisions between values on different processors. The ART algorithm performs better with test cases of high number of views. This theory has only been tested on three similar test cases and requires further investigation by considering a test case with a medium number of views.

The MPI results show a great deal of variance in time to convergence. Detailed investigations should help understanding the source of the variance. In general the MPI implementation suffers from communication overheads, especially the parallelisation of the inner loop. In this version, results are communicated after each view has been considered. This leads to an insufficient work load / communication ratio and should be avoided. The Open-MP implementation allows for the synchronisation in memory, therefore the performance does not suffer from communication latency. However the performance is still influenced by an uneven work load balance in the static decomposition because corrections are only carried out on non-zero elements. An appropriate data decomposition can be easier achieved with Open-MP by using a different schedule clauses. The Open-MP version is superior in the small grain parallelisation. The mixed-mode version was implemented to use the advantages of both shared and distributed memory implementations. The mixed-mode showed good speed ups and allows for greater scalability with good performance results compared to the pure MPI and Open-MP versions. With an increasing popularity in multi-core chips, the mixed-mode version should be tested and developed further for test cases with higher resolutions (~640x480 pixels). Nonetheless implementing a mixed-mode version is also more difficult to implement and the code is harder to maintain.

Good speed ups were achieved in tests with low number of processors. The efficiency often dropped with an increasing number of processors (~8) and the use of a high number of CPU needs to be questioned for the given problems. The parallel implementations seem more suitable to work station clusters than super computers. Test cases with a very large number of views and very high resolutions were not available due to the costs and limitations of the experimental equipment.

The large test case with a high resolution (142x40 pixels) and a high number of views (64 views) had 6 times as much data compared to the medium test case with the same resolution. Therefore the large test case requires fewer iterations for the reconstruction (~5 times less) and
less run time (∼30%) on one processor. This trend only accounts for the sequential algorithms. In parallel, the case with higher number of views shows better performance overall, but differences are much smaller.
Chapter 8

Conclusion

Further investigations are required to fully understand and improve the combustion process in order to develop more sophisticated and efficient burners. Scientists and engineers use experimental and computational studies to investigate the turbulent flame structures. Floyd [16] developed a sensor technique that uses computed tomography of chemiluminescence to reconstruct flame structures. CT uses a set of projection measurements of a field in order to estimate the field. The successful reconstruction of the three dimensional objects is often limited by the computational efforts required.

Modern computers allow the exploitation of parallel computing and hence two CT algorithm (ART and MART) were parallelised for different memory systems. A shared memory parallelisation was developed using the Open-MP directives; whereas the distributed memory version uses a message passing interface (MPI) to communicate between processors. Modern computer chips consist of multiple cores and to take advantage of that architecture a mixed-mode version was created to maximise performance improvements.

The different implementations were tested on three different test cases. A small test case consisting of a high number of views at low resolution, a medium test case consisting of a low number of views at high resolution and a large test case which had a high resolution and a high number of views.

The MPI versions showed a great deal of variance in the results. The gain of executing the inner loop of the algorithm was lost in communication overheads and additional iterations required as a results of the reconstruction averaging. The outer loop parallelisation showed good reduction in run times for the MART algorithm and the ART algorithm low processor numbers. At higher number of processors a speed up was achieved at low efficiency.

The Open-MP implementation also suffered from the averaging of the results as well as synchronisation overheads due to the static decomposition. Alternative decompositions showed better speed ups and results improved in the small grain implementation (inner loop). Yet again the efficiency dropped at higher numbers of processors. The scalability is limited by the available shared memory.
The mixed-mode parallelisation results were better than results with MPI. Good speed ups were achieved with moderate efficiencies. The implementation also scales better than the Open-MP version and is well suited for modern computer architectures with multi-core chips. The work load balance was maximised by choosing the best schedule clause. However, the performance is still limited by the communication overheads and the averaging of results.

The two algorithms behaved differently. The ART algorithm seems to suffer from a relaxation factor and higher number of iterations. The MART algorithm is more stable with noisy data compared to ART. It also leads to faster convergence times, sequentially as well as parallel.

Overall the parallelisation is probably not suited for super computers due to the limitations of the algorithms and their inefficiency. The main limitations were the additional iterations due to the averaging and the communication/synchronisation overheads. More processors often resulted in higher communication overheads. Good performance improvements were achieved on smaller number of CPUs which makes the problem more suitable for local clusters that consist of a few multi-core chips (≈ 20). The results were very case dependent and different test cases might show better performance data.
Chapter 9

Future work

The following chapter outlines some suggestions for future work to improve the performance of the parallelisation and to continue the research in the field of combustion.

The MPI performance was limited by the communication overheads and the additional number of iterations required for the solution to converge. The communication pattern cannot be influenced because of the nature of the algorithm. On the other hand the reduction of the reconstruction from each processor can be investigated further. Averaging the results limits the correction in each iterations and better solutions might be possible. A short investigation was carried out into using the maximum or minimum value from all processors. This lead to an over- or under-prediction of the reconstruction and so it is not suitable for the reconstruction technique. A waited average could be used based on the weight factor used in the correction calculation. Even so using a weight factor increases the communication because weights need to be known in order to take them into consideration.

A different output format might be more portable. HDF5-files have their advantages in terms of storing three dimensional fields, but certain libraries are required, which limits the portability of the code, unless results are always viewed on the same computer. An alternative format, such as tec-files, might be equally suitable to store three dimensional data. Routines to write tec files are easy to implement and no special libraries are required. This makes the format very portable and files can be viewed with open-source software.

The parallelisation should also be evaluated on larger test cases. Test cases with large number of views (\(\sim 60\)) and a very high resolution (\(\sim 640 \times 480\) pixels) should be tested and evaluated. The different implementations might behave slightly different under very heavy loads. Unfortunately it was not possible to create such a test case during this project because the laboratory underwent some reconstruction and phantom data could not be created.

Floyd [16] has shown on a pilot scale that the technique works very well. The algorithm was computationally too expensive for high resolution reconstructions based on a high number of views. The parallelisation of the algorithms enabled a reduction in time to convergence. Hence the technique can be developed for a larger number of cameras with higher resolution and...
applications that help with real scientific and engineering problems. The break through success by Floyd should be continued to develop, validate and apply a new experimental technique that permits the investigation of the geometry and motion of turbulent flames in three dimensions (space and time) proving greater insight into combustion.
Bibliography


Appendix A

Additional Results

The following section shows additional results of the parallelisation of the ART and MART algorithms. More detailed results of the non-zero alteration to the ART algorithm are presented. The Open-MP implementation has also been tested with alternative schedule clauses and chunk sizes to maximise the work load balance.

The ART algorithm was changed to only consider voxels that are of value bigger than 0.0. A significant reduction in time to convergence was recorded in the large test case on a large number of CPU’s. No significant improvements were noticed in the other two test cases. Figure A.1 shows the speed ups for the different test cases for the ART algorithm initialised with 1.0 (ie. 142_40_64_zero) and the updated ART algorithm that only considers non-zero values initialised with 1.0.

![Speed up ART Algorithm - only non-zero](image)

Figure A.1: MPI outer loop ART algorithm - Speed up - updating only non zero voxel
Figure A.2 shows the run time for the parallel ART algorithm over the inner loop using the shared memory directive Open-MP with different schedule clauses and chunk sizes. The chunk size and schedule clause have an effect on the time to convergence. For example the dynamic schedule with chunk size 5 does not perform as well as most of the other decompositions. In general the majority of schedule clauses and chunk sizes result in similar run times. Big improvements in the time to convergence are noticed for small number of processors before the synchronisation overheads limit the potential speed up on higher number of processors.

![Open-MP inner loop- Schedule Clauses - Run Time (s)](image)

Figure A.2: Open-MP inner loop ART algorithm - alternative schedule clauses - Run Time (s)

As mentioned in the main report, the results from the shared memory implementation are averaged over the number of processors at the end of each iterations. As a result of the averaging, additional iterations are required to compensate for under estimating the reconstruction. The number of iterations depends on the number of processors and the reconstruction problem. Table A.1 gives an overview of the number of iterations required for a reconstruction of the individual test cases using the parallel ART algorithm with a convergence factor of $1e^{-6}$. The percentage change of additional iterations is also shown. Some cases require as little as 15% extra iterations, where as some decompositions required more than 2.5 times as many iterations. However all measurements resulted in some reduction in time to convergence.
The parallel MART algorithm behaves differently. The two smaller test cases require less iterations in parallel for lower number of CPUs, which leads to the speed up. Hence the non-linear speed up is mainly a result of unbalanced work load and communication overheads. On the other hand, the large test case requires more iterations in parallel. Although speed ups were also recorded for this test case, it is clearly limited by the extra number of iterations required. A more detailed study is necessary to fully understand the behaviour of the large test case.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Number of Iterations</th>
<th>Change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64 142_40_10 142_40_64</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>180 1068 192</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>250 1563 221</td>
<td>38 46 15</td>
</tr>
<tr>
<td>4</td>
<td>343 2140 382</td>
<td>90 100 98</td>
</tr>
<tr>
<td>6</td>
<td>429 1958 551</td>
<td>138 83 186</td>
</tr>
<tr>
<td>8</td>
<td>557 1547 347</td>
<td>209 44 80</td>
</tr>
<tr>
<td>10</td>
<td>529 1638 677</td>
<td>193 53 252</td>
</tr>
<tr>
<td>16</td>
<td>418 N/A 565</td>
<td>132 N/A 194</td>
</tr>
</tbody>
</table>

Table A.1: ART - Number of iterations required for number of processors using MPI

Table A.2: MART - Number of iterations required for number of processors using MPI

Table A.3 gives an overview of the number of iterations and the run times of the alternative approach where the average of the changes has been recorded. The approach worked best for test cases with low number of views. The higher number of view test cases suffered from instability at high number of processors and convergence was slow. Especially the small test case suffers at high number of processors and the performance is worse than the chosen approach. Therefore the averaging approach of the reconstruction is the prefered method.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Number of Iterations</th>
<th>Change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64 142_40_10 142_40_64</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>112 497 85</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>45 451 171</td>
<td>-59 -9 101</td>
</tr>
<tr>
<td>4</td>
<td>32 413 228</td>
<td>-71 -16 168</td>
</tr>
<tr>
<td>6</td>
<td>90 77 253</td>
<td>-20 -84 197</td>
</tr>
<tr>
<td>8</td>
<td>110 88 337</td>
<td>-1 -82 296</td>
</tr>
<tr>
<td>10</td>
<td>155 191 349</td>
<td>38 -62 310</td>
</tr>
<tr>
<td>16</td>
<td>119 N/A 315</td>
<td>6 N/A 270</td>
</tr>
</tbody>
</table>

Table A.2: MART - Number of iterations required for number of processors using MPI

Table A.3: Number of iterations and run times for an alternative approach

<table>
<thead>
<tr>
<th>No. of Processors</th>
<th>Number of Iterations</th>
<th>Run Times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64_18_64 142_40_10 142_40_64</td>
<td>64_18_64 142_40_10 142_40_64</td>
</tr>
<tr>
<td>1</td>
<td>180 1068 192</td>
<td>26.4 200.1 191.2</td>
</tr>
<tr>
<td>2</td>
<td>282 1329 221</td>
<td>16.7 172.6 176.3</td>
</tr>
<tr>
<td>4</td>
<td>432 1869 382</td>
<td>15.8 167.3 156.1</td>
</tr>
<tr>
<td>8</td>
<td>557 1855 437</td>
<td>22.6 206.1 115.3</td>
</tr>
<tr>
<td>10</td>
<td>661 1638 400</td>
<td>22.5 233.9 120.2</td>
</tr>
</tbody>
</table>

Table A.3: Number of iterations and run times for an alternative approach
Appendix B

Experiments

The following section outlines the experimental work by Floyd [16] in more detail. Floyd undertook a phantom study to determine the influence of the number of views on the reconstruction. It is easy to see how the view resolution determines the upper limit of the reconstruction resolution. On the other hand, it is less clear how the number of views affects the reconstruction resolution. Therefore Floyd looked more closely at the relationship between the reconstruction resolution and number of viewing angles [17]. He found that a minimum of 10 views is required to reconstruct an object successfully.

Another point of focus in his work was the application of CTC for the reconstruction of a premixed turbulent opposed jet. The CTC sensor had been implemented with multiple cameras and used to measure the premixed flame in a Turbulent Opposed Jet (TOJ) burner. These types of burners were developed to establish idealised flamelets in a compact domain. They offer good optical access and the possibility for simple numerical simulation. The original development was driven forward at Imperial College in London by Mastorakos et al. [34] and later on by Lindstedt [30]. The burner examined by Floyd was designed and manufactured at TU Darmstadt, Germany. Figure B.1 shows an illustration of the burner set up.
Each nozzle provides a premixed fuel/air mixture and is concentric with a larger diameter nozzle that provides a stabilising co-flow of air. Both streams have the same equivalence ratio and speed. As a result the stagnation point is an equal distance from both nozzles. The twin flame brush is stabilised by the two streams and centered at the stagnation point. The turbulence in each nozzle is created by a turbulence generating plate (TGP), which is a metal sheet perforated with holes. Floyd burned methane in air with the mixtures set using Bronckhorst mass flow controllers. To avoid soot, only lean mixtures were considered. Floyd used Leutron PicSight P32M cameras, which were synchronously triggered to get instantaneous flow field data. The synchronous cameras were triggered by an on/off external electronic signal, which triggers an exposure of predefined length. The system allows the use of up to 127 cameras with a maximum resolution of 656 x 494 pixel.
Appendix C

Script

A script (fig. C.1) was written to create new test cases based on the phantom that was used during the code development. The script generates a predefined number of views as pgm files with different viewing angles. A high resolution test case with a high number of views was created to simulate heavy work loads in the reconstruction. The second test case is a low resolution case with a high number of views. The small test case represents an experimental set up with a high number of commercial cameras at low resolution.

```
li="View pixels [i,j] = 142 40"
lo="Angle about Y (rad) = "
lx="Angle about X (rad) = 0.0"
Projection form = cylo
Focal length (n) = 35e-3
Focal ratio = 1.6
Object distance (n) = 2
 CCD Pixel size (n) = 9.9e-6
View width (n) = 0.1
Principle point [i,j] = 20 7"
i=1
while [ $i -le 64 ]; do
 c=${a}[0.0490874]
j=${i} - 1
 ang=`echo "80 * $j" | bc`
   echo "$li"
 ${lo} $ang
 ${lx} $ang
   i=${i} + 1
 done
```

Figure C.1: Script to generate views based on phantom