Architecture Design for Efficient Non-uniform Fast Fourier Transform (NuFFT) Implementation on FPGA

BY

ALEX IACOBUCCI
B.S, Politecnico di Torino, Turin, Italy, 2014

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2017

Chicago, Illinois

Defense Committee:

Rashid Ansari, Chair and Advisor
Mojtaba Soltanalian
Mariagrazia Graziano, Politecnico di Torino

This is for you, Grandpa.

ACKNOWLEDGMENTS

This dissertation is the final work of my Master Graduate studies, that I have been pursuing for the last two years. I have had the chance of being part of a unique double degree program between Politecnico di Torino and University of Illinois at Chicago, for which I am very grateful. It was a fulfilling experience, and it gave me the opportunity of exploring a reality I have always dreamed being a part of. I would like to thank Lynn Thomas for all of her constant help in successfully managing the difficulties that arose during the project.

Financially supporting myself during this period was not easy, and a big thank you goes to the Department of Electrical and Computer Engineering at UIC, funded by the grant 2016-03393 issued by Intel Altera, for its support.

Professor R. Ansari from UIC, Professor M. Graziano from Politecnico di Torino, G. Nash and U. Cheema contributed to the research with valuable advice and expertise, and I would like to thank them for assisting me during this process.

A further thank you goes to my friend and colleague Emanuele, whom I have shared the research process with, and that made it much more enjoyable. A heartfelt thanks also to my friends, which have been and are a fundamental support and essential part of my life, both to those that are physically present here in the United States, and those that I still feel close to me, although they are in a different country.

Finally, I would like to thank my family from the bottom of my heart, in particular my mother and my grandmother, two of my strongest pillars. If it wasn't for them, who constantly

believed in me, supported me in any possible way, and pushed me to always reach for the best, I would not be here.

AI

CONTRIBUTIONS OF AUTHORS

In the first Chapter, I provide the reader with an Introduction to the problem, an outline of this Thesis, and a description of the most relevant existing work. Chapter 2 contains a physical and mathematical background of the MRI scan mechanism and its relation to NuFFT. In Chapter 3, I state possible approaches to solve the MRI reconstruction problem, focusing on Re-Gridding. Figure 6, used when describing Density Compensation, is taken from E. Pezzotti in (1). Permission for the use of the image has been asked and given, as proved in Appendix B. Chapter 4 contains a description of the OpenCL architecture designed and implemented for the MRI reconstruction process as part of the research project, focusing on techniques for kernel and deapodization computation. In Chapter 5 I tackle the choice of the optimal convolution kernel. Chapter 6 describes methodologies to achieve software parallelism, and trajectory re-ordering mechanisms, both in software and hardware. Chapter 7 lists the results of the synthesized architectures. Figure 31 and Figure 32 are borrowed from the paper submitted to FPL 2017, which was accepted as a poster (19), and are therefore shared with E. Pezzotti. Finally, further possible developments are described, such as an extension to 3D imaging.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

MRI	Magnetic Resonance Imaging
NMR	Nuclear Magnetic Resonance
NuFFT	Non-uniform Fast Fourier Transform
FPGA	Field Programmable Gate Array
SSIM	Structural Similarity Index Metric
PSNR	Peak Signal to Noise Ratio
HDL	Hardware Description Language
UIC	University of Illinois at Chicago

SUMMARY

The pivotal role of Fourier Analysis in discrete-time signals processing has been firmly established and is common knowledge. Consequently, several algorithms and techniques have been developed to perform efficient computation of Forward and Inverse Discrete Fourier Transform, with the Fast Fourier Transform (FFT) at the top of the list. Most of these algorithms have been designed to deal with the common case of data sampled on regular rectangular grids. However, in some relevant real life applications, the input signal is not sampled uniformly, hence raising the issue of computing an accurate yet computationally efficient Fourier Transform. The algorithm for meeting this challenge is usually referred to as Non-uniform Fast Fourier Transform.

A critical application involving irregular data sampling is Magnetic Resonance Imaging (MRI). MRI has quickly won its place among the most used medical diagnosis techniques, due to its lower impact on the human body and its high accuracy in distinguishing abnormal tissue from normal tissue. One of the drawback in MRI is the long acquisition time, and the unwanted artifacts that can be present in the reconstructed image. To overcome both of these disadvantages, non-Cartesian acquisition trajectories have been designed and are now often employed in actual acquisitions. However, while they offer lower scan times and aliasing, the samples are not regularly spaced and hence introduce the challenge of efficiently computing a NuFFT.

In view of the relevance of the problem, this thesis investigates an improved implementation of NuFFT by seeking to exploit the promise of FPGA-based acceleration. The goal of this thesis is to provide an effective implementation using OpenCL, a recent higher level language, to solve

the problem in the context of MR Imaging. The objective is to prove that by combining OpenCL and FPGAs it is possible to achieve results valuable from both an academic and commercial standpoint, while taking advantage of the simpler design process provided by OpenCL.

This work focuses on demonstrating that by following the guidelines of OpenCL programming, and exploiting the advances in FPGA technology, it is possible to build solutions that are competitive with respect to academic standards. To begin with, the host code has been optimized to achieve high parallelism and overcome the bottlenecks induces by disk operations. Diverse approaches were designed and synthesized to exploit known acquisition trajectories in order to boost performance. By analyzing the interpolation kernels and their impact on the reconstruction Structural Similarity Index Metric (SSIM), optimal values have been identified for the convolution window size and the oversampling factor, yielding a satisfactory trade-off in accuracy and computational time. Furthermore, this work shows that while tuning convolution parameters can significantly impact both performance and accuracy, a proper knowledge of the dependency of the reconstruction on the convolution kernel guarantees the chance of achieving the desired throughput/accuracy values, thereby making the proposed architecture suitable for a wide range of applications.

CHAPTER 1

INTRODUCTION

While most of common real world problems in signal processing involve regularly sampled data, some applications, especially in the medical field, can acquire samples non uniformly in Fourier domain, introducing the challenge of Non-uniform Fast Fourier Transform. Magnetic Resonance Imaging is one of the main application where sampling patterns are not necessarily Cartesian.

The importance of MRI in medical imaging has already been stated and discussed. Both academic and market research have focused abundantly on trying to develop increasingly efficient solutions: CPU approaches were the first to be investigated, and proved that pure software was unable to achieve throughput values suitable for modern MR applications - real time imaging to name one. The need for hardware acceleration soon became evident, and parallel processing with GPUs and languages like CUDA were investigated and often used for commercial applications. Recently, technology improvements led to stunning enhancements in FPGA performance, to a point where optimized VHDL/Verilog FPGA implementations can compete with those running on GPUs. Consequently, a renewed interest arose with regards to FPGA-based solutions.

At the same time, the trend in development tools leans more and more towards higher level languages, that leverage the difficulties in platform compatibility while providing simpler design frameworks. The latter feature, which is extremely important as the complexity of the system scales up. OpenCL is one of these modern higher level languages, and with its C like syntax it

provides a flexible tool to describe parallel processing applications that can work on multiple hardware platforms.

From the standpoint of our research, we have to thank Altera (now part of Intel) for the grant which funded the research this work is a result of. They provided the economical means to sustain the research, together with the latest FPGA technology and the technical support from some of their experts. Thanks to the grant, a research group was initiated, with the main objective of proving the suitability of Altera OpenCL and Altera FPGAs for efficient and competitive MRI solutions. A main OpenCL architecture has been developed and tested, and a poster has been submitted and accepted at the IEEE FPL 2017 conference. The proposed architecture has then been further investigated, focusing on different aspects, and two Master Thesis have been proposed, specifically this one and the one by E. Pezzotti [1]. Due to the fact that the proposed architecture is shared, and that some parts of the research were cooperative in nature, the two theses present a common starting point, and are bound to mutually cite each other when certain topics are covered.

1.1 Previous Works

Due to the relevance of efficient implementation of MRI reconstruction in the medical world, it has been the focus of extensive academic research. Sorensen et al. in [2] developed a solution tackling the entire MRI reconstruction problem in all of its phases for both CPU and GPU computing devices, using HLSL for the latter. Kestur et al. in [3] start targeting FPGAs, but focus on the Re-Gridding phase only. FPGA implementation was also targeted by Cheema et al. in [4], where they propose a novel architecture in Verilog to enhance power efficiency in the

Re-Gridding phase only. A more recent work by Li et al. in [5] proposes a Labview FPGA implementation tackling most of the reconstruction phases.

Current literature seems to lack analysis of comprehensive solutions where the reconstruction process is analyzed deeply as a whole and then implemented on FPGA devices by means of high level description languages. This work aims at filling this gap: FPGAs have proven to be a very convenient platform, especially when the MegaFlops/Watt are considered as an optimization index, and are hence worth investigating; moreover, higher level languages guarantee portability, together with easier development and shorter time-to-market.

1.2 Thesis Outline

This work begins with a physical and mathematical description of MRI. A set of equations are formulated to justify why it is mathematically meaningful to reconstruct an image from raw data as an Inverse Fourier Transform. The concept of Non-uniform Fast Fourier Transform is then stated, together with its relation to MRI. Some of the possible approaches to solve the NuFFT are then described, with particular focus on Re-Gridding. The deapodization phase is tackled as well, and the two approaches to compute its values (on-the-fly, LUT-based) are implemented and compared.

Afterwards, the problem of the optimal choice for the convolution kernel is examined. The most commonly used interpolation functions are analyzed according to relevant accuracy criteria. The best set of design parameters to obtain high accuracy while reducing computational time is then experimentally determined. The thesis also discusses the different approaches to compute

the kernel values (direct computation, nearest neighborhood interpolation, linear interpolation), and analyzes their trade off in terms of accuracy and hardware utilization.

The software running on the host machine is re-designed to exploit parallelism and ensure subsequent processing of different frames with no race conditions and maximum efficiency, overcoming the bottleneck of disk operations.

The architecture is proposed and explained in detail, together with the challenges faced and the techniques adopted to solve them. The architecture is then updated using the results from the previous chapters, and compiled to run on the board. Performance results are extracted and compared with the existing literature.

In the final Section, possible extension of this architecture to 3D MRI are briefly discussed.

CHAPTER 2

BACKGROUND

This Section aims at providing a theoretical background to Magnetic Resonance Imaging. We firmly believe that, although the work proposed in this thesis does not necessarily require knowledge of the physical principles behind data acquisition, a comprehensive understanding of the area of interest is fundamental for a deep mastery of the subject.

2.1 Nuclear Magnetic Resonance (NMR)

The physics behind MR Imaging relies on a property of atomic nuclei called Nuclear Magnetic Resonance (NMR). Nuclei are characterized by an intrinsic angular momentum $\mathbf{p}$ called spin. The spin angular momentum is parametrized by the spin quantum number s, so that:

$$|\mathbf{p}| = \frac{h}{2\pi}\sqrt{s(s+1)} \tag{2.1}$$

where h is the Planck's constant.

For some of the nuclei, s equals 0, and the NMR property cannot be exploited. If $s \neq 0$, a magnetic moment μ arises such that $\mu = \gamma p$, where γ is the gyromagnetic ratio. When nuclei with non zero magnetic moment are exposed to a strong external non-aligned magnetic field B_0, their magnetic moment will tend to align in the direction of B_0, being subject to a torque $L = \mu \times B_0$. However, due to the nuclear spin, the magnetic moment does not simply align, but starts a precession motion around the axis defined by B_0, with frequency:

$$\omega_0 = \gamma B_0 \tag{2.2}$$

ω_0 is referred to as Larmor frequency. The gyromagnetic ratio γ is very important in determining the Larmor frequency for a given nucleus, and is a property strictly related to the nucleus atomic composition.

Until now, the phenomena were described according to the principle of the classical mechanical model. However, when working at the sub-atomic scale, the quantum mechanics has to be taken into account. According to the latter, the magnetic moment vector can orient itself with respect to the external field B_0 in q possible ways, where:

$$q = 2s + 1 \tag{2.3}$$

Each orientation corresponds to a certain energy state for the particle.

Consider the hydrogen atom, that constitutes nearly 99% of the human body. Its spin number is $s = \frac{1}{2}$, hence allowing two possible orientation for the magnetic moment. More specifically, one orientation is parallel to B_0 (lower energy state), while the other is anti-parallel (higher energy state). At equilibrium, the population of atoms at the lower energy state exceeds that of atoms at the higher energy state by an amount described by the Maxwell-Boltzmann distribution, and such that:

- Higher temperatures result in more abundant population in the higher energy state.
- A higher magnetic field B_0 results in more abundant population in the lower energy state.

In general, at room temperature, and with a $|\mathbf{B_0}|$ in the common range [1-9] T, the number of hydrogen atoms whose magnetic moment is parallel to the external fields exceeds its anti-parallel counterpart.

The overall macro-effect is the perception of a *bulk magnetization* $\mathbf{M}$ given by the sum of the contributions of each magnetic moment:

$$\mathbf{M} = \sum_{i=0}^{N_m} \boldsymbol{\mu_i} \tag{2.4}$$

The majority of the contributions cancel out, but the over-population of atoms in the lower energy state results in a measurable bulk magnetization in the direction of $\mathbf{B_0}$. $\mathbf{M}$ behaves as a big unique dipole moment. If $\mathbf{M}$ is forced away from its equilibrium position, it will start precessing around B_0 with frequency w_0 equal to the Larmor frequency. It will hence acquire a component on the xy plane, so that:

$$\mathbf{M} = [0, 0, M_{z0}] \rightarrow \mathbf{M} = [M_x, M_y, M_z] \tag{2.5}$$

with $M_x, M_y \neq 0$. The easiest way to tip $\mathbf{M}$ out of equilibrium is applying a magnetic field $\mathbf{B_1}$ perpendicular to $\mathbf{B_0}$ and oscillating at the system resonance frequency, w_0.

If $\mathbf{B_1}$ is switched off, M_x, M_y will go back to zero, following an exponential decay curve. This phenomenon is referred to as *spin-spin relaxation time*; the decay is characterized by the time constant $T2$. While spin-spin relaxation is taking place, M_z will go back to its starting

value M_{z0}, growing at a different exponential rate in a phenomenon referred to as *spin-lattice relaxation*, characterized by the time constant $T1$.

The variation in $\mathbf{M_{xy}}$ and M_z generates a time-varying magnetic field. A properly placed coil can detect the variation in magnetic field through the oscillating voltage produced according to Faraday's law of induction. It can be proved that the magnitude of the voltage $V_m(t)$ perceived at the coil is proportional to M_{z0}. Therefore, we have a way of measuring the strength of the bulk magnetization, which is itself a measure of the *proton density* for the area under study. This type of signal is commonly called Free Induction Decay (FID).

Up to now, we have considered the actions and response of a unique voxel of tissues. The signal that is actually received by a coil is, however, the superposition of the contributions by multiple voxels. Each voxel has a different molecular composition, and will hence provide a FID at a difference frequency, with different T1/T2 and different amplitude. How is it possible to extract information about single voxels from the mixed and confused signal received at the coil?

2.2 Magnetic Field Gradients

Consider the effect of the application of a magnetic field gradient G_x along the x axis, which is superimposed to the constant field $\mathbf{B_0}$. As described earlier, the resonance frequency w_{0i} for voxel v_i is given by the Larmor equation, so that:

$$w_0(x) = \gamma(G_x + B_0) \tag{2.6}$$

Hence, it is possible to obtain voxels with different resonance frequencies based on their position. This technique can be exploited to obtain specific information about each voxel.

In the following Section, the methodologies applied to obtain a 2-dimensional MRI image are described.

2.3 Slice Selection Gradient

The first step consists in selecting one slice of the sample: the acquired image will be a 2D rendition of the selected slice. To achieve this, a permanent magnetic gradient in the direction of $\mathbf{B_0}$ is applied, so that the resonance Larmor frequency varies linearly along the z axis. A slice can be defined as a portion of z such that all the resonance frequencies belong to an interval $\Delta\omega$ centered at a given ω_c. As we have described above, FID is generated by tipping the bulk magnetization vector through the application of an RF signal at the proper frequency. If the RF signal is chosen to include only components in the range $\Delta\omega$, then only that slice will be excited and will contribute to the measured signal at the coil.

Once a slide has been properly selected, what is missing is a way of encoding spatial information in the generated signal. Rephrasing, we need a way of being able to distinguish the contribution given by each individual voxel in the slice.

2.4 Spatial Encoding

Consider a two dimensional gradient $\mathbf{G}(t)$ applied on the selected slice, where $\mathbf{r}$ uniquely defines one voxel. Each voxel experiences a variation in resonance angular frequency with respect to the slice main w_0, so that:

$$\Delta\omega(\mathbf{r}, t) = \gamma\mathbf{G}(t) \cdot \mathbf{r} \tag{2.7}$$

or, going from radians to Hertz:

$$\Delta f(\mathbf{r}, t) = \frac{\gamma}{2\pi}\mathbf{G}(t) \cdot \mathbf{r} \tag{2.8}$$

The frequency difference translates into a shift in phase between signals generated by different voxels. The amount of phase shift accumulated at a generic time t after the application of $G(\mathbf{r})$ at $t = 0$ can be computed integrating the frequency variation over time:

$$\begin{aligned} \Delta\phi(\mathbf{r}, t) &= \int_0^t -2\pi\Delta f(\mathbf{r}, \tau)d\tau \\ &= -2\pi \int_0^t \frac{\gamma}{2\pi}\mathbf{G}(\tau) \cdot \mathbf{r}d\tau \\ &= -2\pi\mathbf{k}(t) \cdot \mathbf{r} \end{aligned} \tag{2.9}$$

Where $\mathbf{k}(t)$ is defined as:

$$\mathbf{k}(t) = \int_0^t \frac{\gamma}{2\pi}\mathbf{G}(t)dt \tag{2.10}$$

Hence, the contribution provided by an individual voxel to the received signal is:

$$\begin{aligned} \rho(\mathbf{r}, t) &= \rho(\mathbf{r})e^{i\Delta\phi(t)} \\ &= \rho(\mathbf{r})e^{-2\pi i\mathbf{k}(t)\cdot\mathbf{r}} \end{aligned} \tag{2.11}$$

The signal received at the coil is the result of all the contribution by the voxels, and can be written as:

$$d(t) = \int_V s(\mathbf{r})\rho(\mathbf{r})e^{-2\pi i\mathbf{k}(t)\cdot\mathbf{r}}dV \tag{2.12}$$

where V is the volume of the slice that has been selected, and $s(\mathbf{r})$ is the coil sensitivity function with respect to the position of that voxel. If we assume the coil to have unitary sensitivity, so that just one coil is used for the whole acquisition, Equation 2.12 becomes:

$$d(t) = \int_V \rho(\mathbf{r})e^{-2\pi i\mathbf{k}(t)\cdot\mathbf{r}}dV \tag{2.13}$$

By taking a closer look at this equation, it can be noticed that the factor $e^{-2\pi i\mathbf{k}(t)\cdot\mathbf{r}}$ is interpretable as a Fourier basis. By rewriting it in terms of $\mathbf{k}$, Equation 2.13 becomes:

$$d(\mathbf{k}) = \int_V \rho(\mathbf{r})e^{-2\pi i\mathbf{k}\cdot\mathbf{r}}dV \tag{2.14}$$

Because of Equation 2.14, the signal received at the coil can be interpreted as the Fourier Transform of the magnetization decay as a function of space. The domain of the Fourier Transform is a set of values for $\mathbf{k}$ that reside in the so-called *k-space*, a suitably introduced mathematical formalism: acquiring values from the coil can be viewed as sampling k-space in the coordinates defined by Equation 2.10.

Since the magnetization decay is directly proportional to the proton density in the tissue under analysis, the objective for imaging is obtaining an expression for $\rho(\mathbf{r})$. This can be achieved through an Inverse Fast Fourier Transform, so that:

$$\rho(\mathbf{r}) = \int_{k-space} d(\mathbf{k}) e^{2\pi i \mathbf{k}\cdot\mathbf{r}} dk_x dk_y \tag{2.15}$$

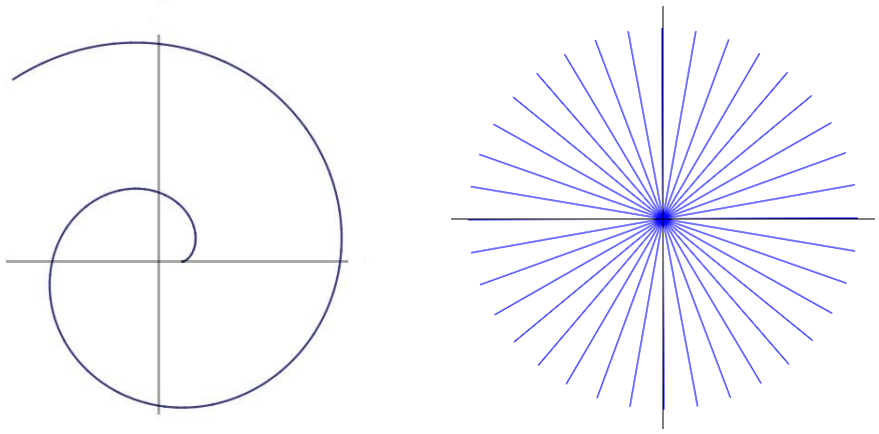
where the z dimension is not included since a single two dimensional slice is being considered.

It is worth mentioning that the values for $d(\mathbf{k})$ are complex numbers, although the signal measured at the coils is obviously real, because of its physical nature. This is due to the fact that the signals are typically acquired with two quadrature coils, a pair of orthogonal coils driven 90° out of phase with each other. Each acquisition corresponds then to a pair of values, one per each coil, that are set as the real and imaginary part of a complex number.

2.5 k-space Acquisition Trajectories

As mentioned above, coil measurements can be interpreted as sampling data in k-space. The coordinates of the samples can be chosen by properly varying the gradients. A set of common trajectories in actual MRI acquisition can be observed in Figure 1. While for a long time Cartesian trajectories were the standard due to the ease in acquisition, they have been lately been substituted by trajectories offering much lower acquisition times and better accuracy in terms of introduced artifacts, such as spiral and radial trajectories.

The choice of a trajectory, and the set of parameters describing a trajectory (e.g. number of spiral interleaves for spiral trajectories, desired spatial resolution, etc.) depends on several factors, and are beyond the scope of this work.



(a) (b)

Figure 1: Set of common acquisition trajectories in actual MRI scans. (a) Spiral trajectory (b) Radial trajectory

2.6 Obtaining the image: the Fourier Transform

Equation 2.15 provides a way of determining the proton density at any desired spatial location **r**. The formula in the Equation corresponds to the mathematical formulation of an Inverse Fourier Transform.

Generally speaking, a Fourier Transform is an operator decomposing a function onto an orthogonal basis. In two dimensions, the starting domain is usually referred to as image domain, while the Fourier domain is commonly referred to as *spatial frequency* domain. The basis for the 2D Fourier Transform is the function:

$$e^{2\pi i(ux+vy)} = cos(2\pi(ux+uy)) + i \cdot sin(2\pi(ux+uy)) \tag{2.16}$$

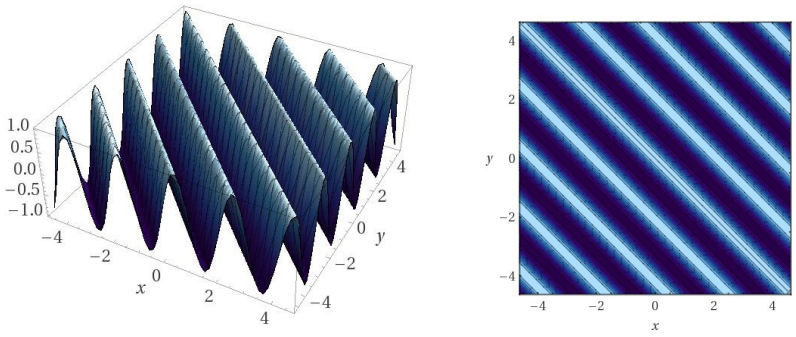


Figure 2: 3D Plot (a) and contour (b) of the real part for one 2D Fourier Transform basis.

and is such that both the real and imaginary part describe a sinusoid on the plane, as shown in Figure 2. The formula describing the $\mathscr{F}$ operator is:

$$M(u,v) = \mathscr{F}\{m(x,y)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} m(x,y)e^{-2\pi i(ux+vy)}dxdy \tag{2.17}$$

where $m(x,y)$ is the starting image and $M(u,v)$ is its spatial frequency domain counterpart.

Going back from the Fourier domain to the image domain can be achieved through the Inverse Fourier Transform operator $\mathscr{F}^{-1}$, whose formula strongly resembles that of the forward Fourier Transform:

$$m(x,y) = \mathscr{F}^{-1}\{M(u,v)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} M(u,v)e^{2\pi i(ux+vy)}dudv \tag{2.18}$$

Exact computation of Equation 2.18 requires knowledge of the analytical expression for $M(u,v)$ and, in most of real life problems, this privilege is not granted. More commonly, we are

provided with $M[u, v]$, a set of sample values on a grid for $M(u, v)$, and we aim at obtaining $m[x, y]$, the values of $m(x, y)$ for a set of pixels on a uniform grid. In this case, the Inverse Discrete Fourier Transform gives a way of computing $m[x, y]$ as:

$$m[x, y] = \sum_{u=0}^{P-1} \sum_{v=0}^{Q-1} M[u, v] e^{2\pi i (\frac{u}{P} x + \frac{v}{Q} y)} \tag{2.19}$$

If k-space is sampled on a Cartesian trajectory, Equation 2.19 can be directly applied to compute the desired image. But what if, as it is often the case, the trajectory is non-Cartesian?

The task of performing a Fourier Transform starting from sample points that are not uniformly sampled is usually referred to as Non-Uniform Fourier Transform. Note that, hereafter, Fourier Transform and Inverse Fourier Transform will be used interchangeably since switching from one to the other translates into switching one sign in the Fourier basis.

CHAPTER 3

THE MRI IMAGE RECONSTRUCTION PROCESS

3.1 Possible Approaches

The problem of reconstructing MRI Images from random k-space trajectories is not new to the scientific world. Several approaches have been developed during the years, each with different trade-offs between accuracy and performance.

3.1.1 Non-Uniform Fast Fourier Transform (NUFFT)

This approach tries to directly apply the Fourier Transformation to the points in k-space. Recall Equation 2.15; after decomposing the vector ρ into its components x and y, the Equation can be written as:

$$\rho(x,y) = \int_{k-space} d(k_x,k_y)e^{2\pi i(xk_x+yk_y)}dk_xdk_y \tag{3.1}$$

A satisfactory approximation of Equation 3.1 can be obtained with the Riemann sum:

$$\rho(x,y) = \sum_i d(k_{ix},k_{iy})e^{2\pi i(xk_{ix}+yk_{iy})}W(k_{ix},k_{iy}) \tag{3.2}$$

where $W(k_x,k_y)$ is a function defining weight coefficients that compensate for the non uniform sampling density.

In its most general form, this algorithm offers the best reconstruction accuracy achievable with the set of sample points S. However, the algorithmic complexity to reconstruct an image of size $N \times N$ from a set of M samples is $O(MN^2)$, making its direct implementation impractical for actual medical imaging. Previous works [6] have tried to approximate some of the computational steps, reducing the complexity to $O(M \cdot log(N)) = O(N^2 \cdot log(N))$ [6], given that M is taken equal to $N \cdot N$. However, the approximation resulted in a loss of accuracy that made the method comparable with simpler and more common approaches like Re-Gridding.

Due to its high accuracy, the summation in Equation 3.2 has been used to reconstruct the reference images for the evaluation of image reconstruction quality in successive Sections.

3.1.2 Iterative Approaches

Another approach investigated in the literature tries to model the problem as a set of matrix equations to be solved with iterative methods. Variants of the least-square and of the conjugate gradient approaches are the most common techniques inspected.

3.2 Interpolation-based Reconstruction

3.2.1 General Idea

The interpolation based approach seeks to reduce the time complexity issues introduced by the IFFT on non regularly spaced k-space samples. To achieve that, the k-space samples are interpolated to recreate the continuous k-space function, which is then re-sampled on a Cartesian grid. The image can then be obtained from the Cartesian grid through a standard 2-dimensional IFFT.

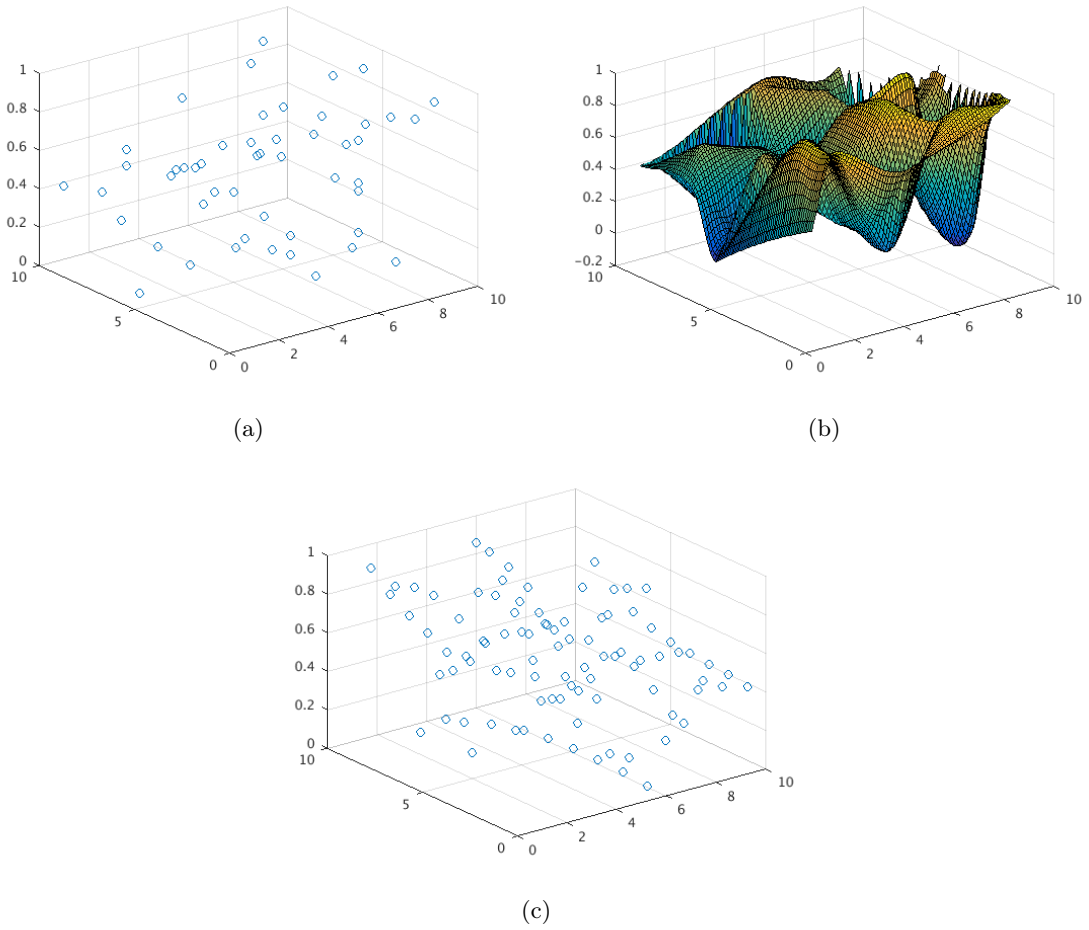


Figure 3: (a) Set of points sampled randomly in k-space. (b) Reconstructed k-space function as a result of the interpolation process. (c) Output of the re-sampling of the interpolated function on a Cartesian grid.

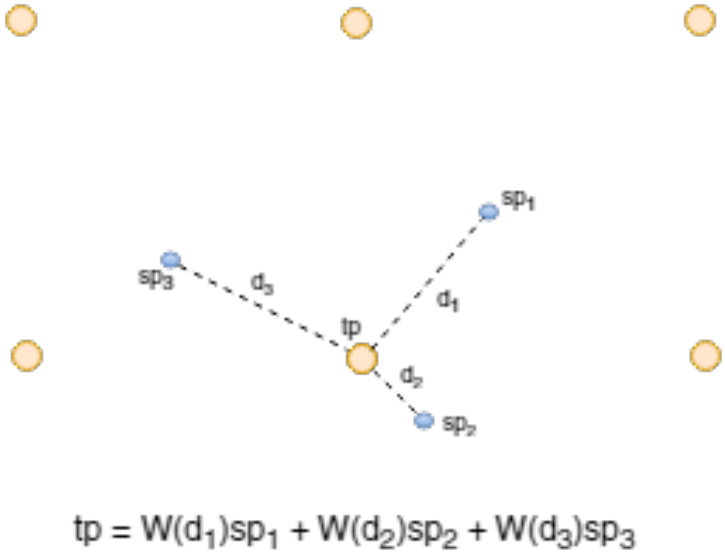


Figure 4: Example of grid point estimation process. $W(d)$ is a weight function depending upon the distance d from the target point.

A simpler interpretation of this approach is the following: the objective is determining a good approximation for the values of $G \times G$ points on a Cartesian grid in k-space. These points will then be transformed via a 2D IFFT to yield the reconstructed image. To estimate the value at each grid point, the values of neighbor sample points are used, after being properly weighted with a coefficient proportional to the distance from the grid point.

The paragraph above provided a hint on the whole reconstruction process which is, however, more complex, and consists of multiple stages, described in the following sections.

3.2.2 Density Compensation

The Density Compensation phase needs to be introduced to make up for the non uniformity in the k-space sampling density. The undesired effects induced by the latter are observable in

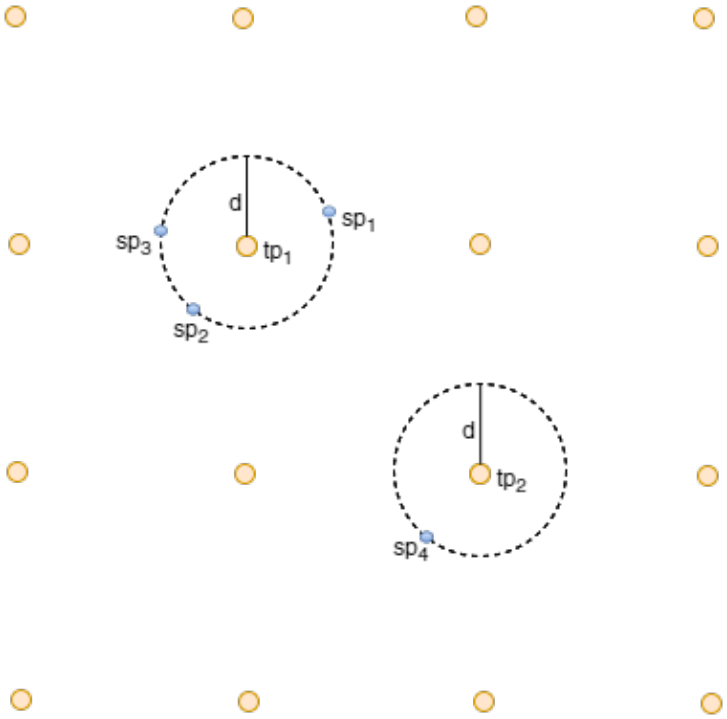
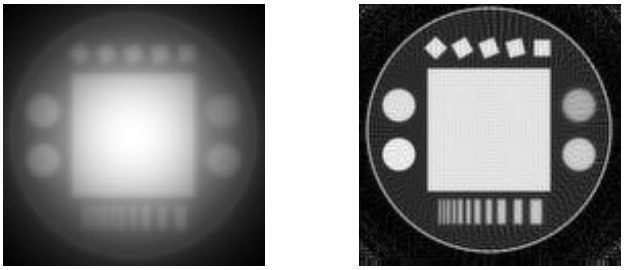


Figure 5: Effect of non homogeneous sampling density.

Figure 5. Consider two grid points tp_1, tp_2 with the same exact value V. Point tp_1 is in an area with high sampling density, where three source points $sp_i, i = 1, 2.3$ are present, all of them at a distance d from tp_1 and with value v_1. The value calculated for tp_1 is then $3\Phi(d) \cdot v_1$, where $\Phi(d)$ is a weighting coefficient depending on the distance d. Point tp_2 is in an area with low sampling density, and only one source point sp_4 is present, at a distance d from tp_1 and with value v_1. The value calculated for tp_2 is then $\Phi(d) \cdot v_1$. It's easy to see that, because of different sampling densities, the value for the two points turn out to be very different.



(a) (b)

Figure 6: Improvements due to proper Density Compensation. (a) Image reconstructed with non-compensated source points. (b) Image reconstructed with compensated source points.

Density compensation techniques cope with this problem, and can guarantee significant improvements in image reconstruction quality, as shown in Figure 6. The theory and implementation of Density Compensation has been covered more in depth by E. Pezzotti in [1], and will not be treated here. For sake of completeness, we will assume that Density Compensation has already been computed when referring to the sample points function $M_s(k_x, k_y)$.

3.2.3 Re-Gridding

Re-Gridding is the stage where the k-space function is approximated through an interpolator and re-sampled over a Cartesian grid. When describing the theory behind Re-Gridding, two main phases can be identified.

Interpolation phase

During the interpolation phase, the objective is reconstructing $\hat{M}(k_x, k_y)$ to approximate $M(k_x, k_y)$, the k-space 2-dimensional function that has been sampled during the MRI Scan.

$M(k_x, k_y)$ is a function $R^2 \rightarrow C$ depending exclusively on k_x and k_y, coordinates used to identify a generic point in k-space. The set of sample points available can be represented by:

$$M_s(k_x, k_y) = M(k_x, k_y) \cdot S(k_x, k_y) \tag{3.3}$$

where S is a set of two-dimensional Dirac deltas centered at the source points coordinates:

$$S(k_x, k_y) = \sum_i \delta(k_x - k_{xi}, k_y - k_{yi}) \tag{3.4}$$

In its broader sense, interpolating a function given a set of its points means finding the values for the function by combining the available samples. Interpolation methods can be described in terms of convolution:

$$\hat{M}(k_x, k_y) = M_s(k_x, k_y) * C(k_x, k_y) \tag{3.5}$$

where the function $C(k_x, k_y)$ is referred to as the interpolation kernel function. How does convolution result in interpolation? The 1D case is shown for simplicity, but the argument can be easily extended to two dimensions. Consider a set of source points uniformly sampled with interval $T = 1$:

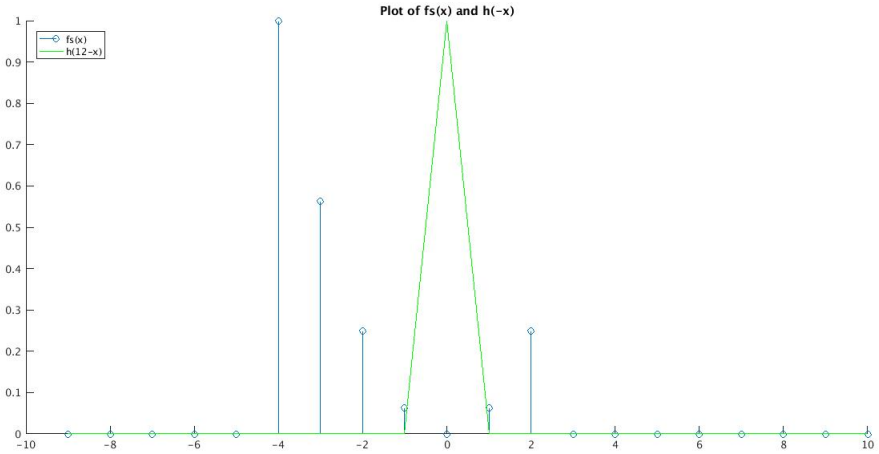
$$f_s(x) = f(x) \cdot \sum_{i=0}^{N} \delta(x - x_i) = \sum_{i=0}^{N} f(x_i)\delta(x - x_i) \tag{3.6}$$

and a triangular interpolation kernel with width $2T$. To reconstruct the value at point x_0, as in Figure 7, according to the formula for convolution, we have to flip the kernel function $h(x)$, translate it with an offset x_0, and then compute the integral:

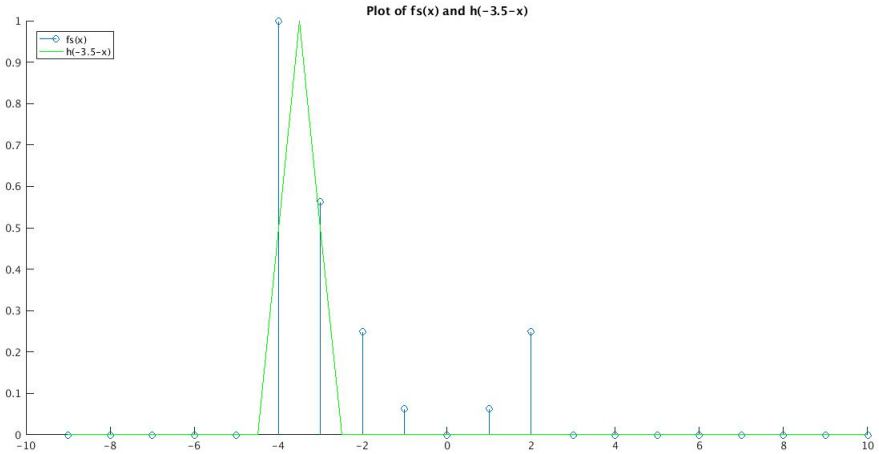
$$\begin{aligned}\hat{f}(x_0) &= \int_{-\infty}^{\infty} f_s(\tau)h(x_0-\tau)d\tau \\ &= \int_{-\infty}^{\infty} (\sum_{i=0}^{N} f(x_i)\delta(x-x_i)) \cdot h(x_0-\tau)d\tau \\ &= \sum_{i=0}^{N} \int_{-\infty}^{\infty} f(x_i)\delta(x-x_i) \cdot h(x_0-\tau)d\tau \\ &= \sum_{i=0}^{N} f(x_i)h(x_0-x_i)\end{aligned} \tag{3.7}$$

The value obtained corresponds to a weighted average of the two samples closest to x_0. It can be proven that convolution with that particular triangular kernel corresponds to linear interpolation. By properly choosing and tuning the kernel function, interpolation can become more and more accurate. To understand what are the characteristics of an ideal kernel function, consider what happens in the image domain.

The sampling operation in k-space introduces aliasing in the image domain. While for uniform sampling it is possible to predict the position and amplitude of the aliased replica, the situation differs for non uniform sampling. Studies [7] have shown that non uniform sampling introduces both in-band and out-band aliasing noise, and that the amplitude and width of the replica varies depending on multiple factors; it is hence not possible to give an exact mathematical formulation of the aliasing noise. Nonetheless, reconstructing the image requires a low-pass



(a)



(b)

Figure 7: Example of convolution computation for $x_0 = -3.5$

filtering operation on the area of interest, and a cut-off of the undesired noise. Calling $c(x, y)$ the function used for low-pass filtering, to reconstruct the image the following must be computed:

$$\hat{m}(x, y) = \mathscr{F}^{-1}\{M_s(k_x, k_y)\} \cdot c(x, y) \tag{3.8}$$

which corresponds, in Fourier domain, and due to the properties of Fourier Transform, to the following convolution:

$$M_s(k_x, k_y) * \mathscr{F}\{c(x, y)\} = M_s(k_x, k_y) * C(k_x, k_y) = \hat{M}(k_x, k_y) \tag{3.9}$$

The function $C(k_x, k_y)$ is exactly the interpolation kernel function. It has been proved that the ideal low-pass filter function is a 2-dimensional gate: consequently, the ideal interpolation function is its Fourier Transform, a *Sinc* function:

$$\sqcap(x, y) \xrightarrow{\mathscr{F}} k_x \cdot Sinc(\pi k_k) \tag{3.10}$$

Unfortunately, *Sinc* functions have infinite support, which implies that we should store and compute the convolution for many sample values, making it not practical to use. Consequently, a windowed kernel is generally used, that is a kernel which limited support. More insights on the choice of an optimal kernel function can be found in Section 5. It is also worth noticing that, for this application, the convolution is circular: sample points at the edge of k-space will influence points on the other side of the spectrum.

Re-Sampling phase

After $\hat{M}(k_x, k_y)$ has been calculated through interpolation, it is sampled over a Cartesian grid so that standard FFT algorithms can be applied on it. By convention, k-space is scaled so that $k_x, k_y \in [-\frac{1}{2}, \frac{1}{2}]$. This operation can then be described mathematically by:

$$\hat{M}[k_x, k_y] = \hat{M}(k_x, k_y) \cdot \text{Ш}(Gk_x, Gk_y) \tag{3.11}$$

where $\text{Ш}(Gk_x, Gk_y)$ is the Shah function, defined as:

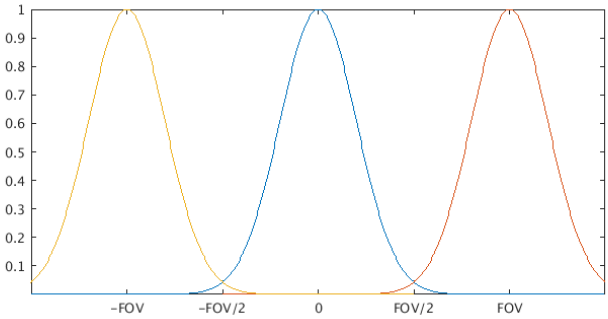
$$\text{Ш}(Gk_x, Gk_y) = \sum_{n=-\frac{G}{2}}^{\frac{G}{2}} \sum_{m=-\frac{G}{2}}^{\frac{G}{2}} \delta(k_x - \frac{n}{G}, k_y - \frac{m}{G}) \tag{3.12}$$

So that k-space is sampled uniformly on a $G \times G$ grid. The number of samples in the Cartesian grid, G, is tunable, and is usually expressed as $G = \alpha N$, where N is the number of pixels on one size of the target image, and α is a tunable factor called *oversampling factor*.

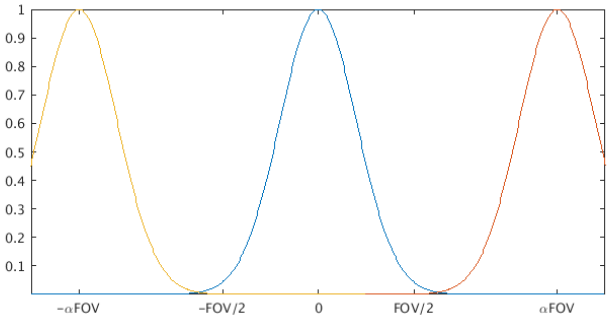
As every standard uniform sampling operation, the Re-Sampling phase introduces aliasing artifacts in the image domain. In particular, consider Figure 8 for the one-dimensional case. The image is replicated at every αFOV, so that increasing α guarantees lower aliasing from the replica.

3.2.4 2D Fourier Transform

The output of the Re-Gridding phase is a $G \times G$ matrix of k-space samples. To finally revert back to the image domain, an Inverse 2D Fourier Transform operation is applied. The equation for the reconstructed image is then:



(a)



(b)

Figure 8: Image replication due to Re-Sampling for the 1-dimensional case. (a) Replication with $\alpha = 1$. (b) Replication with $\alpha = 1.25$.

$$\begin{aligned}\hat{m}[x,y] &= \mathscr{F}^{-1}\{\hat{M}[k_x,k_y]\} \\ &= \mathscr{F}^{-1}\{\hat{M}(k_x,k_y)\cdot \text{Ш}(Gk_x,Gk_y)\} \\ &= \mathscr{F}^{-1}\{((M(k_x,k_y)S(k_x,k_y))*C(k_x,k_y))\cdot \text{Ш}(Gk_x,Gk_y)\}\end{aligned} \tag{3.13}$$

By applying the properties of the Fourier Transform, Equation 3.13 becomes:

$$\hat{m}[x,y] = ((m(x,y)*PSF(x,y))\cdot c(x,y))*(G\cdot \text{Ш}(\frac{x}{G},\frac{y}{G})) \tag{3.14}$$

where PSF(x,y) is the sampling Point Spread Function and is defined as:

$$PSF(x,y) = \mathscr{F}^{-1}\{S(k_x,k_y)\} \tag{3.15}$$

The PSF defines the extent of aliasing introduced by sampling.

3.2.5 Deapodization

As mentioned above, the function used as an interpolation kernel cannot be an infinite *Sinc*. As a consequence, its Inverse Fourier Transform is not a a perfect low-pass filter. Figure 9 shows an example for the 2-dimensional case: on top, a perfect low-pass filter, and its Fourier Transform; on the bottom, a Kaiser-Bessel interpolation kernel and its Fourier Transform. It can be observed that the function at the bottom attenuates the image at its edges, hence introducing unwanted distortion. This phenomenon is referred to as Apodization. This last stage aims at reducing the artifacts introduced by the non ideal interpolation kernels, and therefore takes the

name of Deapodization. From Equation 3.14, it can be noticed that the factor contributing to the Apodization is $(m(x,y) * PSF(x,y)) \cdot c(x,y)$. A possible way of canceling its effect is by dividing the image by $c(x,y)$, the Inverse Fourier Transform of the convolution kernel. The values for $c(x,y)$ can be computed both numerically, or through the analytical expression for the most common interpolation kernel functions. It is worth noticing that, beside the apodization artifacts, the edges of the FOV are also subject to aliasing noise. Deapodizing in presence of high aliasing noise can result in an amplification of that noise with a consequent deterioration of the final reconstructed image. To solve this issue, a partial deapodization can be performed, where the division by $c(x,y)$ is computed only for values at the center of the FOV. The latter is reasonable since the elements placed in the middle of the scanned image are usually more important than those at the edges. Consider the cases in Figure 10. On top, the low oversampling factor results in higher aliasing noise at the edge of the FOV. As a consequence, complete deapodization deteriorates the reconstructed image. On the bottom, the higher oversampling factor ensures that the artifact introduced by the aliasing noise is small compared to the apodization cut-off, leading to an image accuracy improvement when total deapodization is performed.

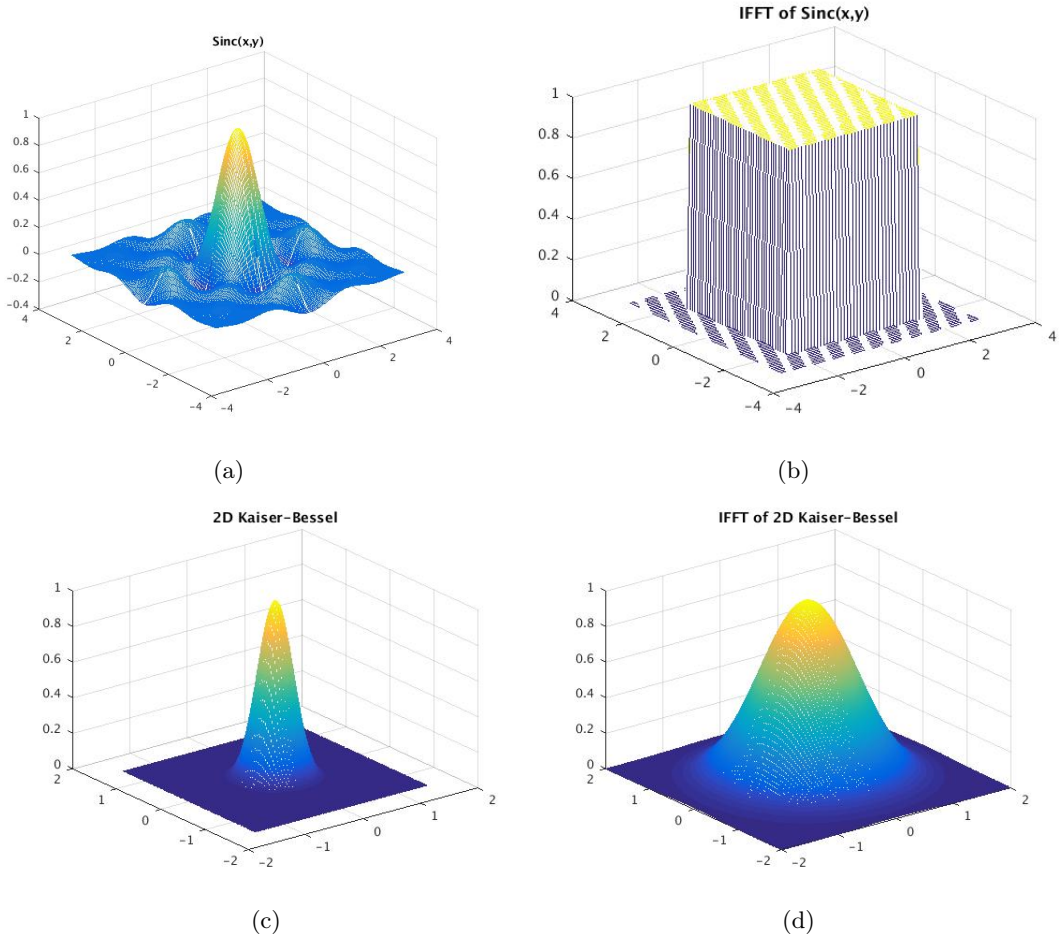
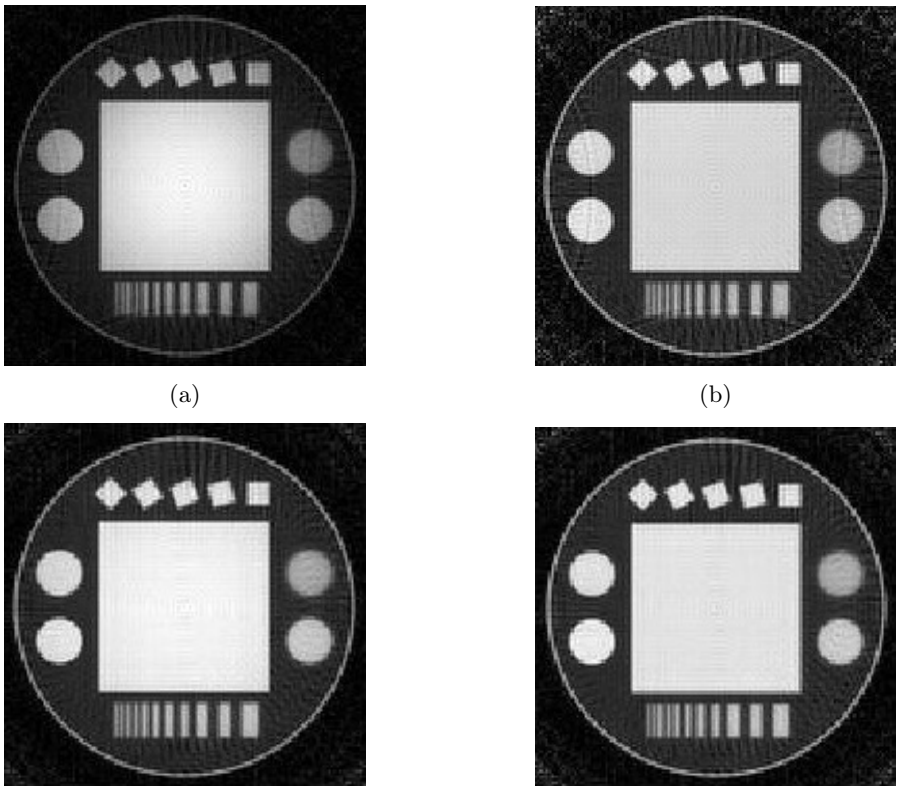


Figure 9: (a) 2D Sinc (b) Perfect 2D low-pass, IFFT of 2D Sinc (c) 2D Kaiser-Bessel (d) IFFT of 2D Kaiser-Bessel.



(a) (b)

(c) (d)

Figure 10: (a) Low oversampling factor, without Deapodization (b) Low oversampling factor, with complete Deapodization (c) High oversampling factor, without Deapodization (b) High oversampling factor, with complete Deapodization.

CHAPTER 4

FPGA IMPLEMENTATION

The theory and the algorithmic approaches described in the previous Sections provide a good basis for the final FPGA implementation. The used design language - OpenCL - and the characteristics of the target hardware impose constraints on the actual implementation, that must be well studied and understood in order to provide an effective solution.

4.1 FPGA Characteristics

The target hardware for this research is an Altera Arria 10 GX FPGA Development Kit. The kit is technologically advanced, and one of the cutting edge products in Altera's catalogue, providing with an high performance FPGA board, fast DDR4 on-chip memory, reduced power consumption with respect to previous generation boards, and high bandwidth connectivity over PCI Express.

I am really grateful to Altera for the possibility of working on such a Tool.

4.2 OpenCL

Open Computing Language (OpenCL) is a royalty-free framework developed to allow programmers to write code able to execute on heterogeneous hardware platforms. When it was conceived in 2008, the objective of the designers at Khronos Group was that of providing a tool to write high level code that ensured portability between diverse physical supports (CPUs, GPGPUs, DSPs, FPGAs), while allowing programmers to easily exploit the parallelism intrinsic

to the underlying hardware. The OpenCL standard defines both a programming language and a system architecture methodology. The language structure is built upon ANSI C (C99), and has been enriched with all the constructs necessary to describe parallelism.

Applications developed in OpenCL work as a combination of a processor running software - commonly referred to as the host machine - and one or more hardware accelerators. Whenever a computationally expensive operation is to be performed, the host machine delegates it to a hardware accelerator, where the operation has been described as an OpenCL *kernel*. In OpenCL, a kernel is the basic unit of code describing an execution flow. Kernels are executed by work-items, the smallest independent unit of execution in OpenCL, and work-items are logically clustered into work groups; code is then compiled so to exploit the available hardware execution units of the underlying device - *processing elements*, in the OpenCL abstraction - to achieve as much parallelism as possible.

Data transfers between host machine and accelerators happen by means of fast interconnection channels based on protocols like PCI Express. Bindings between host and accelerator are not automatic, and the host code needs to explicitly state read and write operations on the shared buffers. Despite the interconnection being fast, these operations are still a bottleneck when compared to the accelerator hardware and therefore have to be strategically planned.

4.2.1 Memory Model

The OpenCL standard promotes a model for memory, and the memory available in the devices is abstracted to match the following hierarchy:

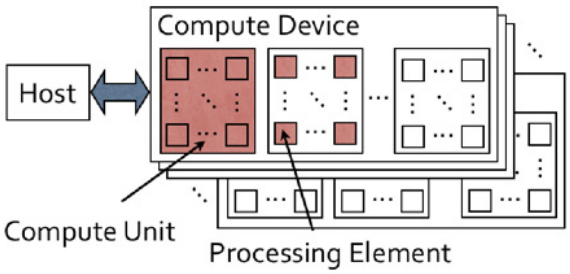


Figure 11: Example of processing element in the OpenCL model; image courtesy of AMD [8].

- Global memory: address space that can be accessed both by the host machine and the accelerator. Usually large in size but with high latency.
- Constant memory: read-only memory set by the host machine. Usually in the same physical location as the global memory, hence sharing its features in latency and size.
- Local memory: memory shared between work-items in a work group. Faster than global memory, reduced in size.
- Private memory: memory local to each work-item. It guarantees the lowest latency, but is extremely reduced in size.

When designing in OpenCL, proper care has to be taken in allocating variables into the most convenient memory level for their purpose. Failure in doing this can result in poor bandwidth usage and load/store dependencies, introducing relevant performance bottlenecks.

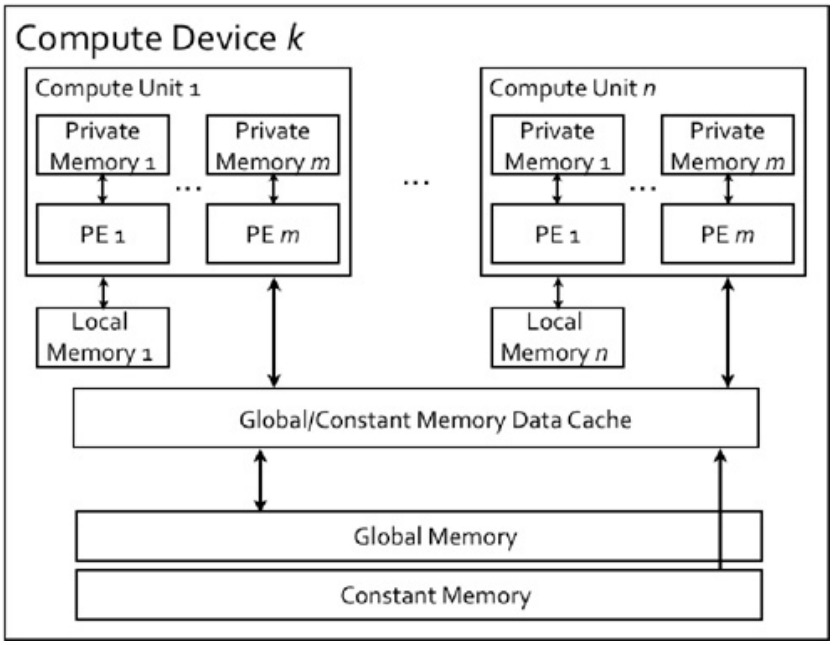


Figure 12: OpenCL memory model; image is a courtesy of AMD [8].

4.2.2 Altera OpenCL for FPGA

The need for increasingly faster computational engines led to the recognition of the potential of parallel processing. FPGAs, thanks to their reconfigurable nature, provide a very flexible tool to perform diverse types of tasks while exploiting pure hardware parallelism. However, the flow for FPGA programming requires the translation of the algorithm to be implemented into cycle-specific hardware descriptions, with data-paths, glue logic, state machines, and interconnections definition, using a low level HDL (Hardware Description Language) like VHDL or Verilog. Attempts to develop high-level languages that translate into HDL have already been made, and surely succeed in their goal of synthesizing working hardware. Unfortunately, their limitation

causes the difficulty found when extracting parallelism from source code that effectively results in high performing parallel hardware. OpenCL overcomes this limitation by letting the programmer explicitly specify the desired parallelism, and by including the concept of parallelism as a feature of the coding language itself.

4.2.2.1 Parallelism in OpenCL for FPGA

In OpenCL for FPGA, every kernel included in the code is synthesized as its own hardware component. Consequently, multiple kernels can be running concurrently. Parallelism inside a kernel is achieved through two techniques:

Complete unrolling Part of the code written as sequential statements can be synthesized as parallel hardware that executes in a small number of clock cycles. By default, complete unrolling underlies sequential statements in non cyclic constructs, but can also be manually specified (and successfully achieved) for loops of reasonable size through the *pragma unroll* N compiler directive. N specifies the desired degree of parallelism to be pursued. Higher N results in fewer clock cycles, at the expense of hardware utilization. Hence, the usage of *pragma* unrolling has to be monitored and well-planned.

Pipelining For computationally expensive loops (and possibly their sub-loops), the compiler tries to infer a comprehensive pipeline. Thanks to pipelining, iterations can be initiated in successive clock cycles, generating a considerable gain in execution latency.

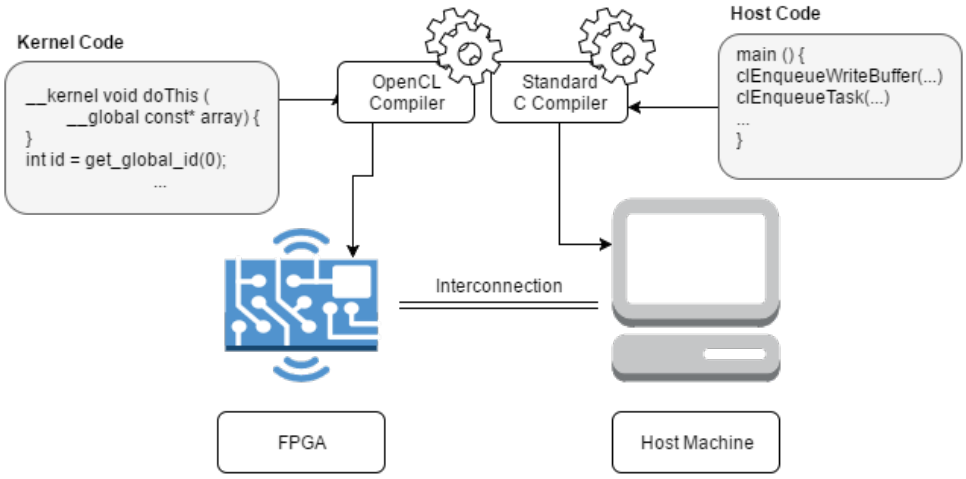


Figure 13: High level block description of the system.

4.2.3 Altera OpenCL

Altera (now part of Intel) has lately been focusing on promoting such an application for OpenCL, and presented the *Altera SDK for OpenCL*.

Beside implementing all of the features required to conform to the Khronos Group OpenCL standard, Altera introduced some additional features. One of the most relevant is *channels*, providing high efficiency inter-kernel communication infrastructures. E. Pezzotti in [1] shows how proper channel utilization can positively impact performance. Moreover, the Altera compiler provides the programmer with detailed reports about the compilation stage, including insights on data dependencies, pipeline stages latency, and forced sequential executions. Careful analysis of the report was key in gaining expertise on proper FPGA-aimed OpenCL programming, and has been part of the whole design process. In addition to that, Altera provides the user with

a framework to emulate the design by software, an essential resource to test the correct logical execution of the code. Finally, the design suite comes with a useful set of post-synthesis profiling tools: timing performance, clock frequencies and bandwidth usage are visually displayed, thereby helping the programmer better identify the features of the synthesized hardware.

For this work, Altera SDK Version 15.0 has been used. However, Altera has recently released the 16.0 version, which is supposed to enhance the existing toolset.

4.2.3.1 Altera OpenCL SDK Best Practices

In [9] and [10], Altera defines a set of good practices to be followed to achieve maximum performance in different applications. For this application, the following set of guidelines applied and were hence complied to as much as possible during the design phase:

1. Prefer single work-items kernel to NDRange kernels when the application is organized in multiple kernels with channels used for inter-kernel communications, and "you cannot break down the algorithm into separate work-items easily because of data dependencies that arise when multiple work items are in flight" [9].
2. Loops should have a number of iterations known at compile time.
3. Simpler code translates into more efficient hardware.
4. The number of accesses to global memory should be limited, and performed in bursts when unavoidable.

4.3 Architecture Overview

4.3.1 First Attempts

In the early stages of development, the research was leaning towards a direct OpenCL translation of the architecture described by U. Cheema et al. in [4]. The latter was implemented in Verilog, and hence at a very low level of abstraction. To translate it to OpenCL, nearly every Verilog component was coded as a separate kernel. Communication between kernels was achieved through both synchronous and asynchronous channels. Many months of those invested in the research have been spent implementing and optimizing this architecture from an emulation and report-based standpoint. Unfortunately, our research group was still missing both the tools necessary to actually synthesize and test the hardware (both the host machine and the FPGA). Once it was possible to test it, the limitations of this architecture in OpenCL became evident: the performance of 1 frame processed per second was achieved.

The failure in the aforementioned approach led us to an analysis of its causes and consequently a better understanding of OpenCL principles.

4.3.2 Adopted Architecture

The architecture that has been adopted as a final design takes into account all of the limitations listed in 4.2.3, while applying suitable techniques to implement the algorithm described in the previous sections. Three main goals have been pursued during the development phase:

1. **Scalability**: the solution had to be adaptable to work with images of increasing size. It must be able to efficiently scale to sizes commonly used for MRI images - usually falling in the range $[128^2, 512^2]$ -, while being able to adapt to higher pixel resolutions.

2. **Speed**: the solution must have competitive performance in terms of throughput, expressed as the number of frames processed per second.
3. **Accuracy**: the solution must be oriented towards practical results, guaranteeing high reconstruction accuracy in terms of visive perception.

Figure 13 provides an overview of the system as a whole. On the right, the host machine runs software to interface with the FPGA. The FPGA has been programmed to run the necessary hardware functions. The two devices communicate via PCI Express. The host machine has control over the FPGA; it decides which OpenCL kernels shall be run, and provides them with the proper arguments. Kernel arguments are passed over the PCIe and stored in the FPGA global memory. When the custom FPGA hardware has finished its computations, the results are copied back to the host via the same interconnection. It is easy to see that communication between the host machine and the FPGA is a potential bottleneck in the computation. Hence, the implementation tries to limit this kind of interactions as much as possible.

In the following chapters, the details of the FPGA hardware are described, followed by an in-depth on the code running on the host machine. The whole design process has been carried through holding the following assumptions true:

Assumption 1. *Given a specific MRI scanner, with a constant set of acquisition parameters, k-space is sampled on a fixed trajectory.*

Assumption 1 comes with a lot of implications, some of which allow for simplifications in the algorithm, and will be covered later in the text.

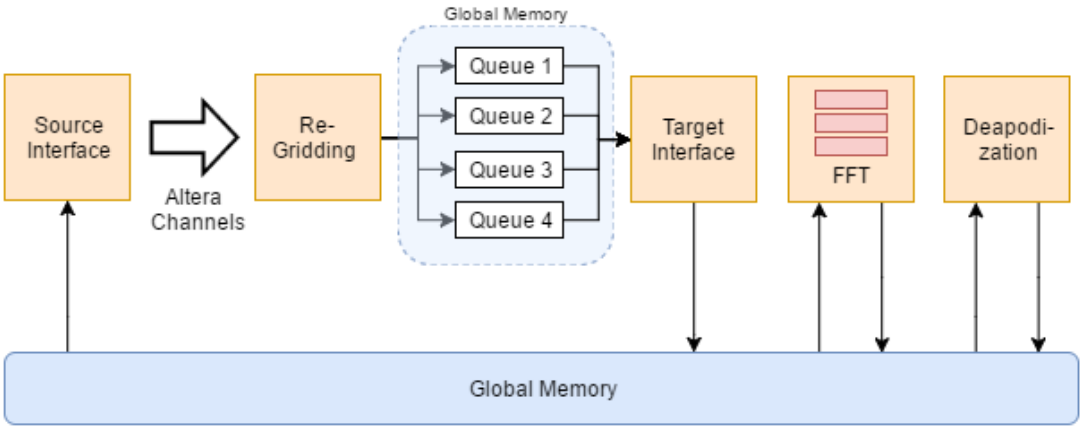


Figure 14: Block architecture for the FPGA implementation.

4.4 FPGA Implementation

Figure 14 shows a high level block diagram describing the architecture used for the system. Every orange box corresponds to a single OpenCL kernel. The following paragraphs describe how the theoretical algorithm has been translated to hardware, and the tasks carried through by each of the depicted kernels.

The main idea underlying the architecture is that of building a scalable system by overcoming the memory limitation of FPGAs with a tiling approach. Rather than proceeding with Re-Gridding on the whole matrix at the same time, a set of areas - referred to as tiles - are defined. Whenever possible and convenient, operations are performed on single tiles, thus limiting the amount of memory resources needed, and making it possible for the same architecture, on the same FPGA, to reconstruct images of arbitrary size with limited loss in performance.

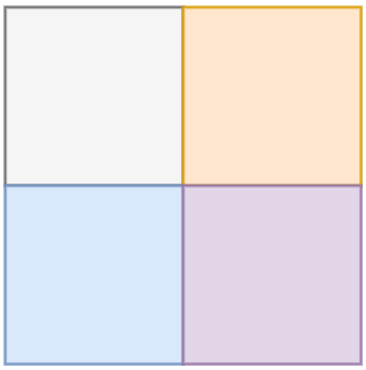


Figure 15: Tile division for $L = 4$.

K-space is thus divided into a set of L tiles. There are no constraints on neither the size of each tile nor the total number of tiles. For the purpose of processing images up to 512^2 pixel size, we have found the division in $L = 1$ (no division, for images of size 128^2 and 256^2) and $L = 4$ to be sufficient. Figure 15 shows the chosen division for $L = 4$.

This type of division has been adopted for two main reasons. First of all, the regularity of the tiling guarantees ease in matrix indexing operations, thus limiting the amount of hardware resources. Further more, assuming a nearly symmetrical sampling trajectory, each tile contains approximately the same number of source points, and this contributes in building more symmetrical and balanced hardware.

4.5 Source Interface Kernel

The Source Interface kernel is, for the current implementation, fairly simple. Its role is that of interfacing the remaining modules with the FPGA global memory: the sample points are passed to this kernel as an argument, and the kernel loops across all the sample points

- whose number is supposed to be known at compilation time. Each sample point is passed to the next kernel through an Altera OpenCL channel. In case Density Compensation with pre-compensation, one of the techniques studied in [1], is being implemented, it can take place during this stage, with no complications in terms of performance and pipeline unrolling. Any other possible operation on the source points can safely be performed in this kernel.

If no operation is being computed on the source points during this stage, a possible performance improvement might be obtained by eliminating this stage, and by programming the Re-Gridding kernel to interface directly with global memory. Table I shows a comparison in resource utilization and frames per second for two architectures working on 128^2 images both with and without the Source Interface kernel. As it can be noticed, there is no difference in resource utilization, and the fps favors slightly the architecture that includes the Source Interface kernel. Hence, the latter kernel has been kept in the final implementation.

TABLE I: COMPARISON IN PERFORMANCE AND RESOURCE UTILIZATION FOR THE ARCHITECTURE WITH AND WITHOUT THE SOURCE INTERFACE KERNEL.

	128	128 (no S.I.)
Logic Utilization	55	55
Dedicated Logic Registers	33	33
Memory Blocks	36	36
DSP Blocks	58	58
fps	234.7647	233.1229

4.6 Re-Gridding Kernel

The main task executed by the Re-Gridding kernel is that of computing the contribution generated by each sample points. It is also in charge of implementing the tile mechanism by means of virtual queues.

The kernel receives the sample points sent by the Source Interface kernel over an Altera OpenCL channel. For each received source point s_i, the kernel computes the set of grid target points that fall in its convolution window. For each target point tp_j, the contribution d generated by s_i to tp_j is computed as:

$$d(s_i, tp_j) = s_i \cdot \Phi(s_i, tp_j) \tag{4.1}$$

where $\Phi(s_i, tp_j)$ is the interpolation kernel value computed by knowing the k-space coordinates of s_i and tp_j. Sections 4.6.1 and 4.6.2 cover in depth how to calculate $\Phi(s_i, tp_j)$.

In order to process every tile separately, there is the need for a mechanism to distinguish and store contributions $d(s_i, tp_j)$ based upon the tile of their corresponding target point tp_j. To achieve that, a set of L queues is used, implemented as vectors in global memory. Each queue k has the function of storing, for tile k, the set of all tuples $d(s_i, tp_j), tp_{j,x}, tp_{j,y})$ generated by the Re-Gridding kernel. To estimate the size of the queues, Assumption 1 can be exploited. Given a known trajectory, estimating the number of contribution per queue is fairly straightforward.

The algorithmic complexity of this step is $O(NW^2)$. The pipeline has been inferred so that every source point is processed in 2 clock cycles. Hence, the actual time complexity for the Re-Gridding kernel is $O(N)$.

4.6.1 On-the-fly Kernel Computation

One way of computing the kernel is by actually implementing on the FPGA hardware the function describing the kernel analytic expression.

Given a coefficient $\Phi(s_i, tp_j)$, it can be evaluated as:

$$\Phi(s_i, tp_j) = C(|s_{i,x} - tp_{j,x}|) \cdot C(|s_{i,y} - tp_{j,y}|) \tag{4.2}$$

where $C(x)$ is the analytic expression for the 1-dimensional interpolation function, that has been rearranged to be centered in $x = 0$ and with support around $[-\sigma, \sigma]$. The advantage of this approach is in the precision obtained in the evaluation of the kernel coefficients. The main disadvantage is in the introduction of further hardware components, with the risk of a overall pipeline slowdown in case of particular hard to compute kernels.

4.6.2 Static Kernel Computation

A second possible approaches resorts to Look-Up Tables. A set of S uniformly sampled kernel values are computed off-line and stored statically in an array Knl on the FPGA, such that $Knl[0] = C(0)$ and $Knl[S-1] = C(\sigma)$. Given a coefficient $\Phi(s_i, tp_j)$, it can be evaluated as follows:

- Compute the absolute value of the distance between s_i and tp_j on both the x and y direction. The distance is then scaled by S:

$$\begin{aligned} dist_x &= \frac{|s_{i,x} - tp_{j,x}|}{\sigma} \cdot S \\ dist_y &= \frac{|s_{i,y} - tp_{j,y}|}{\sigma} \cdot S \end{aligned} \tag{4.3}$$

- Approximate the value of the kernel at $distx$ and $disty$ by using nearest-neighbor or linear interpolation. For nearest-neighbor interpolation, it is sufficient to round the value for the distance to the closest integer:

$$\hat{C}(dist_x) = Knl[round(dist_x)] \tag{4.4}$$

For linear interpolation, consider the generic equation for a line:

$$y = mx + q \tag{4.5}$$

m can be computed as:

$$m = \frac{\Delta y}{\Delta x} = \frac{\Delta y}{1} = Knl(\lceil dist_x \rceil) - Knl(\lfloor dist_x \rfloor) \tag{4.6}$$

While q can be computed as:

$$q = y - mx = Knl[\lceil dist_x \rceil] - (Knl(\lceil dist_x \rceil) - Knl(\lfloor dist_x \rfloor)) \cdot \lceil dist_x \rceil \tag{4.7}$$

Hence, the desired approximation of the kernel function for the point $dist_x$ is:

$$\hat{C}(dist_x) = m \cdot dist_x + q \tag{4.8}$$

This approach guarantees low memory overhead, since it only requires a vector of S elements, plus some more logic for the interpolation computation. The drawback is, however, a small loss in precision when evaluating the kernel coefficients.

When measuring the reconstruction SSIM with different kernel computation techniques, the difference between the obtained values is in the order of 10^{-3}, also for small values of S. Moreover, for $S > 100$, no further increase in SSIM could be detected. Hence, it is worth analyzing the impact of both solutions on hardware resources utilization. Table II shows the FPGA resources needed for the three cases, given a fixed set of reconstruction parameters. The kernel function used for extracting these data is a Gaussian curve. It can be noticed that the on-the-fly computation offers the lowest resource utilization, at least for the kernel under analysis.

Also in this case, the difference is minimal. Consequently, the simpler option has been chosen for the final implementation, that is the nearest neighbor interpolation technique with $S = 100$.

TABLE II: FPGA RESOURCES UTILIZATION FOR THE THREE CASES.

	Linear	Nearest-neighbor	On-the-fly
Logic Utilization	48	55	42
Dedicated Logic Registers	28	33	25
Memory Blocks	36	36	36
DSP Blocks	51	58	47

4.7 Target Interface Kernel

The main task of this kernel is that of combining the contributions generated by the Re-Gridding kernel and obtain the final values for each target grid point.

The kernel processes one queue at a time. While processing queue i, a copy of the portion of the target space is stored in local memory - the fastest available. Contributions are read from the queue and used to update the value of the corresponding target point. When all the contributions in a queue have been processed, the local copy of the target tile space is copied back to global memory. This series of operations is repeated for all queues.

The algorithmic complexity of this stage is $O(NW^2 + G^2)$. Unfortunately, the compiler is not able to fully infer the pipeline, because of data dependency between successive contributions. Indeed, the compiler has to process consecutive contributions that might affect the same target point. If this situation is not properly handled, data stored in the target points can end up being corrupted. Several attempts have been carried out to try and eliminate this dependency, but to no avail. However, the pipeline stage duration has been reduced to 6 clock cycles, which means that race conditions on target points can happen only if any group of 6 consecutive

Figure 16: On top: sequence of contributions that generates race conditions. On bottom: sequence of contributions that does not generate any race conditions.

contributions influences the same target point. Figure 16 shows an example with both a safe and an unsafe set of contributions. OpenCL offers a compiler directive, *ipdev*, which can be used to specify that no data dependency occurs on a given variable. Therefore, if we can ensure that no invalid sequence of contributions is generated, it it possible to force the compiler to fully unroll the pipeline. Given Assumption 1, there are techniques to guarantee the latter condition, and 6.5.1 specifies the one that has been adopted in our implementation. Consequently, the *ipdev* directive could be exploited and the pipeline for the kernel fully unrolled.

The output of this kernel is the fully reconstructed matrix of grid points in k-space, stored in global memory. The same global memory location can be accessed by the next set of kernels.

4.8 Inverse Fast Fourier Transform Kernels

The hardware block performing the Inverse Fast Fourier Transform operation has been adapted from the example code for the 2D-FFT available on Altera's website, at [11]. In terms of implementation, a two dimensional IFFT can be executed as two separate one dimensional IFFT, one operating on the rows of the matrix as isolated vectors, and one operating on the columns. Altera's code provides us with three kernels, which have been designed to be highly customizable in terms of parallelism and hardware resources used:

1. Fetch kernel: it reads the matrix values from global memory, and sends them via Altera OpenCL channels to the second kernel.
2. IFFT kernel: it receives the matrix values via Altera OpenCL channels, and computes the IFFT on the matrix rows. The results are then sent to the next kernel via Altera OpenCL channels.
3. Transpose kernel: it receives the matrix values via Altera OpenCL channels, transposes the matrix, and stores it into global memory.

To perform a complete two dimensional IFFT, the three kernels above need to be run twice in a row. During the first execution, the Fetch kernels should be given the re-gridded matrix as an input. During the second execution, the output of the Transpose kernel for the first iteration should be fed as an input to the Fetch kernel. The output of the second execution of the Transpose kernel is the transformed matrix - the reconstructed image - stored in global memory.

4.9 Deapodization Kernel

The Deapodization operation consists of the normalization of the reconstructed image by a series of factors calculated from the interpolation kernel. Given the relation:

$$C(k_x, k_y) \xrightarrow{\mathscr{F}^{-1}} c(x, y) \tag{4.9}$$

then the Deapodization coefficient $c(x, y)$ can be computed as:

$$c(x, y) = c(x) \cdot c(y) \tag{4.10}$$

where

$$C(k_x) \xrightarrow{\mathscr{F}^{-1}} c(x) \tag{4.11}$$

and $C(k_x)$ is the one-dimensional version of the interpolation kernel. As for the kernel coefficients, also the Deapodization factors can be computed both On-the-Fly and Statically.

4.9.1 On-the-fly Deapodization Coefficients Computation

For the On-the-Fly computation, the analytic expression for the Inverse Fourier Transform of the interpolation kernel can be used, if known. For the Kaiser-Bessel function, it is [12]:

$$c(x) = \frac{sin\sqrt{(\pi W x/G)^2 - \beta^2}}{\sqrt{(\pi W x/G)^2}} \tag{4.12}$$

where β is one of the tuning parameters for the Kaiser-Bessel curve.

4.9.2 Static Deapodization Coefficients Computation

If the analytic expression of the Inverse Fourier Transform of the kernel is not known, or if we want to limit the amount of hardware resources used for computation, the Deapodization coefficients can be obtained by computing off-line the IFFT of the kernel, and by storing them into an array of size N on the FPGA. More precisely,according to the guidelines provided by P. Beatty et al. in [12], the coefficients can be computed as follows:

1. Zero pad the kernel to a vector of size SG.
2. Compute the IFFT of the vector.
3. Create a vector of size SG to host the interpolation function for nearest-neighbor or linear interpolation. For the former, $h(x) = Sinc(SG)$, for the latter, $h(x) = Sinc^2(SG)$, and compute its IFFT.
4. Multiply the two vectors together, and extract the product vector from index $\frac{SG}{2} - \frac{N}{2}$ to index $\frac{SG}{2} + \frac{N}{2} - 1$.
5. Generate a $N \times N$ matrix by multiplying the vector obtained in point 4. with itself.

Both the approaches provide Deapodization values that yield reconstructions with comparable accuracy, and the best choice depends on two main factors: the amount of resources used, and the availability of an analytical expression for the Inverse Fourier Transform of the kernel. Table III shows the variation in resources utilization for the two methods. As it can be seen the computation on-the-fly requires significantly higher logic utilization.

TABLE III: VARIATION IN FPGA RESOURCE UTILIZATION FOR THE TWO DEAPODIZATION COEFFICIENTS COMPUTATION TECHNIQUES.

	Static	On-the-fly
Logic Utilization	30	55
Dedicated Logic Registers	18	33
Memory Blocks	33	36
DSP Blocks	29	58

CHAPTER 5

CHOICE OF INTERPOLATION KERNEL FUNCTION

In this section, the process that led to the choice of the interpolation kernel function and to the tuning of the algorithm parameters is discussed, together with the obtained results. In [13], O'Sullivan shows that the ideal interpolation function is an infinite *Sinc*, which however requires an excessive computational burden. This results in the need for a windowed kernel. The windowing operation, however, introduces two main non idealities:

1. Sidelobes: beside the main central lobe, unwanted lobes are present on the sides, that contribute to amplify the aliasing noise.
2. Main lobe attenuation: the main lobe attenuates the image on the sides of the FOV.

Figure 17 shows the Fourier Transforms for an infinite *Sinc*, and for a windowed *Sinc*. While the main lobe attenuation can be corrected by deapodizing the reconstructed image, the impacts of the sidelobes on the aliasing noise is a fundamental criterion in the choice of the interpolation kernel. In [14], Jackson et al. explore different kernel possibilities, using the relative amount of aliased energy as the metric to be minimized, and show the Kaiser-Bessel function to be optimal in terms of sidelobes effects suppression and computational ease. In [12], Beatty et al. further extend the research in [13], and provide a set of optimal tuning criteria for the Kaiser-Bessel function, using a variation of Jackson's aliasing amplitude as an optimization criterion.

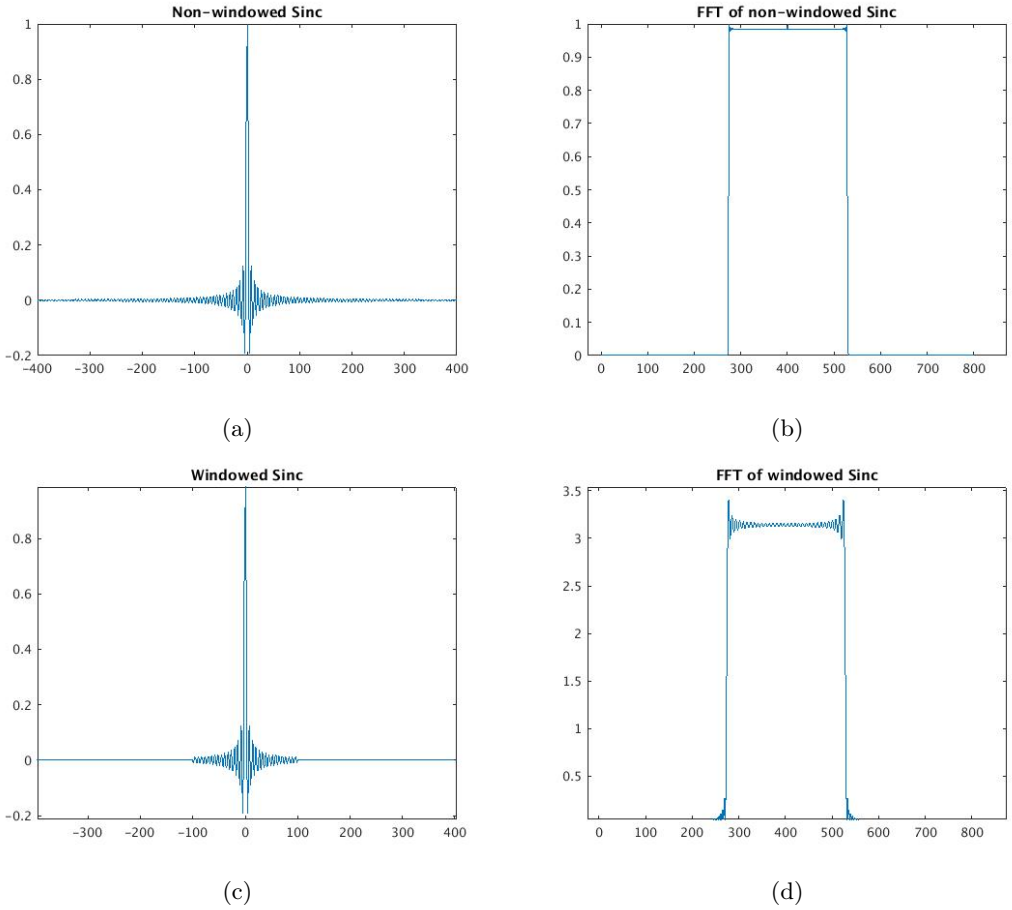


Figure 17: (a) Fourier Transform of an infinite *Sinc* (b) Fourier Transform of a windowed *Sinc*

In our work, different optimization metrics have been explored. Previous works focus mostly on metrics that consider the interpolation phase as a standalone. In this research, the results of the interpolation phase are strictly related to the successive steps and to the final image reconstruction accuracy. The objective of the whole process is, ultimately, recreating an image that is structurally and visually similar to the original, where the original is an ideal reconstruction obtained with more time consuming and accurate techniques. Consequently, the metrics used in this research take into account the final reconstructed image in its relation to a golden reference reconstruction.

5.1 PSNR and MSE

Two common indexes for image quality assessment are the Mean Squared Error (MSE):

$$MSE = \frac{1}{MN} \sum_{n=1}^{N} \sum_{m=1}^{M} [\hat{g}(n,m) - g(n,m)]^2 \tag{5.1}$$

and the Peak Signal-to-Noise Ratio:

$$PSNR = -log_{10} \frac{MSE}{S^2} \tag{5.2}$$

where $g(n,m)$ is the reconstructed image, $\hat{g}(x,y)$ is the reference image, and S is the maximum pixel value [15].

One of the limitations of the MSE is the dependency it has on the image intensity scaling, i.e. on the maximum value that can be assumed by a pixel. PSNR tries to overcome this limitation by normalizing with respect to the maximum pixel value.

In any case, both the indexes rely on a model of the reconstructed image as the sum between the ideal image and an error signal. However, two reconstructed images might have the same MSR/PSNR, yet include different type of errors with different visibility. The latter is due to the fact that these two indexes do not consider at all visual quality, and how the reconstructed image is perceived by human vision, in comparison to a reference standard. Hence comes the need for a metric that models perceptual quality and exploits knowledge of human vision characteristics to effectively estimate visual similarity.

5.2 The SSIM Index

The Structural Similarity Index Metrics (SSIM) has been theorized with "the assumption that the human visual system is highly adapted to extract structural information from the viewing field. It follows that a measure of structural information change can provide a good approximation to perceived image distortion" [16]. The SSIM index has been demonstrated to be a very suitable indicator for image similarity, and can be computed according to the following formula:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.3}$$

where:

1. μ_x, μ_y are the arithmetic means of respectively x and y.
2. σ_x, σ_y are the variances of respectively x and y.
3. σ_{xy} is the covariance of x and y.

4. c_1, c_2 are constants use to avoid instability where one of the factor in the denominator is close to 0. They are computed as:

$$\begin{aligned} c_1 &= k_1 L \\ c_2 &= k_2 L \end{aligned} \tag{5.4}$$

and $k_1 = 0.01, k_2 = 0.03$ by default, and L is the dynamic range of the pixel values [16].

5.3 Interpolation Parameters

When performing the interpolation through convolution, there are some parameters that can and have to be finely tuned in order to obtain a good balance between performance and accuracy.

5.3.1 Interpolation Window Size

As mentioned before, an actual interpolation kernel must have finite support for clear practical reasons. The extent of the kernel support defines a window of points around any sample points s_i that will be affected by s_i, and is referred to as interpolation window size.

Properly tuning the interpolation window size is of tremendous importance for the whole process. As the size increases, the number of grid points influenced by a single sample points increases as well, and consequently more arithmetic operations have to be performed per sample point. In addition, a too wide interpolation window results in much lower reconstruction accuracy, since each sample effects considerably also grid points farther away, that it is not supposed to have big impact on. A practical example of this can be observed in Figure 18, where the convolution has been performed on MATLAB with constant oversampling factor.

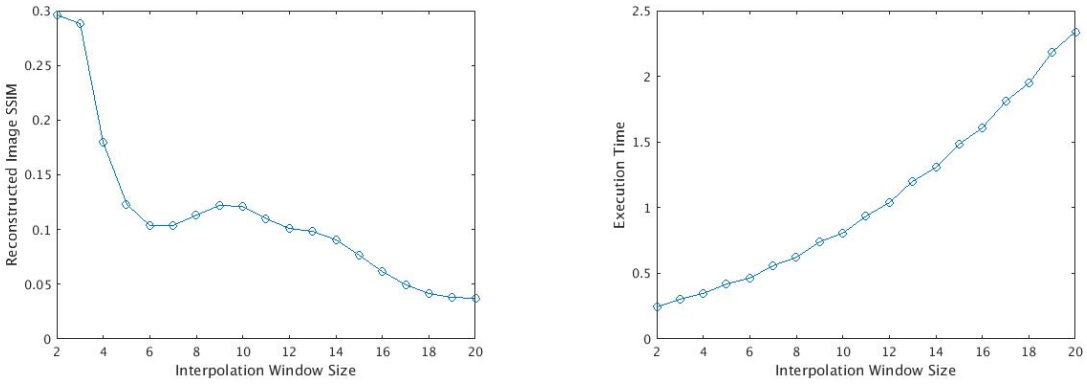


Figure 18: On the left: decrease in reconstruction accuracy with increasing interpolation window size. On the right: increase in processing time with increasing interpolation window size.

5.3.2 Kernel Computation

Another factor to take into account when choosing an interpolation kernel is the hardship in calculating its coefficients. For simplicity, it is desirable to have a function with the property of separability. A two-dimensional function $F(x, y)$ is such that:

$$F(x, y) = f(x) \cdot f(y) \tag{5.5}$$

where $f(z)$ is the one-dimensional version of the function. Given the separability property and the analytic expression for f, the kernel values can be obtained in two ways:

1. By evaluating the analytic expression for both $f(x)$ and $f(y)$, and then multiplying their values.

2. By pre-sampling off-line S kernel values for f, and approximating $f(x)$ and $f(y)$ with simple interpolation methods (nearest-neighbor, linear).

5.4 Possible Candidates

Several possible kernel functions have been studied in the literature, mainly by [14]. Figure 19 shows the normalized plots for the following functions [14]:

- Two term cosine, with equation:

$$\Phi_1(u) = \gamma + (1-\gamma) \cdot cos(\frac{2\pi}{W}u) \tag{5.6}$$

- Three term cosine, with equation:

$$\Phi_2(u) = \gamma + \delta \cdot cos(\frac{2\pi}{W}u) + (1-\gamma-\delta) \cdot cos(\frac{4\pi}{W}u) \tag{5.7}$$

- Gaussian curve, with equation:

$$\Phi_3(u) = exp[-\frac{1}{2}(\frac{u}{\sigma}^2)] \tag{5.8}$$

- Kaiser-Bessel bell, with equation:

$$\Phi_4(u) = \frac{1}{W} I_0 \beta \sqrt{1-(2u/W)^2} \tag{5.9}$$

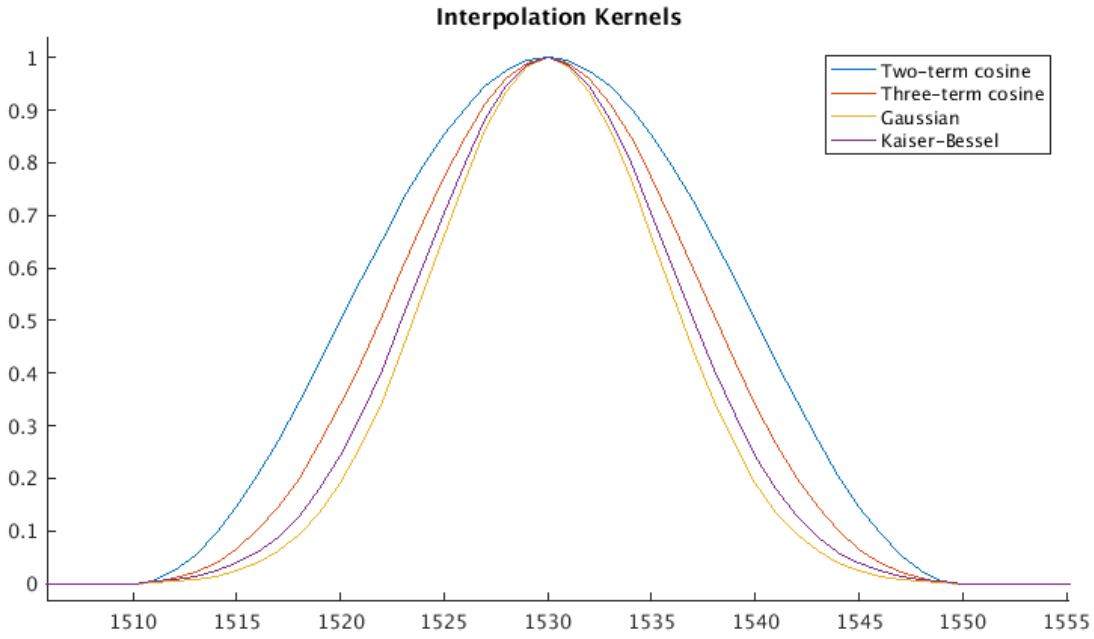
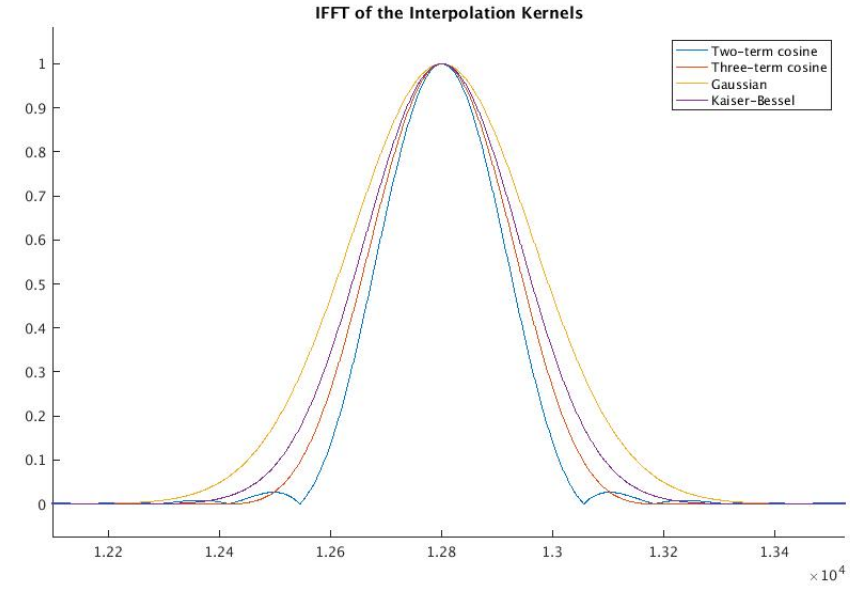


Figure 19: The most commonly studied interpolation kernel functions.

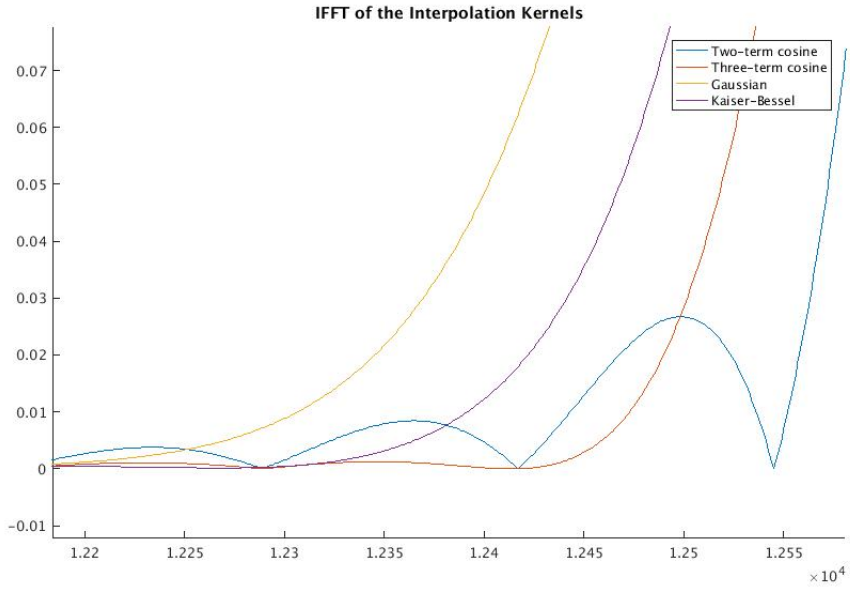
where the variable u is defined in the interval $[-\frac{W}{2}, \frac{W}{2}]$, and γ, δ, σ and β are design parameters. I_0 is the zero-order modified Bessel function of the first kind.

5.5 Optimization Criteria

For the practical application tackled in this work, reconstructing the acquired images with high accuracy is of primary importance. At the same time, performance is key both for recent imaging applications - fMRI and real-time MRI, to name a few - and standard MRI acquisitions. Hence, the optimal kernel should be able to provide optimal accuracy while keeping the execution time as low as possible.



(a)



(b)

Figure 20: (a) Fourier Transform of the four kernels in Figure 19. (b) Zoom-in on the side-lobes of the Fourier Transform of the four kernels in Figure 19.

The parameters that can be tuned and have an impact on performance are, for this architecture, the convolution window size W and the oversampling factor α. More specifically, in the Target Interface kernel, which is the bottleneck to performance, the execution time T is such that:

$$\begin{aligned} T &\propto L(\frac{\#_{sp}W^2}{L} + \frac{(\alpha N)^2}{L} + Q) \\ &\propto \#_{sp}W^2 + \alpha^2 N^2 + LQ \end{aligned} \tag{5.10}$$

where $\#_{sp}$ is the overall number of sample points available. The first contribution is due to the processing of individual target points contributions, while the second is due to the transfer of the local tile back to global memory. The last contribution, Q, is given by the clock cycles that are lost between the pipeline for tile i and that for tile $i+1$, and is neglectable with respect to the others. Let's try to quantify the impact of these terms on the performance of Target Interface.

In academics, the number of sample points is usually taken equal to N^2. In real life acquisitions, the number of source points usually largely exceeds N^2, so that we can safely state that $\#_{sp} \geq N^2$.

Moreover, the oversampling factor usually lies in the range $[1, 2]$, since $\alpha = 2$ has been proven to be more than enough to remove aliasing noise. At the same time, interpolation window values are meaningful for values greater or equal than 2, so that we can sagely say that $W \geq 2$. Additionally, writing back to memory can be performed in burst, so that the operation

is optimized and not much of a computational burden. All of the above highlights that the main contribution to the computational time is caused by the first term of the summation. Consequently, reducing the value for W would benefit performance more than minimizing α. It is worth mentioning that the latter has a great impact mostly on memory utilization, since a $\alpha N \times \alpha N$ matrix has to be stored in memory. Because of this, many related works have focused on reducing α at the expense of higher W values, in order for high memory requiring reconstruction applications - like 3D MRI - to be implementable. In our case, the tiling approach guarantees the solution to be scalable, hence reducing the importance of memory consumption: indeed, the size of the matrix that needs to be stored in the local FPGA buffer can be tuned depending on the tiling scheme.

5.6 Theoretical Results

To choose a convolution kernel among the functions listed in 5.4, the trends in their accuracy values have been identified for a fixed convolution size ($W = 2$, the minimum), while varying the oversampling factor α in the interval $[1, 2]$, and adjusting the tuning parameters.

To identify the interval in which the tuning parameters should swing, the following procedure has been followed:

- All of the kernels parameters were tuned so that the kernel values were approximately 0 for $|x| > W/2$.
- The parameters were coarsely varied in intervals with expanding limits, for varying α, and the maximum reconstruction SSIM was stored for that given interval. The limits expansion stopped when the accuracy for the reconstruction did not increase anymore.

Once the boundaries for the kernel parameters tuning were defined, the parameters were finely varied within the interval together with α.

As a starting reconstruction data set, a 128×128 pixels image provided by J. Pauly in [17] and common in academic research has been used. The image contains multiple sharp edges, introducing challenges in reconstruction and hence constituting a good target for the analysis of Re-Gridding accuracy. The number of k-space samples in the data set is equal to 12288.

The following Section shows the plot for the SSIM obtained by the different kernels when increasing the oversampling factor and the tuning parameters in their meaningful ranges.

5.6.1 Two-terms Cosine Kernel

Figure 21 shows the trend in reconstruction SSIM for increasing values of oversampling factor α and different γ. The interval of suitable γ swings between 0.4 and 0.55, reaching a peak accuracy around $\gamma = 0.5$. The SSIM follows an increasing logarithmic trend, with slow growth for higher values of α. The maximum reconstruction accuracy is around 0.95, and the delta between maximum and minimum accuracy obtained for a given γ is approximatively constant and close to 0.16.

5.6.2 Gaussian Kernel

Figure 22 shows the trend in reconstruction SSIM for increasing values of oversampling factor α and different σ. The interval of suitable σ swings between 0.20 and 0.55, reaching a peak accuracy around $\sigma = 0.35$. For values of σ at the extrema of the interval, the SSIM becomes unsatisfactory, although independent of α and nearly constant. As σ increases, accuracy starts growing logarithmically, reaching peak values below 0.9. For optimal values of σ, the increase in

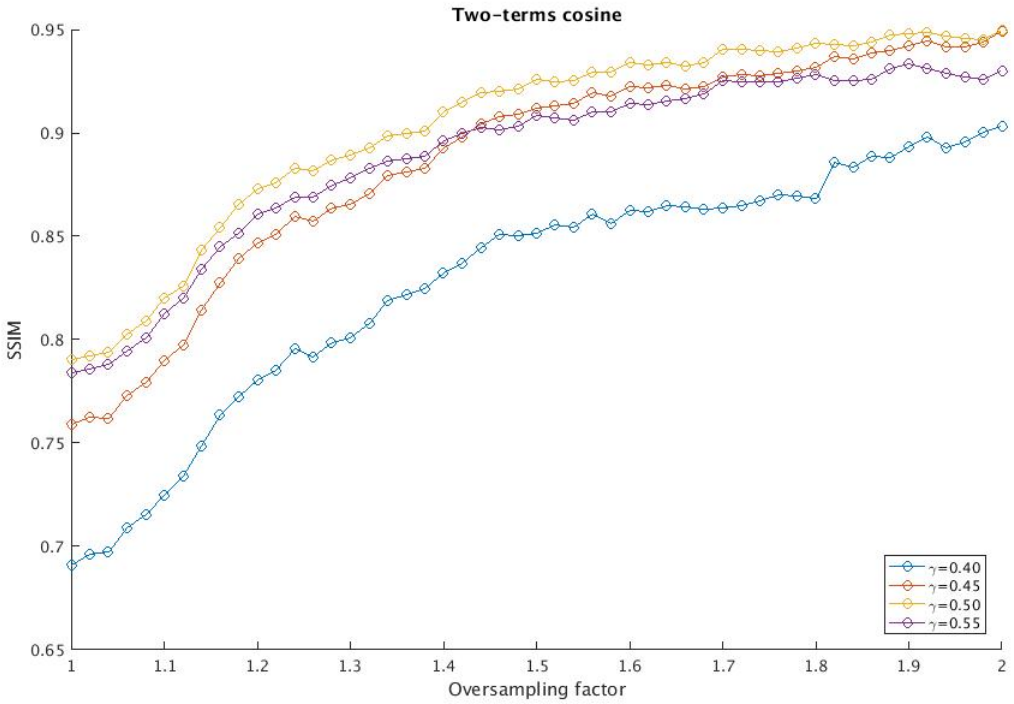


Figure 21: Reconstruction SSIM with varying oversampling factor for different values of the tuning parameter γ in a Two-Term Cosine kernel.

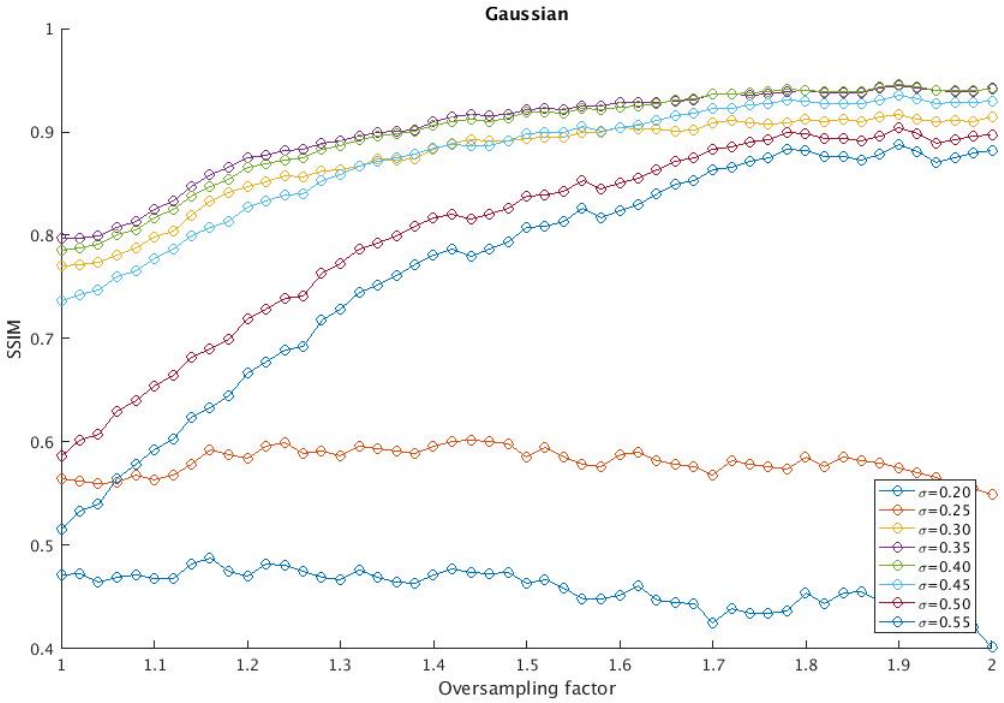


Figure 22: Reconstruction SSIM with varying oversampling factor for different values of the tuning parameter σ in a Gaussian kernel.

SSIM is still logarithmic but less steep, with slow growth for higher values of α. The maximum reconstruction accuracy is around 0.94, and the delta between maximum and minimum accuracy obtained for a given γ is approximatively constant for optimal σ and close to 0.14.

5.6.3 Kaiser-Bessel Kernel

Figure 23 shows the trend in reconstruction SSIM for increasing values of oversampling factor α and different β. The interval of suitable β swings between 10 and 50, reaching a peak accuracy between $\beta = 15$ and $\beta = 20$. The accuracy seems to be less sensitive to small variation

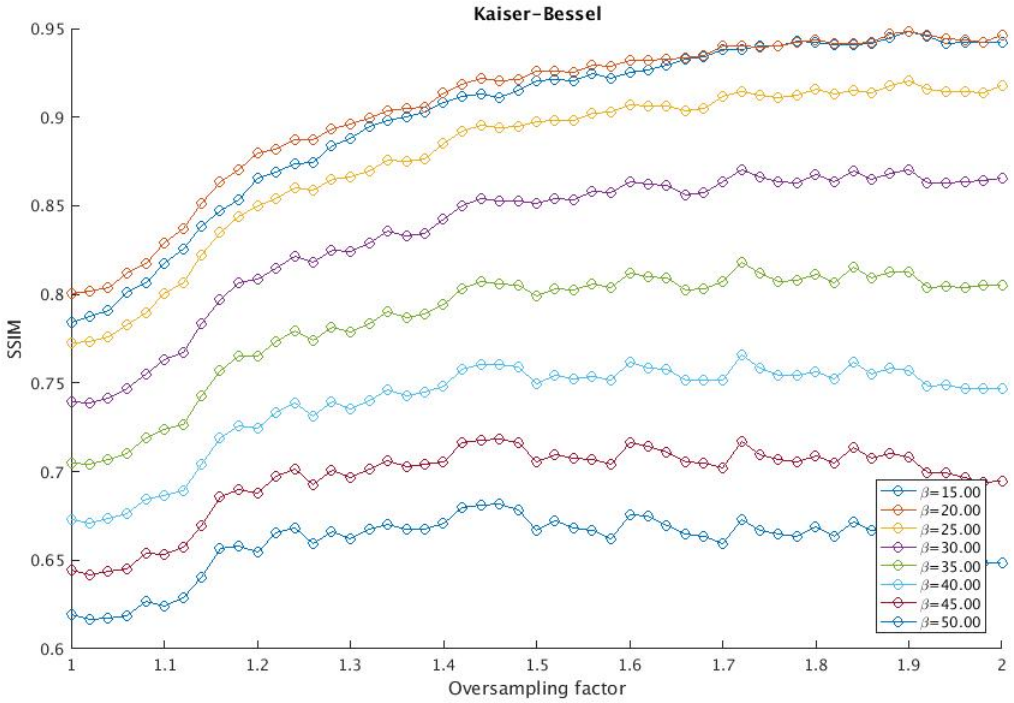


Figure 23: Reconstruction SSIM with varying oversampling factor for different values of the tuning parameter β in a Kaiser-Bessel Kernel.

in β. Moreover, despite the increase in β, the SSIM follows a logarithmic trend with slowly increasing steepness; this is possibly a good feature for cases when β is not perfectly tuned, since we can still expect the Kaiser-Bessel kernel to provide reasonable increase in accuracy as the oversampling factor grows. The maximum SSIM is around 0.95, and the highest delta between maximum and minimum accuracy obtained for a given β is approximatively 0.15.

5.7 Accuracy for Real MRI Images

The same study as in the previous Section has been performed on the dataset consisting of actual MRI cardiac acquisitions in [18].

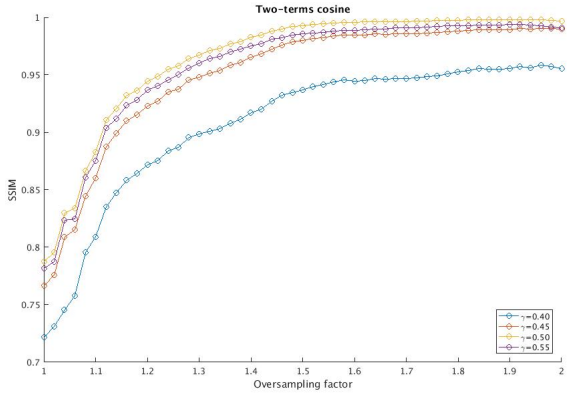
The trends are very similar to those obtained for the image from [17], and the maximum obtained SSIM is even higher, reaching values of 0.995. This is probably due to the higher number of sample points in k-space, which are 311808 for a 150×150 image.

5.7.1 Choice of the Optimal Kernel

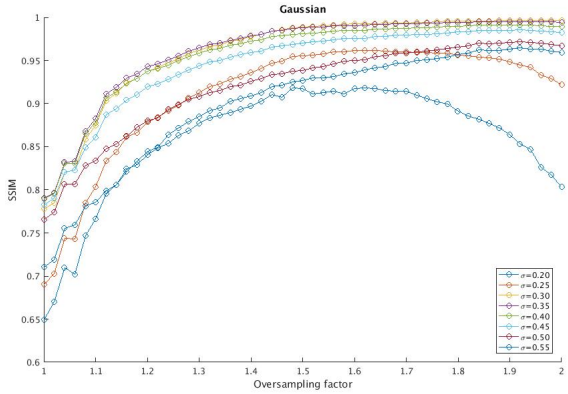
To help identify the optimal kernel parameters, the maximum SSIM achieved by each kernel per each α has been plotted in Figure 25. To choose a value for α, a minimum satisfactory SSIM had to be chosen. Judging from empirical reconstructions, an SSIM of 0.9 seems to be sufficient for optically semi-similar reconstructions. Figure 26 shows the ideal reconstruction against a reconstruction with $SSIM = 0.9$ for the academic image. As it can be observed, no difference is perceivable to the human eye.

While for the cardiac image there is no relevant difference in the choice of one kernel function over the other, for the academic image the Kaiser-Bessel yields better accuracy over the entire domain for α. This is also in line with the theoretical results brought up by [14] and [12]. A value of $\alpha = 1.32$ yield sufficiently big SSIM, and has been chosen for the optimal implementation.

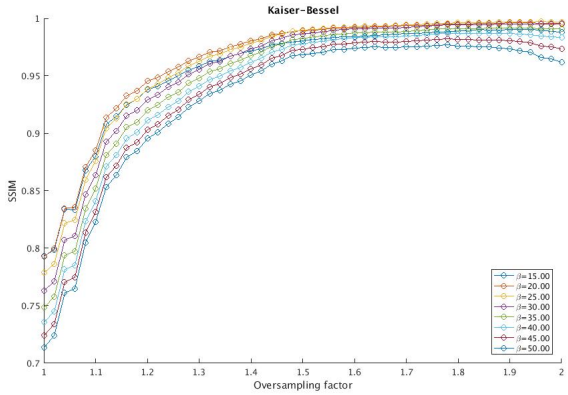
It is also worth noticing that the curves for the Two-Term Cosine and the Three-Term Cosine constitute an exact match. The latter can be explained by the degeneration of one of the functions into the other for properly tuned parameters.



(a)

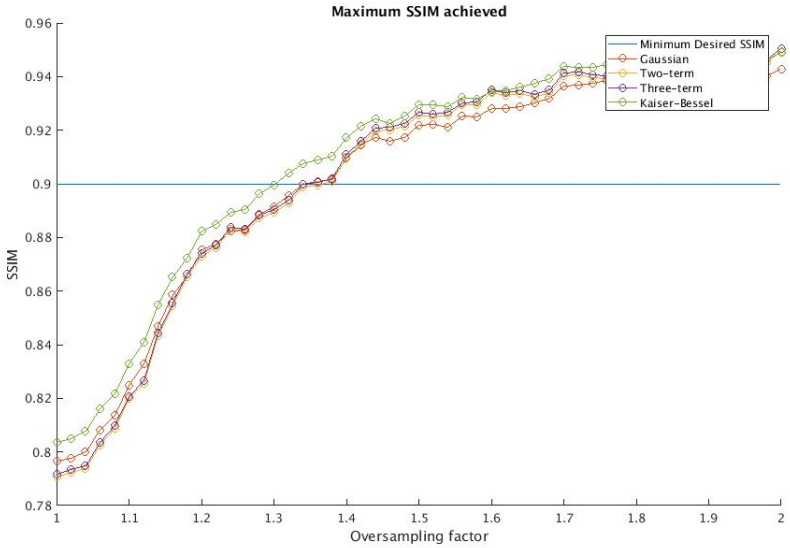


(b)

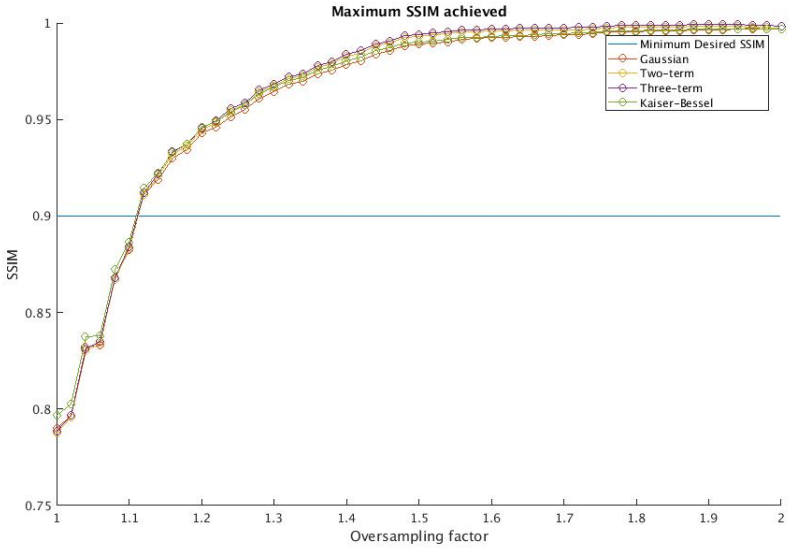


(c)

Figure 24: Trend in SSIM for increasing α for Cardiac images for different tuning parameters. (a) Two-terms Cosine kernel (b) Gaussian kernel (c) Kaiser-Bessel kernel.

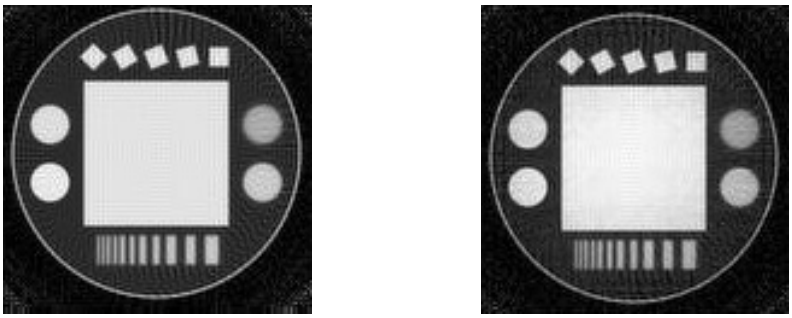


(a)



(b)

Figure 25: Maximum SSIM achievable by each kernel function for different α values on (a) the classical academics image (b) Cardiac acquisition image.



(a) (b)

Figure 26:]
(a) Ideal reconstruction (b) Reconstruction with $SSIM = 0.9$.

CHAPTER 6

HOST CODE

6.1 Host Code Tasks

The OpenCL kernels described above get synthesized by the Altera OpenCL compiler into a board configuration file. To be able to effectively program the FPGA and use the kernels, the host machine has to execute the proper interactions with the board. The code running on the host machine is in charge of the following:

- Get a full understanding of the board environment.
- Load from disk the kernel binaries, and program the FPGA using the PCIe communication channel.
- Initialize the data structures residing in the board global memory; for each structure on the board, its counterpart in the host memory has to be allocated, and their relation explicitly stated. When data has to be copied from the host to the FPGA, and viceversa, the proper command have to be issued.
- Set the kernel arguments, and have them start.
- Manage synchronization between kernels.

6.2 Reading MRI Scanner Frames

For this research work, we assume that data from the MRI Scanner is stored in the host machine local disk. One MRI frame is described by the following files:

1. Sample Points file: binary file containing a set of data structures describing the sample points. Each data structure consists of four fields: k-space x and y coordinates, real part and imaginary part of the sample value.

2. Weight files: set of float numbers used during the Density Compensation step. We assume that to be already pre-computed and available; techniques on how to compute them are presented by E. Pezzottti in [1].

When evaluating the frame rate, multiple different files are being read an processed, and the FPGA output written back to file. A write-back thread has been spawned to take care of the writing back tasks without blocking the main operation flow. Actual parallelism can be obtained thanks to the multi-core nature of the host machine. Due to these precautions, and a well-structured kernel pipeline, the input-output disk operations and the PCIe data transfer do not act as a performance bottleneck.

6.3 Kernels Synchronization

OpenCL kernels share data stored in global memory. As a consequence, to avoid race conditions and inconsistencies, a synchronization mechanism has to be established to regulate their execution start. In particular, the following has been determined:

- The Target Interface kernel shall start after the Re-Gridding kernel has completed.
- The first IFFT Fetch kernel shall start after the Target Interface kernel has completed.
- The second IFFT Fetch kernel shall start after the first IFFT Transpose kernel has completed.

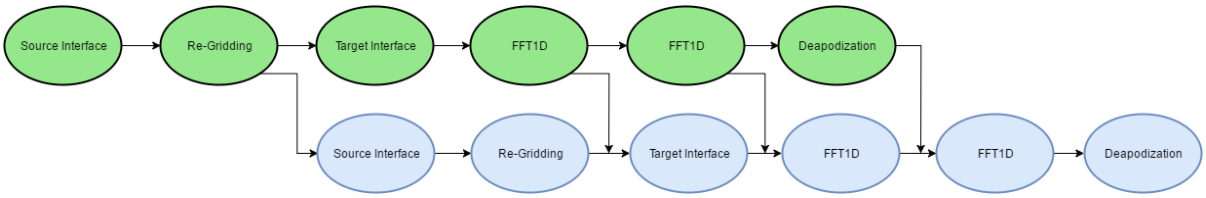


Figure 27: Directed Acyclic Graph to show the synchronization scheme. A child node can start only when its parents have terminated their execution.

- The Deapodization kernel shall start after the second IFFT Transpose kernel has completed.

To implement this synchronization scheme, two approaches have been attempted.

6.3.1 Pthreads Synchronization

For this approach, Unix *pthreads* have been exploited. The host code main thread spawns several threads, one per each kernel. Each thread is in charge of dictating the start of the kernel it is linked to. The synchronization is implemented by means of mutexes: each thread is locked on a different mutex, which is unlocked by the thread it should be preceded by. As an example, consider the synchronization between Target Interface and Re-Griddinig kernels. The thread in charge of the Re-Gridding kernel starts and is locked on mutex m_1. When the Target Interface kernel terminates, it unlocks mutex m_1, hence allowing the Re-Gridding thread to start its kernel.

6.3.2 Altera OpenCL Events Synchronization

OpenCL offers its own synchronization mechanism through structures called *Events*. When setting a kernel, the host code can specify a set of Events that must be completed before that

kernel can start. At the same time, the host code can specify, for each kernel, an Event that is set to completed when that kernel terminates. Therefore, implementing the synchronization scheme in Figure 27 is fairly straightforward.

The Events based approach provides for code simplicity and for better performance in terms of kernel execution delay, and has hence been chosen as part of the final implementation.

6.4 Host Code Pipelining

The final goal of the system is that of processing subsequent frames at the highest rate possible. To achieve that, both the kernel and the host code have been designed to enable software pipelining, allowing processing for frame $i+1$ to begin before the processing for frame i has terminated. Figure 28 shows the pipeline concurrency scheme: the processing for frame $i+1$ can be issued as soon as the Target Interface kernel for frame i has terminated. As it can be observed from the image, the deapodization step for frame i is executed in parallel to the operations on frame $i+1$, hence leading to no increase in the overall processing time. Furthermore, this type of pipelining comes without any cost in terms of memory required, since just one copy for each buffer needs to be allocated.

6.5 Samples Reordering

6.5.1 Software Reordering

As mentioned in Section 4.7, to fully enroll the pipeline for the Target Interface kernel we must ensure that successive sample points do not generate dependencies. Given Assumption 1, and knowing the trajectory of the frames that are going to be processed, it is possible to re-order the samples in such a way that the above condition is guaranteed.

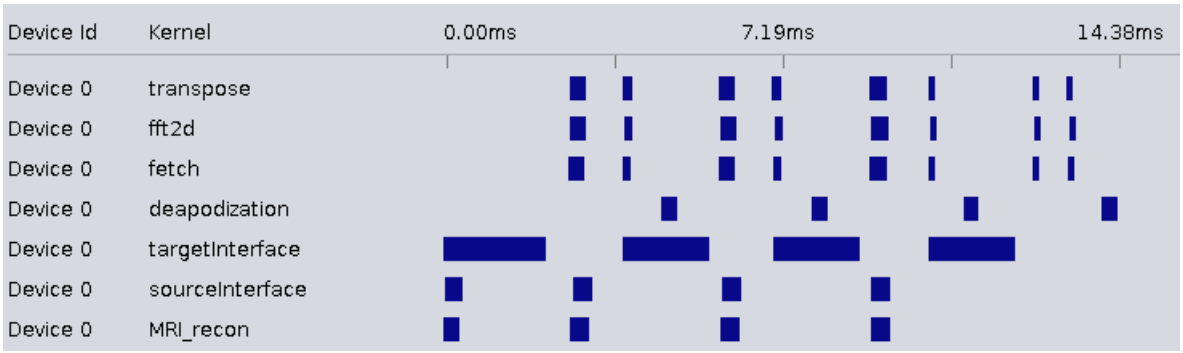


Figure 28: Example of pipelined software execution. The MRIrecon kernel in the picture is the Re-Gridding kernel.

A greedy algorithm has been developed to obtain a working solution in reasonable time. Optimal solutions can be obtained, that provide small gains at the expense of much higher computational time. Appendix A displays the pseudocode for the algorithm used. The idea behind the algorithm is that of progressively finding sample points that can be inserted in a valid sequence, where a valid sequence is defined as an ordered list of samples that, when fed in order to the Re-Gridding kernels, do not generate data dependencies in the Target Interface kernel queues. As soon as a sample that does not generate dependencies is found, it is inserted in the sequence. If no such sample can be found, a suitable amount of *filler points* is inserted in the sequence. A filler point is a fictional sample with value zero for both its real and imaginary part, located in an area of k-space where no other samples are located. By inserting a filler point, we are providing the queues with enough buffer to avoid data dependencies. The drawback with filler points is that more samples have to be processed. If the order of magnitude of the amount of needed filler points is comparable to that of the sample points, a notable delay is introduced.

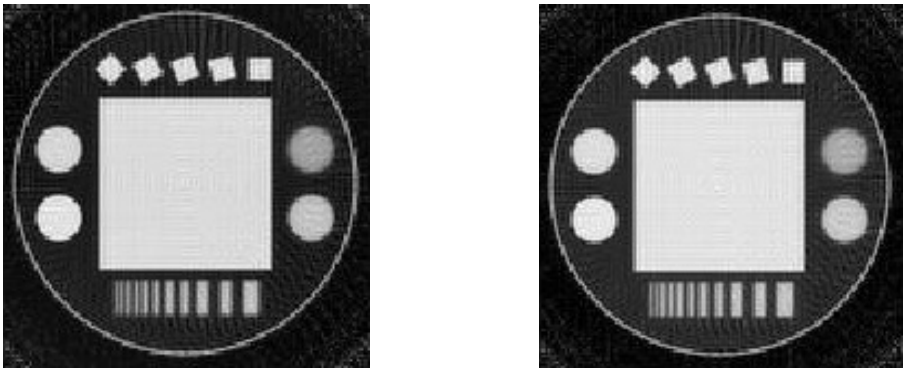
However, for all the tests performed, the number of filler points required were in the range [5,12], hence causing only a negligible increase in the number of samples to be processed.

Once the order of the samples and the position of the fillers for a given trajectory has been computed, it can be stored in a file or locally, and be used for all the frames acquired with that trajectory. When no reordering is performed, values for a frame are read sequentially from the frame files and sequentially stored in an array, which is passed to the Source Interface kernel. With reordering, as the frame files are read, the values are inserted in an array at the index provided by the reordering. The vector has to be slightly resized to accomodate for the filler points, and is then set as a kernel argument for the Source Interface kernel.

6.5.2 Hardware Reordering

The Software reordering approach requires the system to perform an offline operation before the actual processing of a batch of frames. However, while this requirement is acceptable in theory, its shortcomings might manifest themselves in real world situations. As an example, consider the case where the number of elements in a batch is small: the overhead required for fillers positioning might exceed the execution time gain obtained with the *ipdev* directive, hence canceling the benefits introduced with the reordering.

A way of overcoming the limitations of software reordering is by trying to implement a similar feature directly in hardware. Remembering that the objective is ensuring that successive source points do not generate target point contributions that violate the data consistency condition in 4.7, several attempts were made.



(a) (b)

Figure 29: (a) The image reconstructed in emulation (b) The image reconstructed by the FPGA hardware.

6.5.2.1 No condition enforcement

As a first test approach, to understand the extent of the loss in image reconstruction accuracy when data inconsistency occur, the *ipdev* directive was used without enforcing any condition on the input samples. Figure 29 shows the difference in image reconstruction between emulation and actual hardware implementation: as it can be observed, the two images are visually nearly identical. However, if comparing the values for pixel p_{ij} in the two matrices, an error ϵ_{ij} such that $10^{-1} < |\epsilon_{ij}| < 10^{-3}$ was found. This possibly implies that some data corruption, although minor, happens when running the algorithm on the synthesized hardware.

6.5.2.2 Circular buffer

As a first resolution approach, a circular buffer has been implemented for each queue. The circular buffer is of a generic size Q, and each element in the buffer consists of an index and a

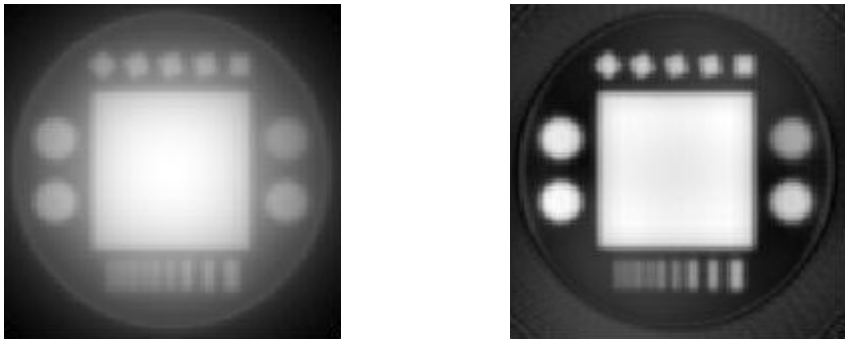
complex value. The buffer is initialized so that all indexes are equal to -1. Whenever a target point contribution is generated for queue i, such that the contribution affects the target point at index idx, the ith circular buffer is traversed: if any element in it has an index equal to idx, then the contribution is summed to the value of that entry in the buffer. If no such element is found, the circular buffer writes its oldest value into the channel to the Target Interface kernel, and the latest contribution is inserted in the buffer. This technique ensures that Target Interface always receives sequences of Q contributions such that all indexes differ. Although the emulation of this technique worked perfectly, the compiler was not able to synthesize the hardware: the compilation failed multiple times, for unknown reasons.

A similar approach can be employed by positioning the circular buffer directly in the Target Interface kernel. The only difference is that when the elements in the buffer are shifted, the first element is written back to memory rather than sent to a queue. This techniques produces satisfactory results in emulation, and the compiler is able to synthesize the hardware. However, the image produced by the FPGA differed from the one obtained in emulation, as shown in **??**.

So far, we have not been able to explain this behavior. The compiler seems to be synthesizing the hardware correctly, yet data corruption still occurs.

6.5.2.3 Shift Register

One possible explanation for the behavior described above is that source code for a circular buffer is challenging for the compiler to translate into hardware correctly. Hence, we tried a similar approach, but using data structure more prone to hardware translation. A shift register was the natural hardware counterpart of a circular buffer.



(a) (b)

Figure 30: Difference between emulation (a) and FPGA (b) images for the circular buffer implementation.

Despite that, the results of the comparison between emulation output image and FPGA output image were similar to those in 6.5.2.2.

6.5.2.4 Hardware version of Software reordering algorithm

After successfully implementing Software reordering as described in 6.5.1, we tried to replicate a similar filler-based algorithm directly in hardware. The same shift register structures were used, one per each queue. Whenever the Re-Gridding generates a contribution, it checks the elements in the shift register. If an element with the same index as the current contribution is found, the register is shifted $Q + 1$ times and Q filler contributions are inserted, followed by the current contribution.

This approach strictly resembles Software reordering. Accordingly, the emulation output image and the FPGA output image should present no significant difference. Unfortunately, the

logic required for the process described above exceeds the cells available in the board. Hence, the solution could not be tested.

6.5.2.5 Approximate dependency resolution

Given the complications in finding an hardware solution that optimally resolved the data consistency problem, we tried investigating the trade-off for a solution that only partially resolved the issue.

In the Source Interface kernel, sample points are read sequentially from an array in global memory. Since acquisition trajectory are continuous in k-space, reading subsequent points dramatically increases the chances of processing source points that violate the consistency condition. If samples from the array are read with a step of size K between successive points then, by properly tuning K, the chances of processing illegal sequences of source points decreases, although they are not deterministically null. This technique comes at no expense in terms of hardware resources. Unfortunately, although the emulation and FPGA images are visually identical, there are still relevant differences in the pixel values, despite being smaller than when no enforcement at all is applied.

CHAPTER 7

RESULTS

In this Chapter, the actual results in terms of performance for the proposed implementation are presented, together with comparisons with the most relevant works in the related literature.

7.1 Comparison with Previous Works

To compare the architecture with other works in the literature, the same set of configuration parameters had to be adopted, in order for a fair comparison to be possible. To achieve that, the oversampling factor α was set to 2, the number of source points was taken equal to $N \times N$, where N is the size of one side of the reconstructed image, and the convolution windows W was considered equal to 4. For the convolution, a Kaiser-Bessel kernel with nearest-neighbor interpolation has been chosen, and $S = 10$. Deapodization was performed on-the-fly.

The metric for performance are *fps*, frames per second, and represents the number of different MRI images that can be thoroughly processed per second. For our measurements, multiple different frames have been fed to the system, and each reconstructed image was written back to file. The overall execution time was provided by Altera OpenCL Profile.

Figure 31 shows the comparison of the proposed architecture for the Re-Gridding phase only with respect to the CPU implementation in [2] and the FPGA implementation in [3]. The proposed work outperforms both for all image sizes.

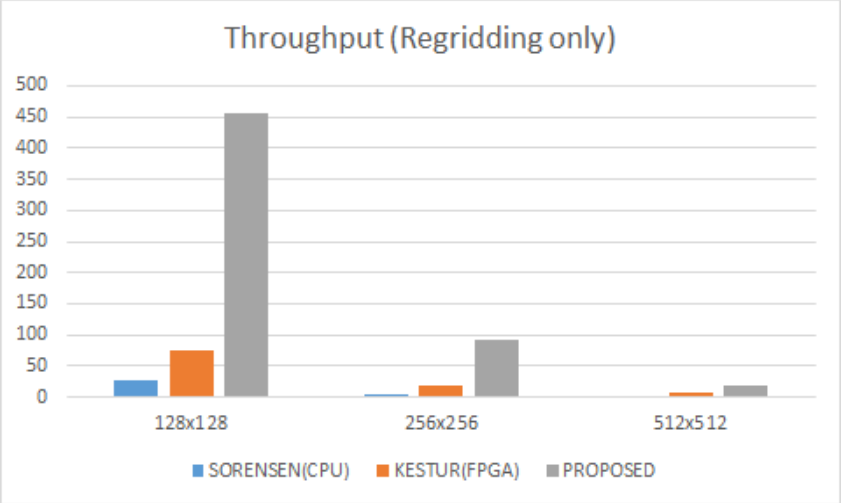


Figure 31: Throughput for the architecture when performing Re-Gridding only, with respect to Sorensen et al. and Kestur et al., for increasing image size.

Figure 32 compares the proposed architecture to the GPU solution by Sorensen et al. [2] when all the MRI reconstruction phases are carried through. The number of frames per seconds processed are comparable for all image sizes.

We tried to compare the proposed solution to the work in [5], but it is particularly evasive when describing the benchmark used for acquiring the proposed results, making every fair comparison attempt hard.

In terms of accuracy, no comparison with previous works was possible, since accuracy metrics were either absent, or very different from the ones used.

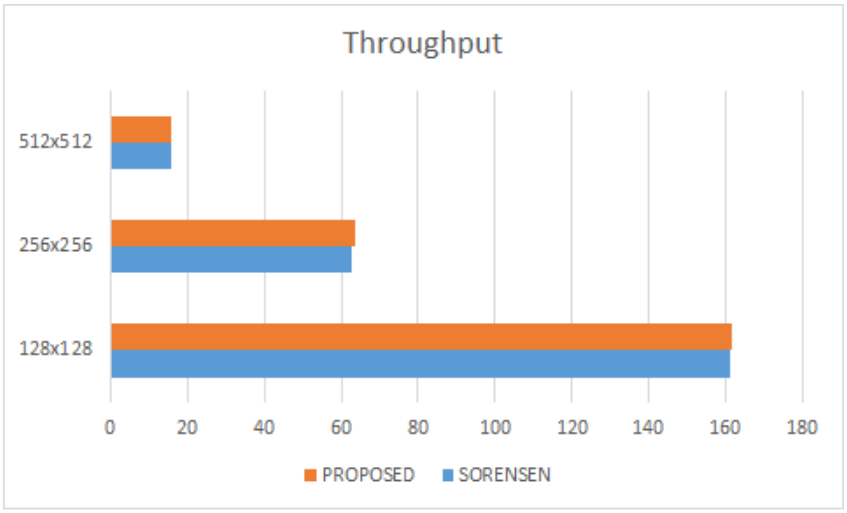


Figure 32: Throughput for the architecture when the whole reconstruction is performed, with respect to Sorensen et al., for increasing image size.

7.2 Performance for Optimized Architecture

The architecture has then been compiled with the optimal parameters found in the previous Sections. More specifically, the oversampling factor α has been taken equal to 1.32, a value of 2 was chosen for the convolution window W, and the number of source points was taken equal to $N \times N$, where N is the size of one side of the reconstructed image.

The architecture for images of size 512×512 was recompiled with the above parameters. The performance in terms of frames per seconds, with respect to the architecture in 7.1, received a boosts of 200%, reaching a 20.3, while still providing an SSIM higher than 0.9. For smaller images, this trend is amplified even more. The bottleneck in this improvement is then

represented by the host code, and by the data transfer between host machine and FPGA over PCI Express, over which there is little control.

7.3 Further Work

7.3.1 3D Extension

One of the possible applications for MRI is 3 dimensional imaging. The goal of 3D imaging is that of reconstructing a volume of the body part under scan. This can help providing the medical experts with more insights and a broader vision of the area under control.

Obtaining a volumetric representation of a body area with MRI scanners can be achieved in two different fashions:

1. By acquiring and reconstructing adjacent 2D slices. The reconstructed slices can then be assembled together to create a volume. A loss in accuracy is given by the presence of borders between slices, possibly leading to imperfections in the 3D image.
2. By acquiring the entire volume of image with one unique acquisition. This can be achieved by means of 3 dimensional gradient application, and produces images with higher spatial resolution and SNR (Signal to Noise Ratio). However, acquisition times are much longer, since more complicated gradient patterns have to be followed.

While for case 1, the problem reduces to successive reconstruction of 2D images, to handle case 2 the approach described in this work has to be slightly adapted.

Source points are acquired in 3 dimensions, and therefore consists of 3 coordinates. When interpolating the contribution by a sample point s_i, the convolution windows expands over a

cube of side σ, so that points in a micro-volume surrounding s_i are influenced. The convolution kernel will be a function $\Phi(x, y, z)$ whose value depends on the distance in space between s_i and the target point under analysis. More specifically, if the kernel is separable:

$$\Phi(x, y, z) = \Phi_{1D}(x)\Phi_{1D}(y)\Phi_{1D}(z) \tag{7.1}$$

then the transformation back to image domain can be obtained with a 3D Inverse Fourier Transform, implementable as a the application of a 1D IFFT on every direction. Regarding the division of the target space tiles, they turn 3 dimensional. Beside a different indexing procedure in the array representation of the tile, few changes have to be implemented in the code. The scalability of the solution is not compromised, although the choice of the number of tiles L must be more carefully designed. Supposing that the target space volume is of size $\alpha N \times \alpha N \times \alpha N$, the execution time for the Target Interface kernel is:

$$\begin{aligned} T &\propto L(\frac{\#_{sp}W^3}{L} + \frac{(\alpha N)^3}{L} + Q) \\ &\propto \#_{sp}W^3 + \alpha^3 N^3 + LQ \end{aligned} \tag{7.2}$$

Moreover, consider that to fit the processing for a 2D image of size $N \times N$ in the FPGA a number of tiles $L = L_0$ had to be chosen. The memory requirements for a corresponding 3D image of size $N \times N \times N$ is αN times as much as for the 2D counterpart. Hence, we can expect the required number of tiles to be around $L = \alpha N \cdot L_0$. The third term, LQ, becomes equal to $\alpha N \cdot L_0 Q$ and is not neglectable anymore, although not being the most relevant contribution.

It is easy to see that the increase in reconstruction time is significant with respect to the 2D case. However, if the image size is kept small, the full volume processing can be expected to terminate in a few seconds.

One possible drawback is the increase in logic utilization due to the higher indexing complexity. A possible solution is the substitution of floating point arithmetic with fixed point arithmetic, as proposed by [1], at the expense of some accuracy and computational time.

7.3.2 Combination with Other Techniques

A further chance for performance gain relies in the combination of the described approach with other techniques. In [1], E. Pezzotti describes how to exploit the symmetry properties of k-space to reduce the number of effectively processed sample points without significant loss in accuracy. By merging this technique with the optimal interpolation kernel choice, the computational time can be reduced even further.

CHAPTER 8

CONCLUSIONS

The problem of MRI reconstruction from non-Cartesian trajectories is not new to the academic world, and has been the subject of extensive research. In this work, we tackle the problem targeting FPGAs as a destination hardware, and exploiting novel high level design tools like OpenCL.

The physics behind Magnetic Resonance Imaging is described, and the Fourier Transform as link between k-space and image domain is proved. The problem of Non-uniform Fast Fourier Transform is then stated, together with possible solutions, with focus on Re-Gridding based techniques. All the phases that constitute reconstruction by Re-Gridding are portrayed, both from a mathematical standpoint and from the effect they have on the final image. Each phase is implemented, and the trade-offs in accuracy, performance and resource utilization are faced and solved favoring a good balance between the three.

More specifically, the kernel functions in the interpolation phase are analyzed, aiming at minimizing the oversampling factor while keeping the convolution windows size to the smallest possible value of 2. The features of some of the most common kernels are highlighted, and the Kaiser-Bessel function is identified as the best choice. Moreover, analytic kernel computation is compared to the linear and nearest-neighbor interpolation of a sampled kernel, showing that for sufficiently high kernel sampling values the techniques show no relevant difference in reconstruction accuracy.

The OpenCL standard and its main features are described, together with the coding practices suitable for the problem under analysis. The various design attempts are portrayed and the cause for their failure deduced. Following, the final proposed architecture is explained in detail, going through the challenges faced. Among the latter, the concept of trajectory reordering is introduced: techniques for exploiting known trajectories to achieve lower-latency pipelining are designed and attempted both at the software and hardware level. Software reordering achieved the goal through a greedy algorithm that reaches semi-optimal results. Most of the attempts for hardware reordering failed because of compiler limitations and with no error description, showing that there is still room for improvement in the new synthesis tools that are part of the OpenCL design flow.

The code running on the host machine has been optimized and multi-threading exploited to achieve high parallelism and overcome the bottlenecks introduced by disk operations and by buffer read/writes with the FPGA over PCIe. Multiple different frames are read subsequently, and effectively processed and then written back to disk.

The architecture was synthesized and the bitstream programmed on the FPGA. The results produced by the hardware matched those predicted with the emulation, and the obtained results in terms of performance were compared to the existing literature. The comparison show that the proposed solution matches and sometimes outperforms current standards in terms of frames per second, while maintaining the property of scalability, and high reconstruction accuracy.

Overall, this work demonstrates that the combination of modern FPGA technologies and higher abstraction design languages can yield results comparable to low level HDLs and GPUs,

while keeping the advantages intrinsic to FPGAs: flexibility and low power. Moreover, the benefits of high-level languages are underlined, as very complex architectures can be described with more human friendly C-like structures, simplifying design and hence reducing time-to-market. At the same time, however, the current limitations of such new languages are exposed, as fully exploiting their capabilities requires expertise that is not common knowledge and must be acquired through a sometimes long trial-and-error process. Hardware synthesis can at times fail with apparently no reason, or produce results unexpected from the compilation reports. The latter are all downsides that are caused by the smaller exposure and usage that these languages have had. We firmly believe that they will be overcome over time, and that the trend in complex hardware solutions design will privilege higher level languages, particularly when combined with flexible supporting devices like FPGAs.

APPENDICES

Appendix A

SAMPLES REORDERING PSEUDOCODE

```
coord ← List of source points coordinates;
SP ← Empty list;
Mark all elements in coord as unchecked;
while coord != {}:
    bool sp_inserted ← False;
    while coord has unchecked elements:
        select unchecked source point si;
        bool allowed ← can si be inserted in SP without violating conditions?
        if allowed == True:
            SP = SP + si;
            Mark all elements in coord as unchecked;
            sp-inserted ← True;
            break;
    else:
        Mark si as checked;
        if sp-inserted == False:
            FillerPoint fp <- generate-filler-point();
            SP = SP + fp;
            Mark all elements in coord as unchecked;
```

Appendix B

LETTER FOR PERMISSION TO USE COPYRIGHT MATERIAL

Department of Electrical and Computer Engineering University of Illinois at Chicago

03/14/2017

Emanuele Pezzotti,

I am writing to request permission to use images and graphs from your publication (Pezzotti, E: Efficient Non-uniform Fast Fourier Transform (NuFFT) Implementation for MRI Processing on FPGA, Master's Thesis, University of Illinois at Chicago, 2016) in my thesis. This material will appear as originally published. Unless you request otherwise, I will use the conventional style of the Graduate College of the University of Illinois at Chicago as acknowledgment. A copy of this letter is included for your records. Thank you for your kind consideration of this request.

Sincerely,

Alex Iacobucci

737 S. Claremont Avenue, Chicago, IL, 60612

_____

The above request is approved.

Approved by:_____Emanuele Pezzotti_____Date:_03/14/2017___

CITED LITERATURE

1. Pezzotti, E.: Efficient Non-uniform Fast Fourier Transform (NuFFT) Implementation for MRI Processing on FPGA. Master's thesis, University of Illinois at Chicago, 2016.

2. Sørensen, T. S., Schaeffter, T., Noe, K. Ø., and Hansen, M. S.: Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. IEEE Transactions on Medical Imaging, 27(4):538–547, 2008.

3. Kestur, S., Park, S., Irick, K. M., and Narayanan, V.: Accelerating the nonuniform fast fourier transform using fpgas. In Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, pages 19–26. IEEE, 2010.

4. Cheema, U. I., Nash, G., Ansari, R., and Khokhar, A. A.: Power-efficient re-gridding architecture for accelerating non-uniform fast fourier transform. In 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pages 1–6. IEEE, 2014.

5. Li, L. and Wyrwicz, A. M.: Design of an mr image processing module on an fpga chip. Journal of Magnetic Resonance, 255:51–58, 2015.

6. Dutt, A. and Rokhlin, V.: Fast fourier transforms for nonequispaced data. SIAM Journal on Scientific Computing, 14(6):1368–1393, 1993.

7. Maciejewski, M. W., Qui, H. Z., Rujan, I., Mobli, M., and Hoch, J. C.: Nonuniform sampling and spectral aliasing. Journal of Magnetic Resonance, 199(1):88–93, July 2009.

8. Staff, A.: Opencl and the amd app sdk v2.4. http://developer.amd.com/resources/articles-whitepapers/opencl-and-the-amd-app-sdk-v2-4/, 2011.

9. Intel: Intel FPGA SDK for OpenCL - Best Practices Guide, 2016.

10. Intel: Intel FPGA SDK for OpenCL - Programming Guide, 2016.

11. Intel: OpenCL 2D Fast Fourier Transform Design Example, 2016.

CITED LITERATURE (continued)

12. Beatty, P. J.: Rapid gridding reconstruction with a minimal oversampling ratio. IEEE Transactions on Medical Imagingl, 24(6):799–808, 2005.

13. O'Sullivan, J. D.: A fast sinc function gridding algorithm for fourier inversion in computer tomography. IEEE Transactions on Medical Imaging, 4(4):200–207, Dec 1985.

14. Jackson, J. I., Meyer, C. H., Nishimura, D. G., and Macovski, A.: Selection of a convolution function for fourier inversion using gridding [computerised tomography application. IEEE Trans. Medical Imaging, pages 473–478, 1991.

15. Veldhuizen, T.: Measures of image quality. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node18.htm, Accessed on: 08/11/2016.

16. Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P.: Image quality assessment: From error visibility to structural similarity. IEEE TRANSACTIONS ON IMAGE PROCESSING, 13(4):600–612, 2004.

17. Pauly, J.: Material for the course ee369c: Medical image reconstruction. http://web.stanford.edu/class/ee369c/, Accessed on: 05/05/2016.

18. Kawaji, K.: Cardiac mri k-space data set. http://hdl.handle.net/11466/wbsB5C, 2014.

19. Pezzotti, E., Iacobucci, A., Nash, G., Cheema, U., Vinella, P., and Ansari, R.: Fpga-based hardware accelerator for image reconstruction in magnetic resonance imaging (abstract only). In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '17, pages 293–293, New York, NY, USA, 2017. ACM.

20. Filomena Santarelli, M.: Advanced Image Processing in Magnetic Resonance Imaging, chapter Basic Physics of MR Signal and Image Generation. CRC Press, 2005.

VITA

NAME	Alex Iacobucci
EDUCATION	
	Master of Science in Electrical and Computer Engineering, University of Illinois at Chicago, May 2017, USA
	Specialization Degree in Embedded Systems Computer Engineering, Dec 2016, Polytechnic of Turin, Italy
	Bachelor's Degree in Computer Engineering, Jul 2014, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Nearly bilingual
	2015 - IELTS examination (7.5/9)
	A.Y. 2015/16 One Year of study abroad in Chicago, Illinois
	A.Y. 2011/2012 to 2014/15. Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Fall 2016	Graduate Research Assistantship (RA) position (20 hours/week)
Fall 2015	Italian scholarship for TOP-UIC students
WORK EXPERIENCE	
May 2016 - Dec 2016	Software Engineer at HERE, Chicago. Worked in Full Stack Development, using Scala and Java for the back-end, and AngularJS and JavaScript for the front-end. Researched and implemented Natural Language Processing solution, with focus on Natural Language Understanding and Information Extraction. The solutions involved state-of-the-art techniques in Artificial Intelligence.