Rapid on-line reconstruction of non-Cartesian Magnetic Resonance images using commodity graphics cards

Grzegorz Tomasz Kowalik

Institute of Cardiovascular Science,

University College London

Research Degree: Cardiovascular Science

I, Grzegorz Tomasz Kowalik confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

This work grew from a tinny *'I'll show you all!'* into a bellowing *'take it off me!'*; from no more than 4 hours a day tapping on a keyboard into 20-hour marathons; from sunny rays coming through a big bright window into a 16°C sterile and fluorescent-lit environment; from little toy projects into crying orphans …

This work is dedicated to all volunteers that endured all the *'new inventions of mine'* …

To all proof-readers for suffering a version-after-version of this text and their invaluable comments and suggestions that made this work so much richer …

To all *'circle drawers'*, especially having in mind Bejal Pandya; that *'gobbled'* hundreds of thousands of images bringing my projects to their conclusions …

To radiographers: Rod Jones and Wendy Norman for their help in running the examinations and all the useful insights into the clinical MR world …

To Vivek Muthurangu for his continuous guidance and practical, sober drive to stay concentrated on the bigger picture, but most of all, for being positive, resilient and forbearing in the face of my many flaws …

To Jennifer Steeden for being an amazing colleague, friend and teacher without whom I would not be able to finish; whose patience withstood my many attempts; for keeping me focused and on track …

… for all the help, goodwill and kindness; for all the jokes and talking-to that brought me to the conclusion of this work; for helping and pushing me to put this work together,

Thank you

Abstract

In Magnetic Resonance Imaging, energy of electromagnetic waves is used to excite protons placed in a static magnetic field. This generates a signal, which is further spatially encoded with linear magnetic field gradients. The signal exists in frequency domain called *k*-space. Traditionally, the signal is sampled in lines stored on a Cartesian grid. Next, Fast Fourier Transform is applied to generate images. However, the consecutive manner (line-by-line) of this strategy makes it very slow. Faster sampling strategies exist, but acquisitions with them require a more complex image reconstruction process. There is an obvious trade-off between acquisition time and complexity of image reconstruction. Real-time assessment protocols for day-to-day clinical work demand both data acquisition with rapid sampling trajectories and fast, robust image reconstructions.

Computational solutions in form of parallel architectures can be used to aid image reconstruction, which has been proven to significantly speed-up reconstruction process. Regrettably, this is often done in off-line mode, where the data need to be downloaded from the scanner and reconstructed elsewhere. This process hinders the clinical workflow substantially.

This work describes challenges entailed with translation of advanced imaging protocols into the clinical environment; (i) use of the advanced sequences is limited by their reconstruction time, and (ii) fast implementations exist but they still run in off-line mode. These were addressed and resolved with development of a novel online, heterogeneous image reconstruction system for Magnetic Resonance Imaging. The external platform was designed to support fast implementation of advanced reconstruction algorithms. An external computer equipped with a Graphic Processing Unit card was integrated into the scanner's image reconstruction pipeline. This allowed direct access to high performance parallel hardware on which the rapid data reconstruction can be realised. Also, the automation of data transmission and reconstruction execution has preserved the non-interrupted assessment workflow.

Table of Contents

List of Equations

List of Figures

List of Tables

1. Introduction

1.1 Magnetic Resonance physics fundamentals

1.1.1 Signal generation

Atomic nuclei with odd atomic weight and/or odd atomic number have angular momentum ($\vec{J}$) referred to as a spin (1-4). Nuclei with non-zero spin have an associated magnetic moment ($\vec{\mu} = \gamma\vec{J}$), where γ is the gyromagnetic ratio which is a property of specific nuclei. The magnetic moment magnitude is given by the following equation;

$$|\vec{\mu}| = \frac{\gamma h}{2\pi}\sqrt{I(I+1)} \qquad 1\text{-}1$$

Equation 1-1 Magnitude of a magnetic moment.

Where I is the nuclear spin quantum number, which relates to nuclei atomic mass and charge number and h is Planck's constant. For a nucleus to be MR active I must be non-zero, therefore generating a non-zero magnetic moment. However, in thermal equilibrium and absence of a strong external magnetic field the direction of $\vec{\mu}$ is random. In consequence there is no net magnetic field from a population of spins.

When placed in an external magnetic field ($\overrightarrow{B_0}$) the spins undergo several changes. Primarily, the spins are separated (quantized) into $(2I+1)$ quantum states (Zeeman splitting) (4, 5), as shown in Equation 1-2.

$$m_I \in -I, -I+1, \ldots, I \qquad 1\text{-}2$$

Equation 1-2 Possible spin quantum states under an external magnetic field.

Therefore, nuclei where $I = \frac{1}{2}$ (i.e. ^{1}H, ^{13}C, ^{19}F etc.) have two possible quantum states $-\frac{1}{2}$ and $\frac{1}{2}$. Importantly, these quantum states have discrete energy levels directly associated with them (Equation 1-3).

$$E = -\vec{\mu} \cdot \overrightarrow{B_0} = -\mu_z B_0 = -m_I \frac{\gamma h}{2\pi} B_0$$

$$E_\uparrow = -\frac{\gamma h}{4\pi} B_0$$

$$E_\downarrow = \frac{\gamma h}{4\pi} B_0$$

$$\Delta E = E_\downarrow - E_\uparrow = \frac{\gamma h}{2\pi} B_0$$

1-3

Equation 1-3 Quantum state energy formulations.

These energy states are important in determining the magnetic properties of the spin population. In an external magnetic field the magnetic moments associated with individual spins precess around the magnetic field axis as shown in Fig. 1-1. In this state the magnetic moment can be defined in terms of its component along the external field (μ_z), an orthogonal component ($\mu_{x,y}$), an angle (θ) and the precession frequency.

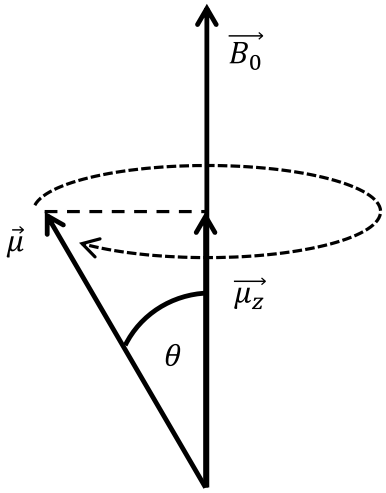

Fig. 1-1 Schematic visualization of a magnetic vector precessing around an external field.

$$\omega_0 = \gamma B_0$$

1-4

Equation 1-4 Precession frequency

The orientation of these precessing magnetic moments (either parallel or anti-parallel to the external field) is determined by the energy levels of the spins. Spins aligned with parallel directionality are in the *low* energy state ($E_\uparrow$) and those in anti-parallel alignment are in the *high* energy state ($E_\downarrow$).

In a given spin population the proportion of the low to high energy spins is governed by the Boltzmann distribution;

$$\frac{N_{low}}{N_{high}} = e^{\frac{\Delta E}{KT_s}} \qquad 1\text{-}5$$

Equation 1-5 Ratio of the low to high energy spins.

The ratio of spins in the lower energy state (N_{low}) to those in the higher energy state (N_{high}) is a function of the energy difference between the states (Equation 1-3) and the temperature of the system (T_s), K - the Boltzmann constant. Consequently, the ratio increases with the strength of the external field and falling system temperatures. There is always a small excess of nuclei in the *low* energy state. For the room temperature (25^{o}C) and B_0=1.5T there are ~10 in a million protons (^{1}H – Hydrogen molecules) in the *low* energy state. This excess is substantial enough to create a nonzero net magnetisation vector parallel to the external magnetic field.

As was mentioned an important property of the magnetic moment under the external field is its precession around it. However, the net magnetic vector does not exhibit the precession around the external field. This is a result of the random distribution of the traversal magnetisation ($\overrightarrow{\mu_{xy}}$) from each nucleus. Nevertheless, an aligned magnetic vector can be *tipped or nutated* into the x-y plane by means of an external magnetic wave. The wave must have the same frequency as the precessional frequency (Larmor frequency). In classical description, the magnetic wave is seen as a force of torque, which is applied orthogonally to the torque of magnetic moment, oscillates as it precesses. In the quantum description, the electro-magnetic energy of the external impulse is used to force spins to change quantum state (Planck's law – absorption). The energy of the external radiation ($E_{ext} = \frac{h\omega_{ext}}{2\pi}$) must match the energy difference between the states (ΔE - Equation 1-3). Thus, the electro-magnetic wave must propagate with the Larmor frequency. The matching frequency is called resonance frequency and the whole process is known as magnetic resonance (MR).

MR static magnetic fields for clinical applications are typically in range of 0.1 to 7 Tesla (T), therefore the resonance frequency of Hydrogen nucleus

(the most abundant element in the human body, as up to ~65 % of it consists of water - H_2O) is in range of radio frequencies and the external impulse is referred to as a radio frequency pulse (RF-pulse).

RF excitation is used to reorient the net magnetisation from alignment with the external magnetic field. After excitation, the *tipped* magnetisation has a non-zero component in a plane that is orthogonal to the external field. Consequently, magnetic resonance generates the signal through the net alignment and synchronisation of the magnetic moments with each other. In this way they do not cancel each other out and the resultant signal can be detected.

These net magnetic field changes ($\vec{m}(t)$) can be measured as an induced electric current in an antenna (more commonly called a coil) placed next to the sample of interest. The signal generated in the receiver coil dies out with time, which is referred to as Free Induction Decay (FID). The signal disappearance is caused by inhomogeneity of the external magnetic field and the spin-spin relaxation (destructive interaction of spins in close spatial proximity). Both factors force magnetic moments to get out of synchronisation and have a destructive effect on strength of the generated signal. This process is dictated by T_2^* time, or the transverse relaxation time (5).

$$\vec{m}(t) \propto \rho e^{-i\gamma B_0 t}$$
$$\vec{S}(t) \propto \vec{m}(t) e^{-\frac{t}{T_2^*}} \qquad \text{1-6}$$

Equation 1-6 Simplified MR signal formulation.

Equation 1-6 describes (in a simplified form) a generated MR signal ($\vec{S}$) as a function of initial net magnetisation vector in the orthogonal plane ($\vec{m}$) that is reduced in time with the exponential rate (T_2^*). Strength of the initial magnetisation ($\vec{m}$) vector is directly proportional to the density of nuclei in a sample (ρ) that took part in MR, while its oscillations are dictated by Larmor frequency.

The *tipped* spins also realign with the external field and recover magnetisation in that direction. This process is called the longitudinal relaxation described with T_1 time. In every multiple of T_1 second magnetic momentum

recovers ~63% of its magnetisation along the direction of the external magnetic field.

The signal described forms the bases of the MR imaging and in the next section I describe how the spatial information is encoded into it.

1.1.2 Spatial encoding

MR experiments allow measurement of nuclei density by measuring FID signal that originated from an examined sample. Although the signal does not carry any spatial information this can be introduced using magnetic field gradients linearly varying through space. Larmor frequency depends on strength of an external magnetic field, which can be modified so it varies across space according to a vector field (gradient) $\vec{G} = \nabla B$ – a linear gradient of magnetic field B. In this case, precession frequency (ω) depends on spatial position vector ($\vec{r}$);

$$\omega(\vec{r}) = \gamma\left(B_0 + \vec{G} \cdot \vec{r}\right) \quad \text{1-7}$$

Equation 1-7 Larmor frequency as a function of linearly varying magnetic gradient.

The magnetisation can be now described by the following formula;

$$\vec{m}(t) \propto e^{-i\gamma B_0 t} \int \rho(\vec{r}) e^{-i\gamma \vec{G} \cdot \vec{r} t} dr \quad \text{1-8}$$

Equation 1-8 Impact of linearly varying gradients on the total magnetization vector.

Equation 1-8 dictates how net magnetic vector changes with strength of $\vec{G}$ and the time this magnetic gradient is applied.

Consequently, the simplest relation between position and precession frequency is obtained by applying a linearly varying magnetic gradient through space. This results in magnetic vectors from different spatial positions rotating with different frequencies. The constant $e^{-i\gamma B_0 t}$ can be demodulated, which is often referred to as moving to *rotating reference frames*, as a thought exercise to simplify the understanding of the processes. In this domain magnetic vectors of spins under no additional external magnetisation (i.e. $\vec{G} \cdot \vec{r} = 0$) are seen as if having no angular momentum. However, magnetic vectors of spins under external magnetic field that deviates from B_0 acquire different phases over time.

These are proportional to a difference in the external magnetic field ($\vec{G} \cdot \vec{r}$) and the length of time of its application (t).

This difference in phase across space can be seen as a wave. Unlike oscillations through time these are oscillations through space. We have a wave of changing phases as we move from one position to another (Fig. 1-2). This can be described as a wave vector. This is traditionally denoted with a letter k, and as every wave it has its own spatial frequency, and wave length λ;

$$k = \frac{1}{\lambda}$$
$$\vec{k} = \frac{\gamma}{2\pi}\vec{G}t \qquad \text{1-9}$$

Equation 1-9 $\vec{k}$ – spatially varying phase of magnetic vectors.

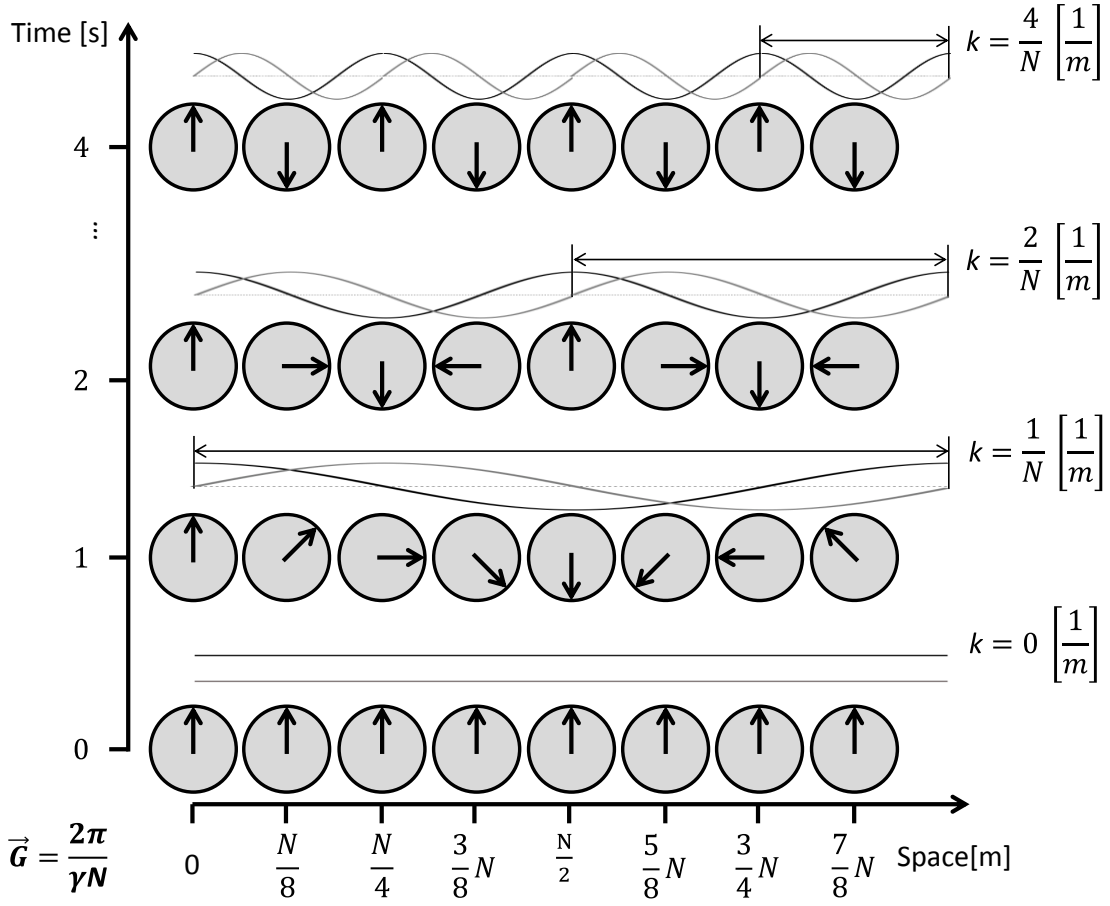

Fig. 1-2 Example of changes induced with a linear magnetic gradient in spatial oscillations (*k*-space position) with time.

The arrows represent spatially distributed spins presented in the rotating frame of reference. Equivalent continuous complex representation of the *k*-space is represented with the black (real component) and grey (imaginary component) plots.

Now, the superposition of all magnetic vectors can be seen as existing in a new domain, k-space domain;

$$\vec{S}(\vec{k}) \propto \vec{m}(\vec{k}) \propto \int \rho(\vec{r}) e^{-2\pi i \vec{k}\cdot\vec{r}} dr \qquad 1\text{-}10$$

Equation 1-10 Proportional relation between MR signal and total magnetic vector.

Equation 1-10 is derived from equations 1-6 and 1-9, and describes the acquired MR signal, depending on a position in k-space. The position in k-space can be changed with accumulation of time in a direction and strength of applied magnetic field gradient (Equation 1-9).

More importantly, the acquired MR signal in k-space is a Fourier Transformation (FT) of nuclei density $\rho(\vec{r})$. Therefore, inverse Fourier Transformation (iFT) can be applied to get the nuclei density distribution. In the case of spatial encoding with 2D magnetic field gradients, this can be visualised in a form of an image.

1.2 Fourier Transform and its properties

The Fourier Transform ($\mathcal{F}$) and its inverse ($\mathcal{F}^{-1}$) for unitary frequencies are formulated as follows;

$$\mathcal{F}_{\zeta}[f(x)] = \boldsymbol{F}(\zeta) = \int f(x) e^{-2\pi i \zeta x} dx$$
$$\mathcal{F}_{x}^{-1}[\boldsymbol{F}(\zeta)] = f(x) = \int \boldsymbol{F}(\zeta) e^{2\pi i \zeta x} d\zeta \qquad 1\text{-}11$$

Equation 1-11 Fourier Transform and its inverse formulations.

The first equation expresses transformation ($\mathcal{F}_{\zeta}$) of function f from x domain into its equivalent form $\boldsymbol{F}$ in the reciprocal domain, ζ. Conversely, the second equation expresses inverse transformation ($\mathcal{F}_{x}^{-1}$) of $\boldsymbol{F}(\zeta)$ into $f(x)$.

The following are properties of the transform that are used in this work. The properties are given without proofs as these can be found in other works;

$$\mathcal{F}_\zeta \sum_{j=-\infty}^{\infty} \delta(x - jM) = \frac{1}{M} \sum_{j=-\infty}^{\infty} \delta\left(\zeta - j\frac{1}{M}\right)$$
$$\mathcal{F}_x^{-1} \sum_{j=-\infty}^{\infty} \delta(\zeta - jM) = \frac{1}{M} \sum_{j=-\infty}^{\infty} \delta\left(x - j\frac{1}{M}\right) \quad \text{1-12}$$

Equation 1-12 Fourier Transform of a sampling function.

δ represents Dirac delta function, which is used to represent discretisation of continuous functions through a sampling process. Fourier Transformation of a sampling function results in a different sampling function.

$$\mathcal{F}_\zeta[f(x-a)] = e^{-2\pi i a\zeta}\boldsymbol{F}(\zeta)$$
$$\mathcal{F}_x^{-1}[\boldsymbol{F}(\zeta-a)] = e^{2\pi i a x} f(x) \quad \text{1-13}$$

Equation 1-13 Fourier Transform of a shifted function.

A shift of a transformed function results in modulation, phase shift in the reciprocal domain.

$$\mathcal{F}_\zeta\left[e^{2\pi i a x} f(x)\right] = \boldsymbol{F}(\zeta - a)$$
$$\mathcal{F}_x^{-1}\left[e^{2\pi i a\zeta}\boldsymbol{F}(\zeta)\right] = f(x + a) \quad \text{1-14}$$

Equation 1-14 Fourier Transform of a modulated function.

A modulation, phase shift of a transformed function results in a shift in the reciprocal domain.

$$\mathcal{F}_\zeta[f(x)g(x)] = \boldsymbol{F}(\zeta) * \boldsymbol{G}(\zeta)$$
$$\mathcal{F}_x^{-1}[\boldsymbol{F}(\zeta)\boldsymbol{G}(\zeta)] = f(x) * g(x) \quad \text{1-15}$$

Equation 1-15 Fourier Transform of multiplication of functions.

Fourier Transformation of a multiplication of functions results in convolution of their representations in the reciprocal domain.

$$\mathcal{F}_\zeta[f(x) * g(x)] = \boldsymbol{F}(\zeta)\boldsymbol{G}(\zeta)$$
$$\mathcal{F}_x^{-1}[\boldsymbol{F}(\zeta) * \boldsymbol{G}(\zeta)] = f(x)g(x) \quad \text{1-16}$$

Equation 1-16 Fourier Transform of convolution of two functions.

Fourier Transformation of a convolution of functions results in multiplication of their representations in the reciprocal domain.

1.3 Image reconstruction

Equations 1-6 and 1-10 give us an insight into the relationship between underlying nuclei density and the generated MR signal. From equation 1-10 we know that the underlying density function can be reproduced accurately, providing we have sufficient knowledge about the signal. The signal was expressed in terms of a new spatial frequency domain (*k*-space). The signal cannot be acquired instantly as *k*-space position is time dependent (Equation 1-9). However, the *k*-space can be *navigated* through using linear magnetic gradients. A sequence of changing magnetic gradients can be visualised in form of a path or trajectory of *k*-space positions. The signal is sampled while the series of gradients is being played out. Therefore a *k*-space path drawn by a predefined series of magnetic gradients is referred to as a sampling trajectory.

From equation 1-6 we know that the *k*-space signal dies out and sampling trajectories cannot be infinitely long. It is impractical to cover the whole *k*-space with single trajectory, as it may happen that by the end of it there is no signal to acquire. Instead a series of MR excitations and sampling on different, complementary trajectories is preferred. The selected trajectory defines how quickly *k*-space can be sampled. Ultimately, the trajectory and sampling strategy should be selected depending on the application, as well as the desired resolution of the reconstructed data.

1.3.1 k-space sampling

The previous section revealed that acquired k-space signal, $\vec{S}(\vec{k})$ is a Fourier Transformation of nuclei density function, $\rho(\vec{r})$. As mentioned, an inverse Fourier Transform operation can be used to find $\rho(\vec{r})$. Although k-space can be seen as a continuous function it is not feasible to acquire data in continuous fashion. The signal can be sampled as its discrete representation. This can be represented with a sampling function (generalised comb function);

$$\text{Ш}_{\vec{a}\in A}(\vec{x}) = \sum_{\vec{a}\in A} \delta(\vec{x} - \vec{a}) \qquad 1\text{-}17$$

Equation 1-17 Generalised comb function – sampling function.

$\vec{a}$ is the position of a sample from a set of sampling trajectory positions (A).

An m-dimensional comb function is defined as

$$\begin{aligned} \text{Ш}_{\vec{N}}(\vec{x}) &= \sum_{i_0}\sum_{i_1}\dots\sum_{i_{m-1}} m\delta(x_0 - i_0N_0, x_1 - i_1N_1 \dots x_{m-1} - i_{m-1}N_{m-1}) \\ &= \sum_{\vec{\imath}} \delta\left(\vec{x} - \vec{\imath}\vec{N}\right) \end{aligned} \qquad 1\text{-}18$$

Equation 1-18 m-dimensional comb function.

$\vec{N}$ represents a vector of distances between δ functions positions.

The discrete MR signal can be formulated in the following way;

$$\vec{\hat{S}}(\vec{k}) = \left(\vec{S}(\vec{k})\text{Ш}_{\vec{a}\in A}(\vec{k})\right) * \text{Ш}_{\vec{N}}(\vec{k}) \qquad 1\text{-}19$$

Equation 1-19 Discrete version of MR signal.

The convolution with $\text{Ш}_{\vec{N}}(\vec{k})$ is used to simulate signal's periodicity. Substituting Equation 1-10 into Equation 1-19 yields the definition of Discrete Fourier Transform (DFT) assuming $\vec{N}$ represents support bandwidth of *k*-space signal ($\vec{S}(\vec{k})$) and trajectory samples (A) are on equidistant positions from $\vec{N}$.

1.3.2 Cartesian trajectory

The traditional way of acquiring data is to read-out data samples on a uniformly spaced Cartesian grid. This means intervals between samples of each of acquisition dimensions are constant. This property allows use of Fast Fourier Transformation (FFT) algorithm to generate images, which is a very robust way of performing DFT(6). Although this simplifies image reconstruction, the data acquisition is very slow, as each trajectory line requires separate RF-pulse excitation.

1.3.3 Partial Fourier

One of the common ways of speeding up Cartesian acquisitions is by applying Partial Fourier technique(7). The Partial Fourier technique exploits redundancy in the acquired signal, assuming it is of real values (a quantity of nuclei density), in order to reduce the amount of data needed to be sampled. The Fourier Transform of a real function is symmetric. The frequency space is centrosymmetric with respect to its origin. If that was true for MR signals only half of *k*-space would need to be sampled, while the other can be calculated based on the read-out signal. Unfortunately, variations in the resonance frequencies, flow and motion may cause phase errors, thus make the frequency signal asymmetric and invalidate this assumption. Different techniques can be applied to correct for slowly varying phase errors (i.e. Conjugate Synthesis(8), Margosian(9), Homodyne(10), Cuppen(11), Projection onto Convex Sets(12) and techniques based on Finite Impulse Response filters(7)). These require calculation of a phase estimate map that is used in a correction step. This is done by fully sampling the central part of the *k*-space. For example, sampling may be limited to ~62 % of one of the encoding dimensions. The middle lines are used to estimate phase changes. Additionally, some techniques use filtering to reduce Gibbs'(5) ringing and/or reorder steps of reconstruction to reduce artefacts due to imperfections in the phase map estimation.

Partial Fourier techniques are limited to structural imaging that may exhibit slowly varying phase errors, as information encoded in a phase of signal is lost with these techniques. Also, it usually does not provide more than ~1.66x speed-up in acquisition time (data reduction: 40 %). Also, it is an undersampling technique and as such it results in loss of signal-to-noise ratio (SNR).

1.3.4 Non-uniform sampling trajectories

A faster way of data sampling is to use a different, more time efficient trajectory than Cartesian (i.e. multi-planar imaging(13)), spiral imaging(14, 15), radial imaging(16). One of the fastest sampling trajectories is spiral trajectory(17), as these cover a large proportion of k-space in one read-out, and make efficient use of the gradient hardware. Regrettably, DFT cannot be directly applied to data acquired on a spiral trajectory, as it is no longer placed on a uniformly spaced grid. To solve this problem a technique, originating from

astronomy(18), called *gridding*(19, 20) is applied. This re-samples data onto a rectilinear grid by convolving it with a kernel function;

$$\overrightarrow{\dot{S}_c}(\vec{k}) = \left(\vec{\dot{S}}(\vec{k}) * \boldsymbol{W}(\vec{k})\right) \text{Ш}_{\vec{N}}(\vec{k}) \qquad 1\text{-}20$$

Equation 1-20 Formulation of convolution onto rectilinear grid.

This allows use of DFT and the convolution operation is followed by FFT;

$$\begin{aligned} \mathcal{F}_{\vec{r}}^{-1}\left[\overrightarrow{\dot{S}_c}(\vec{k})\right] &= \mathcal{F}_{\vec{r}}^{-1}\left[\left(\vec{\dot{S}}(\vec{k}) * \boldsymbol{W}(\vec{k})\right) \text{Ш}_{\vec{N}}(\vec{k}) * \text{Ш}_{\overrightarrow{NM}}(\vec{k})\right] \\ &= \frac{1}{(N^2M)^m} \mathcal{F}_{\vec{r}}^{-1}\left[\vec{\dot{S}}(\vec{k})\right] w(\vec{r}) \text{Ш}_{\frac{\overrightarrow{1}}{NM}}(\vec{r}) * \text{Ш}_{\frac{\overrightarrow{1}}{N}}(\vec{r}) \end{aligned} \qquad 1\text{-}21$$

Equation 1-21 Inverse FT of convolved MR signal.

m is the number of dimensions. The final step is to remove weighting ($\mathcal{F}_{\vec{r}}^{-1}\left[\boldsymbol{W}(\vec{k})\right] = w(\vec{r})$) introduced by the convolution (Equation 1-16);

$$\frac{(N^2M)^m}{w(\vec{r})} \mathcal{F}_{\vec{r}}^{-1}\left[\overrightarrow{\dot{S}_c}(\vec{k})\right] = \mathcal{F}_{\vec{r}}^{-1}\left[\vec{\dot{S}}(\vec{k})\right] \text{Ш}_{\frac{\overrightarrow{1}}{NM}}(\vec{r}) * \text{Ш}_{\frac{\overrightarrow{1}}{N}}(\vec{r}) \qquad 1\text{-}22$$

Equation 1-22 MR reconstruction by *gridding*.

The final result has no aliasing resultant from DFT of the *gridded* signal ($\overrightarrow{\dot{S}_c}(\vec{k})$). However, it is important to note that *gridding* does not remove potential artefacts due to the selected trajectory *imperfections* (i.e. under-sampling). Also, if the selected trajectory function has a varying density of sampling points, this has to be corrected with an additional *k*-space samples weighting (21).

The *sinc* function is the optimal convolution kernel(19) as the resultant weighting due to convolution process has a form of scaling of the final result. However, *gridding* using *sinc* function is impractical as it is too computationally intensive. A number of other kernels with their reciprocal window functions have been studied and are commonly used in MRI(19, 21). These are functions of Finite Impulse Response (FIR), which rapidly decay outside of the selected kernel width, allowing its truncation and fast convolution process. In this work the Kaiser-Bessel function was selected and used for the gridding operations,

due to its characteristics, performance and simplicity in computation of discretized values.

1.3.5 Under-sampling

Some applications (i.e. Cardiac MRI) require a very high data acquisition rate to reduce blurring due to dynamic behaviour of the imaged object. In many cases this cannot be achieved with efficient sampling trajectories alone, and for which data reduction techniques like Partial Fourier are not applicable (i.e. Phase-Contrast MR) or would not make significant difference. The reduction in acquisition time can be realised with under-sampling. This means only a selected subset of trajectory read-out lines are acquired. This is a more general case than Partial Fourier acquisition, which is not limited to continuous sets of Cartesian lines. Any subset of trajectory positions can be selected. This operation is described in terms of under-sampling factor or data reduction factor, which is defined as a ratio of acquired trajectory positions to their total number. Alternatively, it can be described in terms of acceleration factor defined as a reciprocal of data reduction factor. For example, acquisition of only even or odd lines halves the sampling time (data reduction factor of 0.5 or 2x acceleration).

Unfortunately, with under-sampling some information about the signal is lost, which causes artefacts in the form of aliasing (Fig. 1-4). Consider a one dimensional signal $\vec{S}(k)$ (the same principal applies to multi-dimensional signals). The signal is fully sampled with intervals of $M\left[\frac{1}{m}\right]$ and $\vec{S}(k) \neq 0\ \forall k \in \left[-\frac{NM}{2}; \frac{NM}{2}\right]$;

$$\vec{\dot{S}}(k) = \left[\vec{S}(k)\text{Ш}_M(k)\right] * \text{Ш}_{NM}(k) \qquad 1\text{-}23$$

Equation 1-23 1D Cartesian sampling.

Now, inverse Fourier Transform of $\vec{\dot{S}}(k)$ is described as follows;

$$\dot{\rho}(r) = \frac{1}{NM^2}\left[\rho(r)\text{Ш}_{\frac{1}{NM}}(r)\right] * \text{Ш}_{\frac{1}{M}}(r) \qquad 1\text{-}24$$

Equation 1-24 Discrete nuclei density representation from 1D Cartesian sampling.

The sampling function $\text{Ш}_M(k)$ gives rise to periodicity of $\rho(r)$, which is sampled with intervals of $\frac{1}{NM}[m]$.

This explains aliasing for signals sampled with intervals wider than M. For example, acquisition of every other sample gives the following outcome;

$$\dot{\rho}(r) = \frac{1}{2NM^2}\left[\rho(r)\text{Ш}_{\frac{1}{NM}}(r)\right] * \text{Ш}_{\frac{1}{2M}}(r) \qquad 1\text{-}25$$

Equation 1-25 Two times under-sampling of 1D Cartesian sampling.

This means distances between centres of repeated $\rho(r)$ are halved, and result in aliasing (Fig. 1-3 and Fig. 1-4).

Fast imaging requires a compromise between data acquisition reduction that can be applied and visible artefacts in reconstructed images. This directly depends on the application and used sampling trajectory. For example, two times under-sampling of Cartesian trajectory in anterior to posterior direction for cardiac imaging may require no additional steps to remove incurred aliasing. This is because the elliptical shape of imaged cross-section leaves enough space for the alias. Although the outer parts of the body may alias, they do not wrap into the area of interest in the cardiac examination. An imaging plane can be positioned to place a heart's cross-section in the centre leaving it unobstructed by the alias. However, even two times sped-up Cartesian read-out is too slow for many MR applications. These include all real-time cardiac assessments for which data sampling need to be on the level of ~40 ms or faster. Of course, higher acceleration factors can be used, but then the aliasing may start to corrupt the whole imaging space.

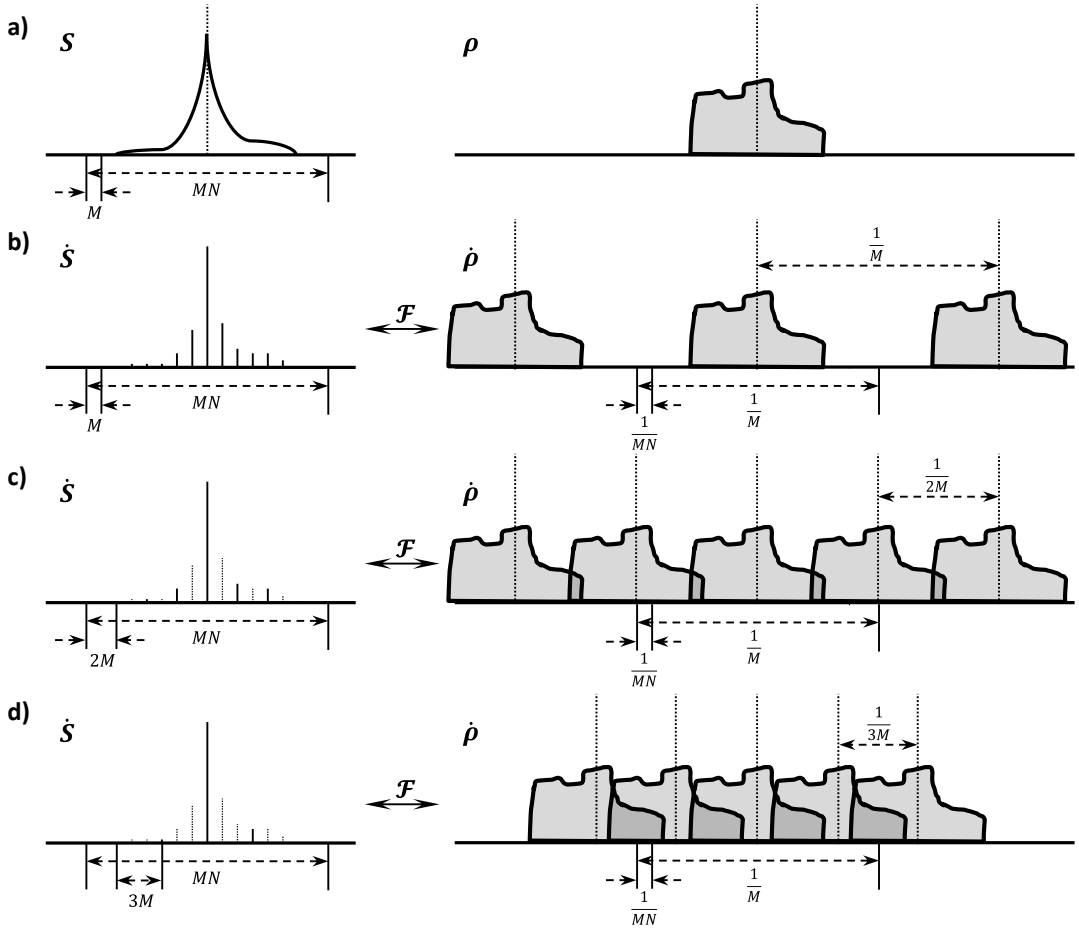

Fig. 1-3 Schematic visualization of1D signal under-sampling.

a) the *k*-space signal and the underlying object; b) discrete representation equivalent to fully sampled reconstruction; (c-d) progressive undersapling of the signal and its impact on the reconstructed object by inverse Fourier transform.

In practice this is not an undersampling technique as the region of interest (support region) is rectangular. Consequently, to preserve the spatial resolution an extent of sampled *k*-space is preserved, while a step between read-out lines is increased to reduce the field-of-view (FOV) in this direction. Nevertheless, implications of acquiring the rectangular FOV are the same as with equivalent undersampling (i.e. loss of SNR).

1.4 Advanced MRI

Under-sampling can be combined with efficient sampling trajectories (i.e. spiral trajectory) to achieve even lower acquisition times. Direct use of *gridding*, as a reconstruction technique, fails in this case. This would result in artefacts in the form of aliasing.

The artefacts can be removed with additional reconstruction steps. For example, reconstruction of under-sampled data through combination of simultaneously acquired data using spatially distributed receiver coils is a well-studied technique with multiple variants(22-25). Alternatively, a temporal domain of data acquired as a series of frames or volumes can be used to *encode* information about the acquired signal(26). In such case a single receiver coil is sufficient to reconstruct the data. However, it is possible to combine both techniques(27-29), which results in improved image quality or increase in possible data acquisition rate.

In this work, the Sensitivity Encoding (SENSE)(24) algorithm for arbitrary sampling trajectories was selected for implementation. Furthermore, the implemented algorithm was combined with the temporal encoding technique (UNFOLD) to double possible temporal resolution of acquired data. Both implementations were used in further described studies. For completion, the following describes the SENSE algorithm based on the original work(24, 27, 30) and its further studies(31), and the UNFOLD technique based on the original articles(26, 32, 33).

1.4.1 Simplified reconstruction by Sensitivity Encoding

To introduce basic concepts the simple case of under-sampled one dimensional signal is considered as introduced in section 1.3.5. This can be seen as an equivalent of under-sampling of the phase encoding dimension in Cartesian sequences (Fig. 1-4).

Deriving from equation 1-25, we can represent the discrete aliased function ($\dot{\rho}(r)$) as a sum of the underlying nuclei density ($\rho(r)$) from specific spatial locations;

$$\rho_i^+ = \rho\left(\frac{i}{MN}\right); \forall_{i\in\left[-\frac{N}{2};0\right)\cap\mathbb{N}}$$

$$\rho_i^- = \rho\left(\frac{i}{MN}\right); \forall_{i\in\left[0;\frac{N}{2}\right)\cap\mathbb{N}}$$

$$\dot{\rho}\left(\frac{i}{MN}\right) = \begin{cases} \dot{\rho}_i^- = \dfrac{\rho\left(\frac{i}{MN}\right) + \rho\left(\frac{i}{MN} + \frac{1}{2M}\right)}{2NM^2} = \dfrac{\rho_i^- + \rho_i^+}{2NM^2}; \forall_{i\in\left[-\frac{N}{2};0\right)\cap\mathbb{N}} \\ \dot{\rho}_i^+ = \dfrac{\rho\left(\frac{i}{MN}\right) + \rho\left(\frac{i}{MN} - \frac{1}{2M}\right)}{2NM^2} = \dfrac{\rho_i^+ + \rho_i^-}{2NM^2}; \forall_{i\in\left[0;\frac{N}{2}\right)\cap\mathbb{N}} \end{cases} \quad \text{1-26}$$

Equation 1-26 Simplified description of 1D aliasing for two times under-sampling.

This formulation is for two times under-sampling. Of course, the formula can be adapted to higher acceleration factors, as long as the number of samples (N) is divisible by the acceleration factor. Also, this simple method is limited to acceleration factors from the set of natural numbers. Both conditions are imposed to guarantee that no gridding operations are necessary and all points are on the same Cartesian grid.

A direct result of two times under-sampling is reduction of a support region or periodicity of $\dot{\rho}(r)$ from every $\frac{1}{M}[m]$ intervals to $\frac{1}{2M}[m]$. In 2D reconstructions, this is referred to as reduction in field of view (FOV). Consequently, both halves of the original function are overlaid on top of each other in each half of the function resulting from the under-sampling (Fig. 1-4). In this form the original signal cannot be determined as the linear equation is under-determined (there are more unknowns than equations).

SENSE(24) compensates for loss of information, due to the under-sampling process, using information about spatial signal weighting, introduced through each of acquisition coils. In principal, the closer a precessing magnetic momentum is to a receiver, the stronger the induced signal is (Fig. 1-4). In this understanding a receiver has spatially varying signal sensitivity dependent on its design. Commonly, these are represented by coil sensitivity maps (sensitivity profiles). The coil sensitivity directly translates into weighting of acquired signals. This additional information can be used to solve the under-sampling problem. A new system of linear equations can be written;

$$C_{j,i}^{+} = C_j\left(\frac{i}{MN}\right); \forall_{i\in\left[-\frac{N}{2};0\right)\cap\mathbb{N}}; \forall_{j\in[0,J-1]\cap\mathbb{N}}$$

$$C_{j,i}^{-} = C_j\left(\frac{i}{MN}\right); \forall_{i\in\left[0;\frac{N}{2}\right)\cap\mathbb{N}}; \forall_{j\in[0,J-1]\cap\mathbb{N}} \qquad 1\text{-}27$$

$$\dot{\rho}_{j,i}^{-} = \dot{\rho}_{j,i}^{+} = \frac{C_{j,i}^{-}\rho_i^{-} + C_{j,i}^{+}\rho_i^{+}}{2NM^2}; \forall_{j\in[0,J-1]\cap\mathbb{N}}$$

Equation 1-27 Simplified description of 1D aliasing including receiver coil weighting.

Here C_j and $\dot{\rho}_j$ represent the spatial weighting introduced with the j^{th} coil and the reconstructed nuclei distribution on a basis of the acquired signal, respectively.

This can be represented in matrix notation;

$$\begin{bmatrix} C_{0,i}^{-} & C_{0,i}^{+} \\ C_{1,i}^{-} & C_{1,i}^{+} \\ \vdots & \vdots \\ C_{J-1,i}^{-} & C_{J-1,i}^{+} \end{bmatrix} \begin{bmatrix} \rho_i^{-} \\ \rho_i^{+} \end{bmatrix} = \begin{bmatrix} \dot{\rho}_{0,i}^{-} \\ \dot{\rho}_{1,i}^{-} \\ \vdots \\ \dot{\rho}_{J-1,i}^{-} \end{bmatrix} = \begin{bmatrix} \dot{\rho}_{0,i}^{+} \\ \dot{\rho}_{1,i}^{+} \\ \vdots \\ \dot{\rho}_{J-1,i}^{+} \end{bmatrix}$$

$$\begin{bmatrix} \rho_i^{-} \\ \rho_i^{+} \end{bmatrix} = \begin{bmatrix} C_{0,i}^{-} & C_{0,i}^{+} \\ C_{1,i}^{-} & C_{1,i}^{+} \\ \vdots & \vdots \\ C_{J-1,i}^{-} & C_{J-1,i}^{+} \end{bmatrix}^{-1} \begin{bmatrix} \dot{\rho}_{0,i}^{-} \\ \dot{\rho}_{1,i}^{-} \\ \vdots \\ \dot{\rho}_{J-1,i}^{-} \end{bmatrix} = \begin{bmatrix} C_{0,i}^{-} & C_{0,i}^{+} \\ C_{1,i}^{-} & C_{1,i}^{+} \\ \vdots & \vdots \\ C_{J-1,i}^{-} & C_{J-1,i}^{+} \end{bmatrix}^{-1} \begin{bmatrix} \dot{\rho}_{0,i}^{+} \\ \dot{\rho}_{1,i}^{+} \\ \vdots \\ \dot{\rho}_{J-1,i}^{+} \end{bmatrix} \qquad 1\text{-}28$$

Equation 1-28 Simple 1D SENSE matrix notation.

This system of linear equations can be created and solved for each of the special positions ($i \in \left[-\frac{N}{2};\frac{N}{2}\right) \cap \mathbb{N}$). The systems are solvable providing the special weights introduced with each coil are not correlated and each imaged spatial position (i) is *seen* by a number of receivers equal or greater to the used acceleration factor.

The SENSE technique works well providing each of the used acquisition coils contributes unique information about the whole imaged object. For this reason phased array coils (or surface coils) are used. These are arrays of smaller coils that can acquire signal simultaneously and independently of each other(34). They provide higher signal-to-noise ratio (SNR), although their signal sensitivity is more localised to close surrounding of the coil and quickly decays with distance. This is opposite to bigger volume coils (i.e. the body coil).

Pre-determination of coil sensitivities is not a trivial task as these can vary depending on experiment conditions. Additionally, the exact position of a coil with respect to an imaged object coordinates is needed. Instead, the coil sensitivities can be estimated on the basis of acquired images themselves(24);

$$C_j(\vec{r}) = \frac{\dot{\rho}_j(\vec{r})}{P(\vec{r})} \qquad 1\text{-}29$$

Equation 1-29 Simple estimation of the coil sensitivities for the SENSE algorithm.

The estimation requires a reference image ($P(\vec{r})$). Optimally, the reference should be perfectly homogeneous; meaning it should not introduce any spatially varying weight to the *underlying* nuclei density ($P(\vec{r}) = b(\vec{r})\rho(\vec{r})$); for example a body coil image. However, it is not possible to simultaneously acquire data with both body and surface coils. Sequential acquisition may result in artefacts due to imaged object motion. Alternatively, the combination of acquired images from multiple surface coils with the *sum-of-squares*(34) can be used as a reference for the estimation;

$$sos = \sqrt{\sum_j \dot{\rho}_j(\vec{r})\dot{\rho}_j^*(\vec{r})} \qquad 1\text{-}30$$

Equation 1-30 Simplified *sum-of-squares* technique for combination of images from multiple phased array coils.

This can be done during a pre-scan for non-dynamic imaging. A fully sampled data set can be acquired for the calculation of the coil sensitivities. These are then used in following accelerated acquisitions. For dynamic objects, for which the profiles can change with time, they can be calculated from the accelerated data themselves. Assuming the sensitivity profiles are slowly varying in space, fully sampling of the central portion of k-space should be sufficient for the calculations. However, this is with an expense of the total acceleration and consequently limits the temporal resolution of the acquired data.

Alternatively, a *sliding window*(27) approach can be applied in acquisitions of series of frames. Rotating between supplementary trajectories, while acquiring data, allows combination of the data into a fully sampled set.

The new combined data can serve as a fully sampled reference for calculation of the coil sensitivities, although it has a lower temporal resolution. Also, in long continuous real-time scans the *sliding window* approach can be used to make the reconstruction resistant to motion.

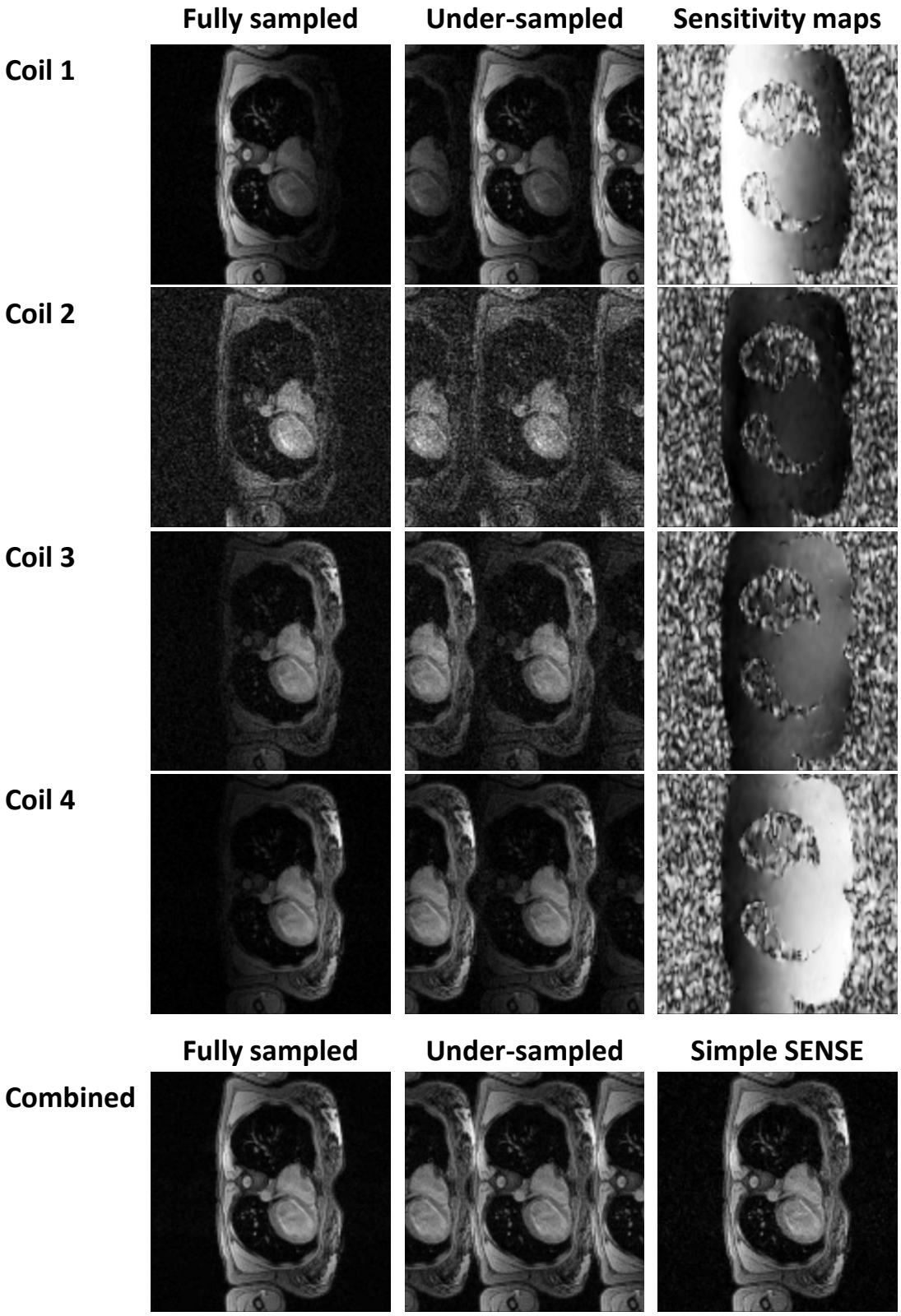

Fig. 1-4 Simple SENSE for uniform Cartesian under-sampling.

Fully sampled, two times under-sampled and sensitivity maps for data acquired with four surface coils are presented in the first four rows. The under-sampled data was created artificially out of the fully sampled data by zero-filling of every other read-out line. The sensitivity maps were calculated according to equation 1-29 using the fully sampled data with the *sum-of-squares* (Equation 1-30) as a reference image. The last row presents combined images (the *sum-of-squares*) for the fully- and under-sampled data sets, and the result of the simple SENSE reconstruction (Equation 1-28).

1.4.2 Sensitivity Encoding algorithm

The formulation of the SENSE algorithm for the under-sampled arbitrary trajectories has a more complex formulation(24, 30). Consequently, it is a more difficult problem to solve, due to the non-uniform sampling pattern of the trajectories.

k-space signal function is reformulated to take in account multiple coils acquisition;

$$\vec{S}_j(\vec{k}) \propto \int \rho(\vec{r}) C_j(\vec{r}) e^{-i2\pi\vec{k}\cdot\vec{r}} d\vec{r} \qquad 1\text{-}31$$

Equation 1-31 MR signal formulation including spatial distribution of a receiver coil.

$C_j(\vec{r})$ - represents spatial weighting introduced by j^{th} receiver coil. The signal is acquired on a sampling trajectory as a discrete function ($\vec{S}(\vec{k})$), which can be seen in a form of the matrix. For purpose of this discussion the following matrix notation is adopted to represent discrete functions;

$$\dot{v}(\vec{a}); \vec{a} \in \boldsymbol{A} = [\vec{a}_0 \quad \vec{a}_1 \quad \cdots \quad \vec{a}_{N-1}] \xrightarrow{\substack{matrix \\ notation}} v_N = \begin{bmatrix} \dot{v}(\vec{a}_0) \\ \dot{v}(\vec{a}_1) \\ \vdots \\ \dot{v}(\vec{a}_{N-1}) \end{bmatrix}; v_n = \dot{v}(\vec{a}_n) \qquad 1\text{-}32$$

Equation 1-32 Matrix notation of discrete signals.

Following this notation a set of $\vec{S}_j(\vec{k})$ can be seen as $\vec{S}_{JM}$. It represents k-space data acquired for a set of $[0; J-1] \cap \mathbb{N}$ coils on $\vec{k} \in [\vec{k}_0 \quad \vec{k}_1 \quad \cdots \quad \vec{k}_{M-1}]$ trajectory samples. J and M are numbers of elements in the set of receiver coils and trajectory points respectively.

The MR imaging can be expressed in the form of matrix notation as a system of linear equations;

$$\rho_N = F_{N,(JM)} \vec{S}_{(JM)} \qquad 1\text{-}33$$

Equation 1-33 MRI experiment described as a system of linear equations (I).

Here, ρ_N is a vector of N pixels/voxels representing the final image/volume. As described by (24), the reconstruction matrix F has to meet the following condition;

$$FE = Id \tag{1-34}$$

Equation 1-34 Identity condition for the SENSE algorithm.

Where Id is identity matrix and E denotes encoding matrix defined as;

$$E_{(j,m),n} = C_j(\vec{r}_n)e^{2\pi i\vec{k}_m\cdot\vec{r}_n} \tag{1-35}$$

Equation 1-35 Encoding matrix definition.

This can be seen in two forms of continuous functions, which were sampled onto columns or rows of the matrix;

$$E_{J,N}(\vec{k}); E_{j,n}(\vec{k}) = \int C_j(\vec{r})e^{2\pi i\vec{k}\cdot\vec{r}}\delta(\vec{r} - \vec{r}_n)d\vec{r} \tag{1-36}$$

Equation 1-36 Encoding matrix rows definition.

$$E_{J,M}(\vec{r}); E_{j,m}(\vec{r}) = C_j(\vec{r})\int e^{2\pi i\vec{k}\cdot\vec{r}}\delta(\vec{k} - \vec{k}_m)d\vec{k} \tag{1-37}$$

Equation 1-37 Encoding matrix columns definition.

The system of linear equations can be now formulated as;

$$E_{(JM),N}\rho_N = \vec{S}_{(JM)} \tag{1-38}$$

Equation 1-38 MRI experiment described as a system of linear equations (II).

Direct matrix inversion of E is not preferable, as the matrix is very large (order of ~10^9 elements). Instead, a solution is found with an iterative process of conjugate-gradient linear solver algorithm (30).

To prevent noise amplification with higher number of iterations, a regularisation formulation is added(35-37);

$$\begin{bmatrix} E\rho \\ \lambda L(\rho - \rho^0) \end{bmatrix} = \begin{bmatrix} \vec{S} \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} E \\ \lambda L \end{bmatrix} b = \begin{bmatrix} \vec{S} - E\rho^0 \\ 0 \end{bmatrix} \Leftrightarrow b = \rho - \rho^0 \tag{1-39}$$

Equation 1-39 Regularisation formulation.

Here ρ^0, L and λ are prior-information about the solution, linear transformation matrix and regularisation factor respectively. In simple words,

regularisation is a way of constraining a solution and is used to prevent noise amplifications with a higher number of solver's iterations. $\lambda L(\rho - \rho^0) = 0$ is a set of constraining equations that when added as a part of the linear equation system force the solution to be a trade-off between the prior knowledge (ρ^0) and the solution to $E\rho = \vec{S}$ equations. The regularisation can itself be a source of artefacts and the regularisation factor (λ) is used to control the constraint *strength*. Optimal selection of the regularisation factor depends on the problem. Commonly an empirical approach is used to find a suitable value for a certain group of problems. The L matrix can be seen as a selection filter that determines which parts of the solution and in what ratio should take part in the regularisation. The simplest regularisation uses $L = Id$ and is known as Tikhonov regularisation(36). In this form the regularisation is equally applied to all of the reconstructed image points with no correlation between them. Although the L matrix can be selected arbitrarily, the optimal selection has to minimise sensitivity to noise. This can be expressed as an inverse of the expected signal intensity on a diagonal; as compared to Tikhonov regularisation. This way assigning higher regularisation *strength* to regions with no signal and reducing it in regions where an imaged object is expected.

To speed up the convergence process, preconditioning in the form of intensity (I) and density (D) correction (Section 1.3.4) matrices are added;

$$\begin{bmatrix} DEII^{-1} \\ \lambda DLII^{-1} \end{bmatrix} b = \begin{bmatrix} D\left(\vec{S} - E\rho^0\right) \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} DEI \\ \lambda DLI \end{bmatrix} g = \begin{bmatrix} D\left(\vec{S} - E\rho^0\right) \\ 0 \end{bmatrix} \Leftrightarrow g = I^{-1}b \qquad \text{1-40}$$

Equation 1-40 Intensity and density corrections.

Simplifying, preconditioning is a way of restricting a solution process to *concentrate* only on selected regions. Excluding regions of low SNR, by assigning a smaller coefficient to them, improves the iterative process. As opposed to the regularisation matrix the preconditioning matrix (I) should reflect the expected signal intensity.

This can be represented in matrix notation;

$$Ag = B \Leftrightarrow A = \begin{bmatrix} DEI \\ \lambda DLI \end{bmatrix}; B = \begin{bmatrix} D\left(\vec{S} - E\rho^0\right) \\ 0 \end{bmatrix} \quad \text{1-41}$$

Equation 1-41 Compact formulation of the linear encoding system.

To ensure convergence for a conjugate-gradient linear solver, a system of equations has to be described by a positive-semidefinite matrix. Left multiplication by conjugate-transpose of A guarantees this criterion;

$$\begin{gathered} Ag = B \\ A^H Ag = A^H B \\ g = (A^H A)^{-1} A^H B \end{gathered} \quad \text{1-42}$$

Equation 1-42 Solution to the linear encoding system.

The final formulation has the following form;

$$\begin{gathered} I^{-1}(\rho - \rho^0) = \left(\begin{bmatrix} DEI \\ \lambda DLI \end{bmatrix}^H \begin{bmatrix} DEI \\ \lambda DLI \end{bmatrix}\right)^{-1} \begin{bmatrix} DEI \\ \lambda DLI \end{bmatrix}^H \begin{bmatrix} D\left(\vec{S} - E\rho^0\right) \\ 0 \end{bmatrix} \\ I^{-1}(\rho - \rho^0) = (IE^H D^2 EI + \lambda^2 IL^H D^2 LI)^{-1} IE^H D^2 \left(\vec{S} - E\rho^0\right) \\ I^{-1}\rho = (IE^H D^2 EI + \lambda^2 I\theta^{-1} I)^{-1} IE^H D^2 \vec{S} \Leftrightarrow \rho^0 = 0; \theta^{-1} = L^H D^2 L \end{gathered} \quad \text{1-43}$$

Equation 1-43 The final linear equations system solved by the SENSE algorithm.

Matrices D, I and L are diagonal of real values. Therefore, their conjugate-transpose is equal to themselves.

Equation 1-43 describes how every step of the iteration process is carried out. Regrettably, even in this form it requires multiplications of the encoding matrix ($E_{(JM),N}$) and its conjugate-transpose with some vectors x_N and y_{JM} respectively. Again, the size of the encoding matrix and vectors makes it a very computationally challenging task. Closer look reveals that each of these operations can be replaced with Fourier Transformation and FFT algorithm can be applied for acceleration. Using equation 1-36 the first operation can be broken down in a following way;

$$E_{(j,M),N}x_N = E_{j,N}(\vec{k})x_n$$

$$(Ex)_j(\vec{k}) = \sum_{n=0}^{N-1} x_n \int C_j(\vec{r})e^{2\pi i\vec{k}\cdot\vec{r}}\delta(\vec{r}-\vec{r}_n)d\vec{r}$$

$$(Ex)_j(\vec{k}) = \int \sum_{n=0}^{N-1} \left(x_n C_j(\vec{r})\delta(\vec{r}-\vec{r}_n)\right) e^{2\pi i\vec{k}\cdot\vec{r}}d\vec{r} \qquad \text{1-44}$$

$$(Ex)_j(\vec{k}) = \int X_j(\vec{r})e^{2\pi i\vec{k}\cdot\vec{r}}d\vec{r}$$

$$(Ex)_j(\vec{k}) = \mathcal{F}_{\vec{k}}^{-1}\left[X_j(\vec{r})\right]$$

Equation 1-44 Spatial domain to *k*-space domain transformation steps.

The $E_{(JM),N}$ matrix transforms $X_j(\vec{r})$ - weighted with C_j coil signal x sampled on $\vec{r}_N$ positions; into *k*-space domain, by taking its inverse FT. Similarly, the equation 1-37 is used to break down the second multiplication;

$$E^H{}_{N,(JM)}y_{JM} = E^H{}_{J,M}(\vec{r})y_{JM}$$

$$(E^H y)(\vec{r}) = \sum_{j=0}^{J-1}\sum_{m=0}^{M-1} C_j^H(\vec{r})y_{(j,m)} \int e^{-2\pi i\vec{k}\cdot\vec{r}}\delta(\vec{k}-\vec{k}_m)d\vec{k}$$

$$(E^H y)(\vec{r}) = \sum_{j=0}^{J-1} C_j^H(\vec{r}) \int \sum_{m=0}^{M-1} y_{(j,m)}\delta(\vec{k}-\vec{k}_m)\, e^{-2\pi i\vec{k}\cdot\vec{r}}d\vec{k} \qquad \text{1-45}$$

$$(E^H y)(\vec{r}) = \sum_{j=0}^{J-1} C_j^H(\vec{r}) \int Y_j(\vec{k})e^{-2\pi i\vec{k}\cdot\vec{r}}d\vec{k}$$

$$(E^H y)(\vec{r}) = \sum_{j=0}^{J-1} C_j^H(\vec{r})\mathcal{F}_{\vec{r}}\left[Y_j(\vec{k})\right]$$

Equation 1-45 *k*-space domain to spatial domain transformation steps.

The $E^H{}_{N,(JM)}$ matrix transforms $Y_j(\vec{k})$ - k-space signals y sampled on $\vec{k}_M$ trajectory positions from J coils; into spatial domain. The k-space signals are Fourier Transformed and multiplied with $C_j^H(\vec{r})$. The resultant products are added-up to create the final result of the operation.

The results are presented as continuous functions, but in practice FFT is used to perform FT and inverse-FT. Voxel/pixel positions $\vec{r}_N$ are located on uniformly spaced grid, which directly allows use of FFT. Conversely, positions of

k-space samples ($\vec{k}_M$) are on an arbitrary trajectory (i.e. spiral, radial), thus they do not have to meet this condition. Therefore, additional gridding operations are needed; these follow and precede inverse-FT and FT respectively.

Described here is the SENSE technique for arbitrary trajectories, which removes aliasing, caused by data under-sampling. A conjugate gradient linear solver is used, which in an iterative way finds an artefact free solution. The iterative nature of this reconstruction process makes it a time consuming procedure, which requires significant computational power. This is a simplified description of the SENSE algorithm for under-sampled arbitrary trajectories. A comprehensive description can be found in the original papers(24, 27, 30) as well as in descriptions of further work on the technique(31).

1.4.3 Temporal encoding

It is a common practice in cardiac MR assessments to acquire a series of the same volume or slice data. This is an occasion to encode temporal information into acquired signal, which can be used to improve image quality and/or remove artefacts arising from under-sampling.

One such technique is Unaliasing by Fourier-Encoding the Overlaps Using the Temporal Dimension (UNFOLD)(26). This method is based on linearity of integration;

$$\int_a^b \big(mf(x) + ng(x)\big)dx = m\int_a^b f(x)dx + n\int_a^b g(x)dx \qquad 1\text{-}46$$

Equation 1-46 Linearity property of integration.

Integral of a sum of functions can be replaced with a sum of integrals of these functions. Consequently, this property also applies to Fourier Transformation;

$$\mathcal{F}_y[mf(x) + ng(x)] = m\mathcal{F}_y[f(x)] + n\mathcal{F}_y[g(x)] \qquad 1\text{-}47$$

Equation 1-47 Linearity property of Fourier Transform.

This is a very useful property, as it means that adding together reconstructed signals is equivalent to the reconstruction of sum of these signals. T frames of the same signal acquired separately with T complementary

sampling trajectories can be combined to create a single, alias free frame. Although it is far from being the optimal solution to aliasing caused by under-sampling, it shows that temporal domain can be used to encode information.

Again, consider a one dimensional example (Section 1.3.5) of k-space signal $\vec{S}(k)$ (Equation 1-23 and Equation 1-24) and its two times under-sampled case (Equation 1-25). The under-sampling caused reduction ($\frac{1}{M} \rightarrow \frac{1}{2M}$) in distance between centres of repeated $\dot{\rho}(r)$ resulting in potential overlap. To create a complementary trajectory to the under-sampled trajectory, the previously used one is shifted by one sampling interval (M). This results in swap from acquiring even samples to odd samples. It can be formulated as follows;

$$\begin{gathered}
\dot{\vec{S}}(k) = \left[\vec{S}(k)\text{Ш}_{2M}(k-M)\right] * \text{Ш}_{NM}(k) \\
\dot{\rho}(r) = \frac{1}{2NM^2}\left[\rho(r)\text{Ш}_{\frac{1}{NM}}(r)\right] * e^{2\pi iMr}\text{Ш}_{\frac{1}{2M}}(r) \\
\dot{\rho}(r) = \frac{1}{2NM^2}\left[\rho(r)\text{Ш}_{\frac{1}{NM}}(r)\right] * \sum_{j\in\mathbb{Z}}\left(cos(j\pi) + isin(j\pi)\right) \\
\forall_{j\in\mathbb{Z}}[cos(j\pi) + isin(j\pi)] \in \{-1, 1\}
\end{gathered} \qquad \text{1-48}$$

Equation 1-48 Result of reconstruction of data on a shifted under-sampled trajectory.

The shift causes every other alias to change its sign from + to -. Consequently, acquiring a series of frames with alternating trajectories results in oscillations of $\dot{\rho}(r)$ through the time domain (t-space). The overlaps are encoded into temporal domain in form of oscillations, as long as the used trajectories are complementary. A general case can be formulated as follow;

$$\dot{\rho}(r) = \frac{1}{HNM^2}\left[\rho(r)\text{Ш}_{\frac{1}{NM}}(r)\right] * e^{2\pi iLMr}\text{Ш}_{\frac{1}{HM}}(r) \qquad \text{1-49}$$

Equation 1-49 General formulation of the impact of trajectory shift on aliases due to under-sampling.

H represents acceleration factor and determines the distance between samples on an under-sampled trajectory. Also, it defines a number of overlaps in each r position. L is the position of the first sample or a shift's distance in number of samples. Both coefficients can be of real positive values, but in this

case a gridding operation must be used to allow use of DFT. The oscillating signal in each r can be represented as;

$$\dot{\rho}(r_a, t) = \sum_{j=0}^{\lceil H \rceil - 1} \dot{\rho}\left(r_a - j\frac{M}{H}, t\right) e^{2\pi i t \frac{j}{H}} \quad \text{1-50}$$

Equation 1-50 Temporally varying oscillations in under-sampled real space.

$\dot{\rho}(r_a, t)$ represents the desired, alias-free signal at a position r_a for a $t \in \mathbb{N}$ acquisition frame. Modulated with $e^{\frac{2\pi i t j}{H}} \forall_{j \in \{0,1,\ldots,H-1\}}$ signals are accumulated across r positions. As it is a periodic function, L (assuming $L \in \mathbb{N}$ and $H \in \mathbb{N}$) was replaced with t - the index of an acquired frame. Fourier Transformed through time $\dot{\rho}(r, t)$ has H peaks at $2\pi \frac{j}{H} \forall_{j \in \{0,1,\ldots,H-1\}}$ temporal frequencies, including a DC component related to the true signal (similarly as in Fig. 1-3). The components can be removed with application of low-pass filter. Positions of each peak are defined by $T\frac{j}{H} \forall_{j \in \{0,1,\ldots,H-1\}}$, where T is the number of frames used in Fourier Transformation.

Although this technique can be used for dynamic objects, it is limited to slowly varying, continuous signals. Wide temporal frequency bandwidth of desired alias-free signal can result in overlap between temporal frequency peaks and prevent or impair artefact removal process(32, 38, 39).

Temporal encoding can be used in conjunction with SENSE algorithm (29, 33, 40). It provides a way of creating coil sensitivity maps from acquired data itself (T-SENSE) (27). If incorporated as an additional reconstruction step, it could allow even higher accelerations (29).

1.5 General Purpose computing on Graphic Processing Units

The following sections (Section 1.5 and 1.6) are based on the publicly available on-line material as well as the documentation provided by NVIDIA, Santa Clara, CA, USA. This is a brief description to familiarise the reader with basic terms and concepts. More comprehensive description is available at the original sources (41-43).Permission to reproduce Fig. 1-5, Fig. 1-6 from the original sources (as indicated) has been granted by NVIDIA.

Graphic Processing Units (GPU) as dedicated vector processors can easily achieve a much higher processing speed than Central Processing Units (CPU) for applications that are based on matrix/vector operations (i.e. signal/video processing, statistics, computer simulations and modelling). The term vector processor, describes processing units dedicated for simultaneous execution of the same operation across multiple data sets (single instruction multiple data architecture, SIMD). A core of these processors is an array of multiple, tightly coupled arithmetic units (also called cores). These are usually less specialised than presently available CPU cores, but their strength lies in their number. Presently, desktop high-end GPU cards have up to 2880 cores (i.e. NVIDIA GeForce GTX TITAN Black), whilst the newest desktop CPUs expose scalability of 6 cores (12 simultaneously executed threads) per processor (i.e. Intel Core i7-4960X).

Rapid growth of the computer game industry, has forced hardware producers to create faster, more sophisticated and easier to program GPUs. With time, they have reached a level of performance exciding one of CPUs. The term General Purpose computing on Graphic Processing Units (GPGPU) was coined by Mark Harris in 2002(41), and relates to the use of GPU cards for general calculations. The idea is simple; identify parts of algorithm that can be executed in parallel and run them on a GPU. Unfortunately, early GPUs with their roots deep in computer graphics and fixed processing pipe-line were not easy to use for other applications. Growing interests in GPGPU and demands for a simple, more general way of programing resulted in transformation of GPUs from dedicated chips for graphics into freely programmable co-processors.

While CPUs have not shown much improvement in floating point arithmetic performance over the last few years, GPUs have kept continuously improving. Recently, the peak computational performance excided 4.5 and 1.3 TFlops (10^{12} floating-point operations per second) for single and double precision operations respectively (NVIDIA GeForce GTX TITAN Black). This left a huge gap between GPU and CPU, which only recently started to close up with commercialisation of Intel's Many Integrated Core (MIC) architecture with Xeon Phi series. The newest Intel Xeon Phi 7120P/D/X (44) has 61 physical cores

capable of running at theoretical peak speed of 2.4 TFlops single and 1.2 TFlops double precision (45).

The peak computational power of GPUs is not always fully achievable, but porting computationally intense parts of algorithms onto the GPU has the potential to greatly increase the speed in comparison to serial code. While developing code for any parallel system, one has to bear in mind the conclusion from the Amdahl's law,

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}} \qquad \lim_{n\to\infty} \frac{1}{(1-P) + \frac{P}{n}} = \frac{1}{(1-P)} \iff \forall_{n\in\mathbb{N};P\in[0,1]} \qquad \text{1-51}$$

Equation 1-51 Achievable speed-up with parallelization according to the Amdahl's law.

The Equaition 1-51 describes how much speed-up (S) can be achieved using n processing units. It is impossible to have code that can be infinitely parallelized. There is always a part that has to be executed in serial manner. This ratio is represented with P which is the proportion of execution that can be run in parallel. The direct conclusion from the equation is that the maximum achievable speed-up is not determined solely by the number of used processing units (n). It is rather an intrinsic property of the algorithm. As n increases to infinity the speed-up *saturates* at the inverse of $(1-P)$. Only algorithms that exhibit very low values of $(1-P)$ can fully benefit from constantly increasing number of processing units. These are referred to as *embarrassingly parallel* problems in parallel computing. These are computational problems that need no or very little effort to split the problem into multiple of parallel tasks. This happens when there is no dependency between the tasks, and hence no communications and/or synchronisations are required between them.

GPUs were designed with these *embarrassingly parallel* problems in mind. Their increasing number of cores executing the same operation on different data guarantees achieving the maximum possible speed-up dictated with the Amdahl's law. However, often it means that a fast CPU version of an algorithm (designed for this architecture) performs very poorly on the GPU.

Consequently, the most effort needs to be concentrated on exposing as much of parallelism of the algorithm while porting onto the GPU platform.

CPU and GPU were developed for different tasks (Fig. 1-5). CPU can quickly cope with purely serial code, since most of its architecture is dedicated to control unit blocks. On the other hand GPU can be seen as a computing engine, since most of its architecture consists of arithmetic unit blocks. It is clear that CPU and GPU do not cope with some tasks as well as their counterpart, and to achieve high performance they should be used in conjunction with each other. However, to achieve this access to the GPUs computational potential needed to be simplified. Namely, the necessary execution scheduling and control flow overweight must be reduced to its absolute minimum as it reduces the P parameter (Equation 1-51). With the release of the NVIDIA's Compute Unified Device Architecture (CUDA), the whole structure of GPU (of this manufacturer) was redesigned to create a new programming interface through which developers could easily access and utilise its computing power. It was a step to create a heterogeneous system in connection with a CPU as execution control unit.

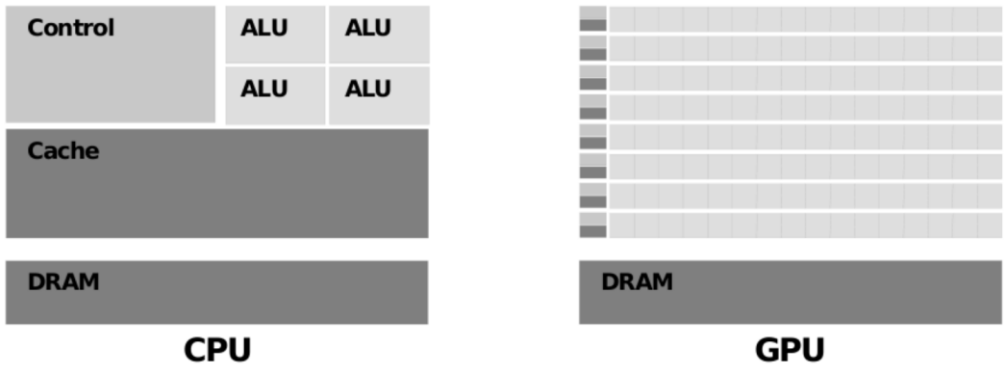

Fig. 1-5 Comparison of CPU and GPU architectures (44).

The CUDA technology is not the only existing GPU enabled programming platform. For example, OpenCL (Open Computing Language, Khronos Group) is a programming platform for software run across heterogeneous systems. As a non-vendor specific platform it simplifies development of code for different hardware architectures. These mostly include mobile devices but are not limited to them.

I chose the CUDA technology as a programing platform, as it provides a very rich and mature code development ecosystem. Also, the technology is continuously developed and improved with the simplicity of programming in mind. The CUDA toolkit comes with the compiler, dedicated debugger, profiler and other tools. A growing community of users ensures continuity in support from other developers and variety of shared code examples. Additionally, the toolkit comes with a set of libraries dedicated to speed-up the most commonly used scientific operations (linear algebra, Fourier Transformations, digital image and signal processing). These are continuously maintained by NVIDIA to ensure their constant optimisation and adoption to new hardware releases.

1.6 CUDA programming model

CUDA introduced the heterogeneous programming model in which code is executed on the *host* - a system equipped with one or more CPUs; and a *device* connected to it – a CUDA enabled GPU card. A CUDA program is written in C/C++ language with some language extensions to introduce GPU specific data types and functions called *kernels*. This way the whole program can be written as a single source code. The code is separated automatically during the compilation process. The GPU specific instructions are extracted and compiled by the CUDA compiler, while the rest is processed by a C/C++ compiler. The final version is then linked together into a single executable file.

Most commonly the CUDA programs adhere to the following execution pattern. First the *host* organises data in a way in which most of the data-parallelism is exposed and partitions it between threads. Next, *device* memory is allocated for *kernel* arguments. The data are sent to the *device* and *kernels* necessary for the algorithm executions are launched. *Kernel executions* are always asynchronous. This means that after scheduling *kernel* calls, the *host* is free to run other calculations. The state of scheduled work can be checked through synchronisation functions. Also, the *host's* execution can be halted until the *device* has finished. This allows simultaneous execution of work on the *host* and *device*. An adopted strategy depends on the application. The successful execution is usually followed with transferring the results into *host's* memory for

analysis and potential further processing. All of the steps are repeated until the whole algorithm has been finished.

To achieve high computational throughput, the *kernel* runs must use thousands of threads. Threads are organised in blocks, which then are arranged into a grid (Fig. 1-6). All *kernel* launches are supplied with a declaration of the size of a block of threads and how many blocks in a grid must be run to complete processing. Blocks and threads are indexed to allow identification of the part of data the thread has to work on.

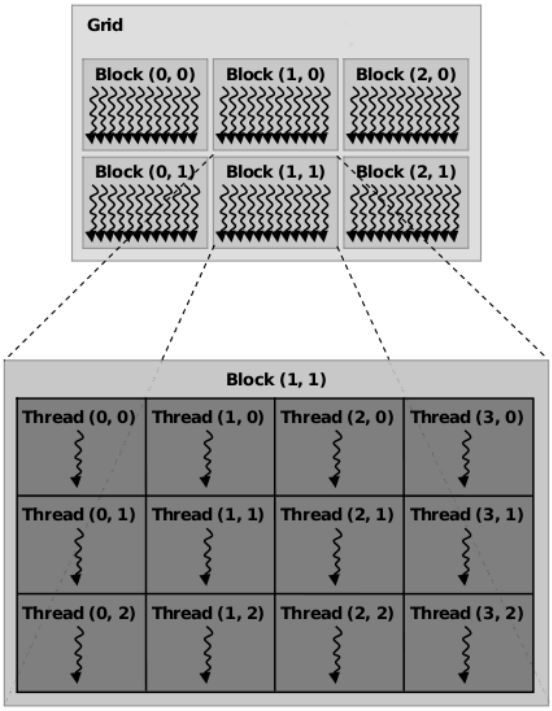

Fig. 1-6 Thread hierarchy (44).

Each *device* contains one or more multiprocessors. Blocks of threads are mapped to the multiprocessors. To facilitate scheduling and management of thousands of threads that run concurrently, the threads are executed in groups of 32 called *warps*. GPU threads are very lightweight in comparison to a CPU one; there is a very little creation and no switching overhead. Each thread has its own set of resources, and once started the *warp* is executed until its work is finished. How many concurrent *warps* are run on a multiprocessor depends on the *kernel's* demands on resources. All threads of a *warp* execute the same instruction, which is referred to as Single Instruction Multiple Threads (SIMT) execution. In the case of conditional branches, if threads of a *warp* do not agree

on a path, the execution is serialised by the multiprocessor. This scheduling simplicity is required to achieve high performance by devoting more clock cycles to actual computation (43).

In the CUDA programming model the most important performance consideration is to make sure that all global memory accesses are coalesced, or as many as possible. Namely, the global memory is accessible to all threads that run on a *device*. The global memory as an off-chip memory is relatively slow. Access to it introduces 400 to 600 clock cycles latency (42). If a sufficient number of *warps* run on a multiprocessor the latency can be hidden by executing the *warps* that are ready while the others are waiting for the data. Accesses to the global memory are done in transactions of 32-, 64- or 128-byte length segments (42). How many transactions are issued depends on the access pattern used by the requesting *warp*. The requirements vary depending on the type of *device*. In general accessed memory has to be a continuous array of elements aligned to the size of the segment. If these requirements are not met the access is performed in sequential manner one for each requested memory element.

The secondary performance consideration is the minimisation of global memory access through efficient use of the *shared* on-chip memory. This memory is divided between thread blocks that presently run on the multiprocessor. Access to this memory is very fast and allows rapid exchange of data between threads of one block. If all threads of a block have to access the same part of global memory it is much more efficient to first load the data to the *shared* memory and then allow the threads to work on it from there. In this way the *shared* memory becomes a software managed *cache* saving the global memory bandwidth. *Cache* is a term used in programming referring to low latency memory that is used as a buffer to speed-up access to data, by preserving most recently used data for potential further access.

One more important performance issue is connected to the adopted SIMT execution model. Divergence within a *warp* is highly undesirable. As mentioned every branching path will increase time of execution. Consequently, condition statements should be based on thread indices to avoid execution serialisation.

2. Motivation

The work presented here was developed for the vascular imaging and physics group at the UCL Institute of Cardiovascular Science. The institute aims to be a world class centre for advancing the field of cardiovascular medicine. The work concentrates on improvement of detection and management of cardiac diseases through the use and development of multimodality imaging techniques. The vascular imaging and physics group's role is to provide state-of-art MR techniques for clinical cardiac MR and improve patient assessment by translation of these advanced methods into the clinical environment.

Cardiac MR is challenging as it requires high temporal resolutions to provide accurate information (uncorrupted with motion blurring images) about the highly dynamic behaviour of the cardiovascular system. This work focuses on flow quantifications (i.e. assessments of blood flow velocities) using phase-contrast MR (PCMR) (46). These types of assessments require repeated *k*-space read-outs for each of the encoded velocity directions, which have a direct impact on the temporal resolution of acquired data. In these cases efficient sampling trajectories (Section 1.3.4) and data under-sampling (Section 1.3.5) have to be combined to allow very high data sampling rate. A good example is a fast real-time spiral PCMR sequence designed and developed by our group (17, 47, 48).

However, as previously indicated the high acquisition rate comes with the expense of reconstruction algorithm complexity. Complex MRI reconstruction algorithms like the SENSE reconstruction (Section 1.4.2) require substantial amount of computational power to keep the processing time within acceptable limits. Secondary to the trade-off between fast acquisition and complexity of reconstruction, is an increase in number of receiver coils. For sufficient artefact suppression it is required to have more independent coils than an acceleration factor, which is especially true for very high accelerations. Higher number of coils increases spatial information of an imaged object which is used to compensate for the under-sampling, although at the same time it increases the size of data that needs to be processed.

In the case of progressive under-sampling, small gains in temporal resolution are quickly outmatched by increasingly longer reconstruction times.

All of these shift the bottleneck of the MR examinations from data acquisitions to their reconstructions and limit the utility of these complex MR techniques within the clinical environment. One extreme example is continuous real-time cardiac MR assessment.

High-performance implementations are required to remove the reconstruction time limitation. This can be realised by exploiting intrinsic parallelisms of the algorithms. As discussed (Section 1.4.2), the MRI reconstruction algorithms are based on solving the set of linear equations (Equations 1-33, 1-38) that represent the process of imaging. Consequently, the whole process can be broken down into matrix operations. Matrix operations are highly parallelisable as there is no data dependency. Algorithms with very little or no data dependency scale well and significantly improve performance when run on higher number of processing units. Theoretically, knowing the size of a problem means that the processing time can be made as short as it is needed, by increasing the number of processing units. This is assuming the time required for the parallel execution preparation (partitioning, scheduling and result collection times) is negligible and only if a problem can be infinitely partitioned into smaller ones. Practically, this is never attainable as there are no problems that can be infinitely partitioned, nor is it possible to have an infinite number of processing units. Consequently, concurrent versions of MR algorithms are limited in implementation and performance by the hardware they are run on.

A common way of providing a multitude of processing units is by organising computers (processors) into *clusters*. A cluster is a set of computing units connected together by one or multiple communication channels capable of working together on a specific problem. As clusters of sufficient number of computers may not be easily attainable for all clinical, research facilities another more commonly available solution is needed.

Graphic Processing Units (GPU) as vector processors were designed to leverage applications relying on matrix operations. They consist of multiple arithmetic units (cores) tightly packed on a single die and capable of simultaneously executing the same instruction on a vector of data elements. The growing computer games industry demands constant improvements of

GPUs and their further development as capable co-processors. In consequence the GPUs are increasingly inexpensive while simultaneously becoming more powerful and simpler to use for general purpose programming.

Fast GPU implementations of MRI reconstructions do exist and a significant improvement in performance as compared to the CPU reconstructions has been previously reported (49-51). However, GPU reconstructions are mostly done in an off-line mode. This means data are downloaded from the scanner and are reconstructed elsewhere, which constitutes a serious drawback to the clinical workflow. To truly facilitate use of the advanced MRI reconstructions, such developments must be incorporated into an online scanner reconstruction pipeline. This is vital in order to improve the overall efficiency of the clinical workflow and enable rapid viewing of the images to check for data integrity prior to finishing the MR examination. In this paradigm, proper data transmission management can be as important as reconstruction's efficient implementation. Time improvements gained on the reconstruction side can be easily counterbalanced with slow transmission processes or lost on unnecessary synchronizations.

The work presented here was not only motivated with the need to provide fast, flexible image reconstruction for computationally intensive reconstructions, but most of all to make them feasible within a busy clinical service. This can be summarised in form of two challenges that needed to be addressed; i) use of advanced MR sequences is limited by their reconstruction time, and ii) GPU implementations exist but run in off-line mode. One possible solution is to create a dedicated exchange protocol over a network that connects the scanner and the external machine. However, this fixed solution is limiting as it would need to be redone for each new application. Thus to not be tied to a fixed configuration, the new image reconstruction system was defined as a distributed system. Active components of a distributed system (i.e. applications running on a scanner or external computer) can be flexibly changed and rearranged. This meant additional system components like new clients for different types of scanners or different reconstructions could be quickly introduced. This way the defined system, not only serves as a leverage providing the fast seamless reconstruction process, but also becomes a

scalable platform for translation of advanced MRI algorithms into the clinical environment.

Integration and scalability were key aspects of succeeding in the project. They had to provide the basic building blocks for development of future MR applications hosted in the heterogeneous distributed image reconstruction system. Thus the implementation was divided into two steps; i) networking with remote execution, and ii) implementation of the reconstruction algorithm. In the course of this work I:

- networked an external computer equipped with a GPU card into the scanner's native image reconstruction system;
- designed and developed the data transmission and remote execution protocol for efficient management of continuous streams of real-time MR data;
- implemented and optimized a GPU based SENSE reconstruction for data acquired on arbitrary trajectories;
- demonstrated the impact and improvement in the type of assessment protocols that can be enabled for patient management with the new reconstruction system;
- presented flexibility of the created system by re-using the created components to combine the SENSE reconstruction with temporal encoding technique (UNFOLDed-SENSE) and adapt the remote reconstruction to accelerated retrospective gating sequences. Both were done without changes to previously implemented data transmission and execution management steps;

3. Distributed image reconstruction system

In the chapter, I describe my work on development of the novel on-line image reconstruction system for clinical/research MRI. The work resulted in creation of the processing framework allowing the transparent integration of the external hardware into the scanner system. The work was published in the following article:

> *Implementation of a generalized heterogeneous image reconstruction system for clinical magnetic resonance*, GT Kowalik, JA Steeden and V Muthurangu; Concurrency and Computation: Practice and Experience (Special Issue) Volume 27, Issue 6, pages 1603–1611, 25 April 2015 (DOI: 10.1002/cpe.3349);
>
> *Appendix 11.6*

and the proceeding of 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013;

> *Implementation of a Heterogeneous Image Reconstruction System for Clinical Magnetic Resonance*, GT Kowalik, JA Steeden, D Atkinson, A Taylor, V Muthurangu; Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science Volume 8384, 2014, pp 469-479;
>
> *Appendix 11.7*

3.1 Introduction

To develop an implementation of a networking framework, its underlying components and functionalities need to be clearly stated and defined. The desired system at its core must provide bidirectional data exchange between the scanner's application and an external application. However, only the scanner's side was expected to make remote execution requests. In this simplistic form, a fixed architecture with a single remote application, a dedicated link and a predefined communication protocol might serve well for the purpose of a single project. However, as much as it benefits from simplicity, this arrangement lacks in flexibility and scalability. It would require re-implementations with each new image reconstruction algorithm or scanning protocol. To avoid this, additional requirements were made; the framework must provide flexibility in extending the system with more scanners and include multiple remote reconstructions. Additionally, asynchronous remote execution calls are needed to allow overlapping of local and remote processes.

I adopted the distributed system architecture in order to fulfil all of the requirements. I distinguished two system components; *servers* – the processes providing image reconstruction functionality, and *clients* – the processes that control connection, data exchange and remote calls. These by definition, are not linked to any specific physical location, but are meant to be a virtual pool of resources not limited to a single connection or arrangement. The distributed system promotes flexible organisation of *clients* and *servers* into separate reconstruction systems. Each system can contain multiple *servers* residing on the same/different machines, as well as *clients* making requests to the same/different *servers*. This way the built system was defined to be a distributed image reconstruction system, based on client-server architecture.

3.2 Client-server architecture

Portability across different hardware and programming technologies is crucial to the envisaged system, as it must allow independent development of each side of the system. Developments of different client side implementations for each MR system cannot have an impact on the server or networking side of

the application. Similarly, technology providing data transmission, remote execution and the way in which the server manages the reconstruction process, need to expose the same modularity. This allows substitution of these components without affecting the rest of the system. For this reason the system framework (Fig. 3-1) was organised into three layers; the Networking layer, Server-reconstruction layer and Client-reconstruction layer.

3.2.1 Networking layer

System integration is achieved by separating the clients and servers with the networking layer. The networking layer provides a single type of networking object. This defines a set of data transmission and remote execution interfaces for the client-server architecture. These must allow bidirectional exchange of data, parameters and identification of connected objects. Allowing only a single generic type of communication simplifies communication between server and client objects. Also, it ensures compatibility when replacing networked objects with different or newer versions.

To reduce the development time the networking layer was based on the Common Object Request Broker Architecture (CORBA) technology. CORBA is a specification of interoperable, multi-platform network based and language independent objects. The specification is managed by the Object Management Group (OMG, Needham, MA, USA). It uses Independent Definition Language (IDL (52)) to define network object interfaces, which can be mapped to different programing languages (i.e. C/C++, Java, Python, etc.). Also, CORBA provides a naming server; this can be seen as an equivalent of Domain Name System (DNS) in the internet. With its help, the physical address of distributed objects is mapped onto easily memorable names. These are publicly available through the naming service. This allows objects to move between different physical locations without making an impact on the system configuration.

The networking layer does not depend on any specific implementation of CORBA. Multiple implementations of CORBA can be used as long as they comply with the standard. Nevertheless, CORBA could be replaced with other technology (i.e. DCOM/COM+), providing that the necessary interfaces and functionalities are supplied.

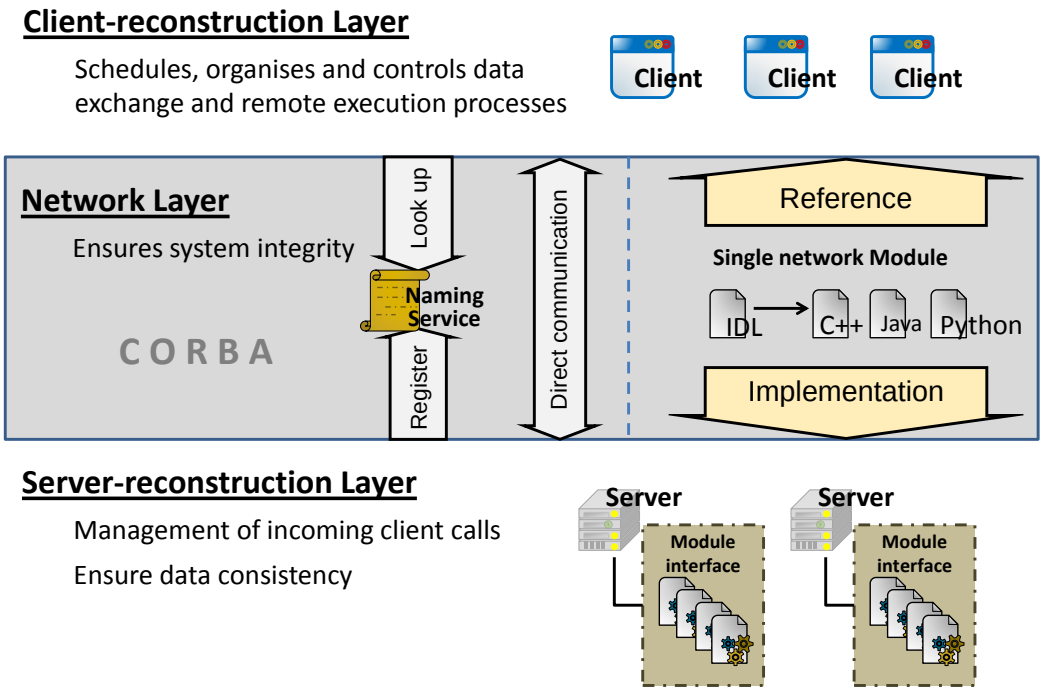

Fig. 3-1 Layered framework for the client-server architecture of the distributed image reconstruction system.

3.2.2 Server-reconstruction layer

The server-reconstruction layer implements a specific functionality to the interfaces provided by the networking layer. Within this layer, different reconstruction algorithms can be implemented. In a simple form these can be realised as a dedicated server object providing a single functionality. However, this would force a one-to-one relationship between a client and a server, rendering it inefficient. It is more desirable to have servers capable of facilitating multiple reconstructions for the same or different clients. In this case, the server's main role is to ensure data consistency and control flow management. To realise this, the reconstructions are defined as objects of specific types. These are defined and stored in the form of modules or libraries. The modules are accessible through a module interface built into the server layer (Fig. 3-1). This approach allows a single server to consist of multiple different processing modules that can be loaded on the client's request. Again, to promote simplicity a single module interface per server is preferred. Of course, different servers can be prepared to access different types of reconstruction modules.

By encapsulating reconstructions, a server can create separate reconstruction objects for each client. This way it is freed to concentrate on

management of incoming calls, which in distributed computing are not guaranteed to come in a serial manner. Networking interface client IDs can be assigned to identify objects which should be used to execute a remote request. This ensures data consistency in the case of connection with multiple clients. However, a single client can make multiple simultaneous requests through different interfaces. In some cases this may be desirable (i.e. transmission of new data may overlap with retrieval of previous results). Thus it was decided that it is best to leave the decision about the synchronisation of access to client's objects, up to the particular server specification.

3.2.3 Client-reconstruction layer

The client-reconstruction layer schedules, organises and controls data exchange and remote execution processes, employing interfaces provided by the networking layer. In the most trivial case only a single remote execution is needed. Sequential calls to the server layer can be encapsulated in a single execution unit (thread). In this case an overlap between stages constituting the reconstruction is undesired and synchronous remote execution is sufficient. This occurs when the required remote execution constitutes a whole undividable step of an algorithm and the next steps are dependent on its result.

In a more general case, it is desirable to allow simultaneous work on local, as well as remote data. Additionally, asynchronous remote calls may be needed to allow overlap between remote execution calls. Unfortunately, in CORBA all remote calls are synchronous and are treated as if they were standard local function calls. This can be seen as a limitation of the technology. However, it promotes the developer discretion in selecting an execution parallelization technique. Also, it makes the potential replacement of CORBA with a different networking middleware easier, as it is only responsible for providing correct data transmission and remote execution, rather than additional local control flow management which may vary from application to application.

3.3 Application life cycle

The new image reconstruction as a distributed application, based on the described client-server architecture was determined to have two system states: the system set-up (Fig. 3-2) and the reconstruction state (Fig. 3-3).

3.3.1 The system set-up state

The system set-up state is maintained by the naming server. This naming service contains a record of servers that registered themselves as available to clients. A desired system instantiation is created *ad hoc* by a client searching for, and connecting to, servers providing the required functionality. The system is destroyed by the client disconnecting from the servers, however, the servers' applications remain awaiting new connections.

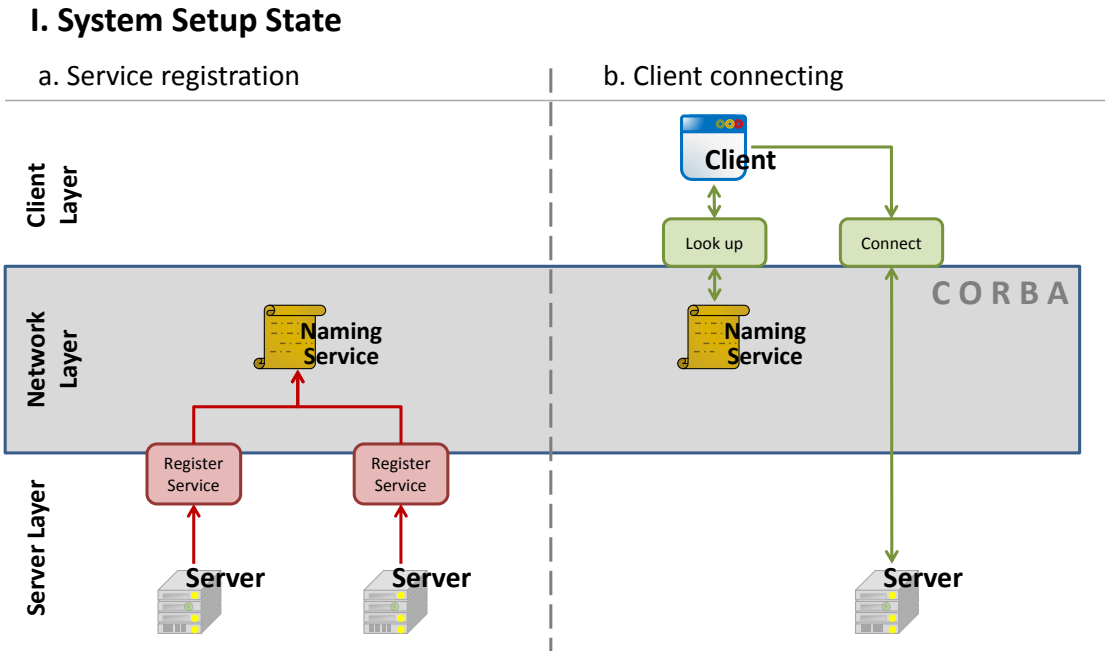

Fig. 3-2 The system set-up state.
An overview of tasks necessary for setting-up an instantiation of the system.

3.3.2 The reconstruction state

The reconstruction state contains four stages (Fig. 3-3); initialization, data transmission, remote execution and result collection. In the initialization stage a client sends the required parameters to the server in order for it to create the necessary data structures. This process is normally done once per reconstruction, since different repetitions usually have the same conditions.

After initialisation, the client invokes the remaining reconstruction stages in a desired order until the whole reconstruction task is done.

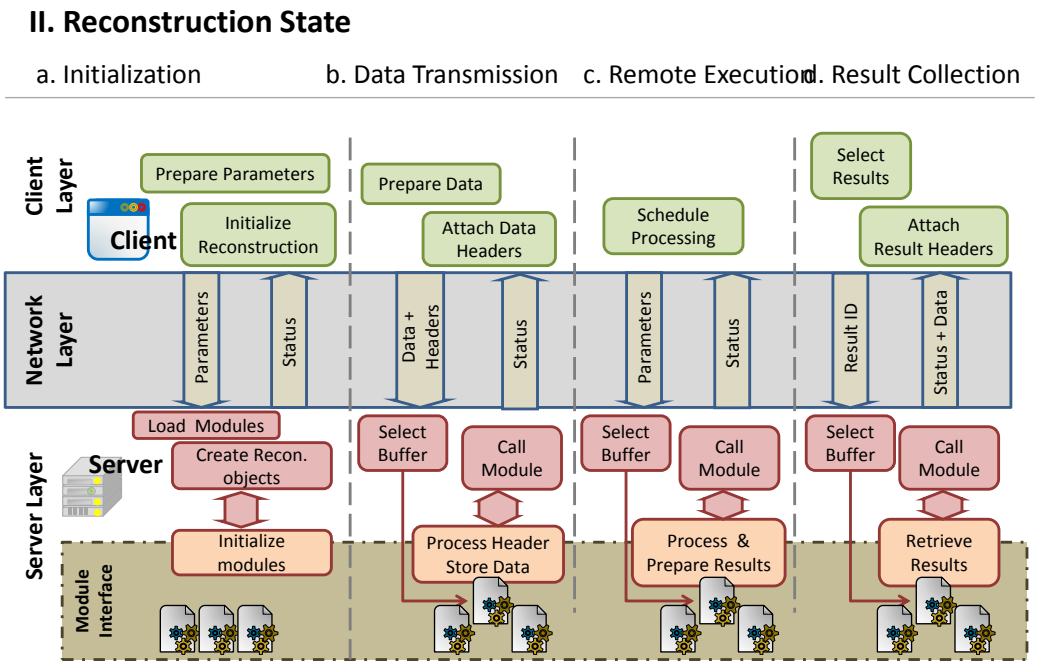

Fig. 3-3 The reconstruction state.

An overview of tasks assigned to each of the independent stages of the application state.

3.4 The implemented system specifics

The proposed client-server architecture does not force a specific order of the aforementioned reconstruction stages. However, for real-time applications proper data transmission management can be as important as efficient implementation of the image reconstruction algorithm. Time improvements gained by efficient reconstruction can be counterbalanced by a slow transmission process or lost on unnecessary synchronizations. For optimal processing of continuous and arbitrary length streams of real-time data, an overlap between data transmission and reconstruction is desired. This can be achieved by buffering of the incoming data and assigning different processing threads to each of the communication and execution stages. The optimal situation is when the reconstruction time is equal or shorter than data transmission. In this case only one additional storage space is required to allow a constant stream of data between computers. However, in the case of

reconstruction being slower than data transmission, it may be beneficial to have more than two buffers.

The same idea behind the encapsulation into different reconstruction module objects is used to facilitate the buffering. A client may order that more than one object of a specific reconstruction type is created for its needs. On the server's side, the buffers assigned to each of the clients may be explicitly separated from each other using an additional indexing structure. This is not necessary as it is the client's role to identify a reconstruction object upon which each action should be carried out; through a unique identifier. Nevertheless, the distinction between objects representing buffers and those for different clients is made. Namely, there is no need to allow communication between reconstructions' objects unless they constitute a part of a bigger coherent reconstruction process, as represented with buffers. The reconstruction of continuous stream of real-time data may require data to be shared between consecutive buffered reconstructions (i.e. to allow a sliding window reconstruction approach or to share coil sensitivity profiles). To enable this, the implemented module interface was extended to take in account possible communication between objects of the same type.

A more detailed description of each aspect of the implemented data transmission and remote execution management for the continuous real-time MR assessments with the described distributed imager reconstruction system is presented in the following sub-sections.

3.4.1 Networking and communication interfaces

C++ implementation of CORBA technology (omniORB, Apasphere Ltd, Cambridge, United Kingdom (53)) was used to implement the networking layer. A definition of the network communication module in IDL is presented in Appendix 11.1. This was used to generate C++ version of the interfaces. The single type of networking object was defined with five networking interfaces. Four directly relating to each of the reconstruction stages described above (Fig. 3-3) and one enabling the system set-up state (Fig. 3-2). Each interface provides the client with a set of input parameters that can be used by a client to identify a remote reconstruction object (data structures, buffers) as well as to

specify a variant of operation, if necessary. Other parameters are used to transmit an arbitrary length of data. All of the interfaces return the status of a requested operation. This is returned after an external execution has finished and it depends on the server's implementation.

3.4.2 Execution and data transmission management

The scanner's native reconstruction system provided a C/C++ based, multi-threaded programming environment for the implementation of the client side of the system. Fig. 3-4 presents the implementation of a client for an incoming stream of real-time data. The whole task of maintaining the reconstruction process was left to the client. That means, the server side does not actively process the data. Each separate step (data transmissions and executions) must be implicitly scheduled and overlooked by the client side. The whole process is controlled by three cross-network groups of threads; *Send threads*, *Process threads* and *Get threads*. Each group of threads controls the processing of different aspects of the reconstruction state, enabling overlapping of data transmission and execution. On the client, these are represented by three control blocks, which work independently from one another, communicating only by passing messages about the completion of the previous stage. The stream of constantly acquired data is divided into sets, which can fit into buffers organised on the external machine. The buffers integrity is protected by a set of locks shared between the control blocks. This mechanism was adopted to prevent overwriting of data currently being reconstructed with newly incoming data. The number of buffers is an arbitrary parameter that is set during the reconstruction initialization stage.

Send threads control the preparation of data for reconstruction. The client separates, labels the data and initiates the sending process. Equivalent threads on the server side store the transferred data in an appropriate format for reconstruction, within a selected buffer. The incoming data is first pre-stored by the client's send control block to avoid transmission of small chunks of data. The transmission takes place if the storage limits are reached or when the last line of data in a set was received. The send control block is responsible to check the status of the buffer's lock, and only transmit data if the buffer is unlocked. When the buffer is filled-up, the send control block locks the buffer

and changes the index of the receiving buffer. This way transmission can continue using a different buffer.

Process threads are responsible for overlooking each side of the remote execution. The client signals readiness for processing by passing an index for the newly filled buffer, to the server. Corresponding threads on the server start the reconstruction and return its status upon completion. If the reconstruction is successful the process control block passes the index of the buffer to the collect control block.

Get threads maintain the process of collecting results. The client sends the index of a result to the server for translation into its specific data storage system. A result is returned if processing for the selected buffer has finished. Next, the client marks the buffer as unlocked and the retrieved data are sent further down the scanner's system for processing, storing or presentation.

This organization of overlapping transmission and remote execution can work smoothly with no interruptions or breaks, providing transmission and reconstruction are faster than the data acquisition.

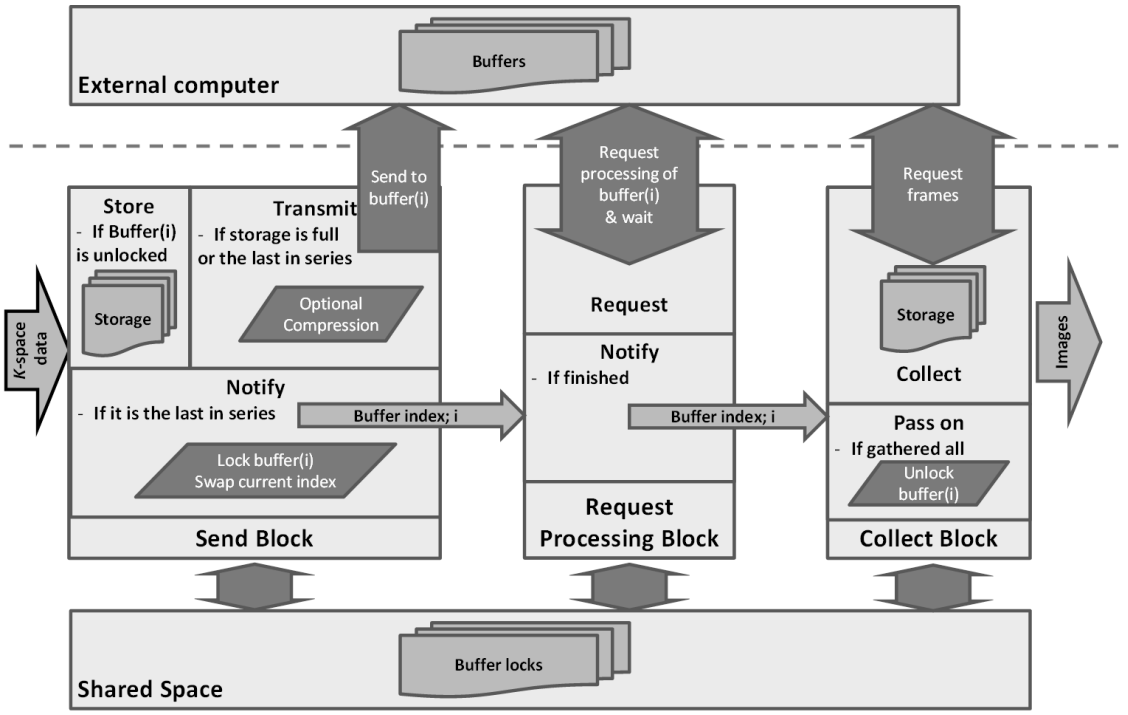

Fig. 3-4 The buffered transmission and remote execution management.

3.4.3 Reconstruction management

A single reconstruction module interface (as presented in appendix 11.2) was designed independently of the server application. This declares entry points through which the server application controls each created reconstruction object. A new reconstruction module must provide a single reconstruction class definition that derives from the declared interface (abstract class *IReconstruction* - Appendix 11.2). This class represents a processing algorithm implemented with the module. As no prior knowledge could be assumed, the module implementation must provide adequate functionality to each of the global functions through which the module is initialised after being loaded (*StartLibrary*), de-initialised prior to being un-loaded (*StopLibrary*) and each new object of the reconstruction is created (*GetIReconstruction*). Although, the module interfaces are declared with continuous real-time data processing in mind, they are not restricted to them. Apart from the necessary functionalities; setting up a new reconstruction (*ReadHeader*), copying data into an object (*SetData*), running reconstruction steps (*PreProcess*, *Process*, *PostProcess*) and retrieving results (*GetResSize*, *GetResultData*), it provides entry points for duplicating of already initialised objects (*CopyBuffer*) and general communication between a group of objects (*ManageBuffers*).

The server application constitutes an execution controller whose main role is to protect data consistency of each of the reconstruction objects. The implemented version was based on a modular approach dedicated for buffered reconstruction. Specifically, it was prepared to handle only a single reconstruction module that was used to generate multiple reconstruction buffer objects. This implementation is limiting in situations where multiple scanners are connected into a system, but perfectly sufficient for the MR hardware, which is presently available to us (a single scanner connected with an external computer).

The server was prepared to efficiently handle continuous real-time data acquisitions. However, it is not limited to this type of acquisitions. On connection and initialisation, the server loads the requested reconstruction module. Next, a single reconstruction object is created and initialised using the incoming reconstruction parameters. The initialised object is then replicated to form the

number of requested buffers. These are identified with an index sent by the client with each request. Also, a set of semaphores and mutex objects (these are special types of variables through which an access to a resource can be managed between multiple concurrently running threads) were created to control access to the buffers. Simultaneous data transmission and processing requests on the same object were forbidden as it would make no sense to allow retrieving of results (or updating in-put data) before processing finished.

Additionally, the server implementation enabled data exchange between buffers. This was done to facilitate potential sliding window type reconstructions. On the reconstruction request, the server first calls the communication interface and passes to it all of the buffers and then calls the processing interface of the selected buffer.

3.5 Data transmission test

The described system was developed and implemented within research/clinical environment (UCL Centre for Cardiovascular Imaging, Institute of Cardiovascular Science, London, United Kingdom). The original installation connected a native reconstruction hardware (2x Intel Xeon E5440 2.8 GHz, 16 GB DDR3) of 1.5 Tesla (T) MR scanner (Avanto, Siemens Medical Solutions, Erlangen, Germany) using a half-duplex Ethernet with an external computer (DELL Alienware Aurora, Intel i7-920 2.7 GHz, 9 GB DDR3) equipped with a GPU card (NVIDIA GeForce, GTX 480 1.4 GHz, 1.5 GB DDR5). The original system was replicated on another 1.5 T Siemens scanner at our institution. Also, the distributed image reconstruction system was installed at collaborating institutions. These include mirror installations at the Heart Hospital, London, United Kingdom, University Hospital Southampton, Southampton, United Kingdom and Yale Magnetic Resonance Research Center, New Haven, United States of America.

As mentioned in the motivation (Chapter 2), in the clinical/research cardiac MR environment there is a high demand for rapid assessment of the cardiovascular system by MRI. The PCMR spiral sequence combined with parallel imaging (SENSE), developed in the unit allows a very fast acquisition of data (17). This allows real-time assessment of biomarkers during active

exercise (17, 54), as well as the response to mental stress (55). In addition, this created and validated sequence constituted a base for more advanced imaging techniques (i.e. prospectively cardiac gated PCMR (48), retrospectively cardiac gated PCMR (56) and high resolution Fourier Velocity Encoding (57)). However the original reconstruction proved very time consuming. This limited its use for research/clinical studies to short acquisitions (3-7 s) due to resulting long reconstruction times (up to 3-5 minutes). The new system was designed to remove this limitation and enable continuous real-time data acquisitions which are unconstrained by the reconstruction time.

For optimal processing of a stream of real-time data an overlap between data acquisition and its reconstruction is required. One of consequences of the external processing is the requirement for data to be transferred onto the external machine. In this case the optimal processing requires both; the reconstruction and transmission times, to be faster than the acquisition. Consequently, it was essential to ensure the data transmission and the new reconstructions would not constitute bottlenecks for the future assessment protocols.

The most important, for this part of the work, was to find the maximum network transmission bandwidth which consequently is the maximum acquisition bandwidth supported by the system. First the optimal size of transmission packets that would fully utilise the network capacity was estimated.

To determine the network transmission performance depending on the size of transferred data, a simple application was implemented within the distributed system. The client generated data sets with increasing size, which were then transferred onto the external machine. The server's role was to receive the data and store it in its memory. The time necessary for each transmission, calculated as the time spent to execute the network transmission interface call (*SetRawData*), was measured on the client side. The transmitted data size ranged from 2048 bytes (B) to 2.1 MB (Fig. 3-5). The tests allowed assessment of the impact of data fragmentation on the transmission performance. The maximum network bandwidth was found to be ~43.4 MB/s. It was estimated all transmissions of ~500 kB or more were achieving 90-100 % efficiency.

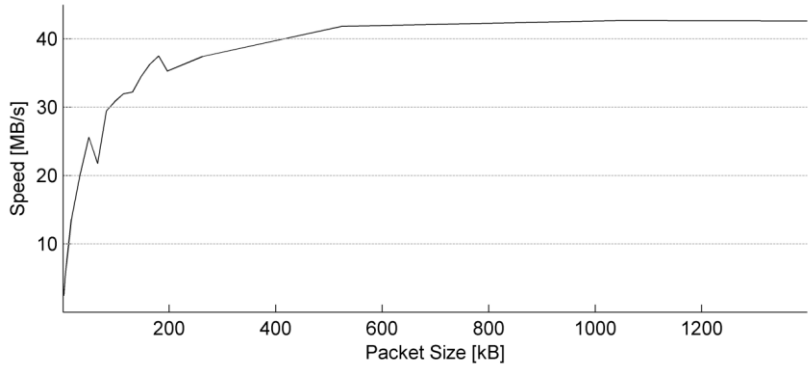

Fig. 3-5 Network transmission speed as a function of transmitted data size.

The results are from the half-duplex Ethernet connection between the native scanner image reconstructor and the external computer.

The acquisition throughput depends on the acquisition trajectory and the number of receiver coils. The Cartesian and radial trajectories can be seen as the least demanding, since only a single k-space line is acquired with each RF-excitation. Assuming a high resolution data; 256x256 matrix twice oversampled in the read-out direction acquired with repetition time (TR): 2.5-5 ms and 12 receiver coils would produce 9.8-19.7 MB/s. More efficient trajectories can produce significantly more data. Typical spiral acquisition; 128x128 matrix would need TR: ~6.92 ms and generate ~32 MB/s (assuming 12 spiral interleaves to fully sample k-space with 2300 read-out points per interleave). These spiral acquisition parameters were used in the following tests.

To validate the initial transmission results the final transmission protocol for the continuous real-time acquisitions (Section 3.4.2) was tested. A set of 60 flow frames (total of 120 data frames) were acquired with four times acceleration. This resulted in total acquisition time: ~2.49 s and data size: ~79.5 MB. The results are presented in Fig. 3-6. The transmission time was measured from the beginning of the transmission until the last of the frames was fully transferred on the external computer. The optimal transmission protocol, defined as the one introducing the shortest latency and providing faster total transmission time than acquisition time, was found for the transmission package size of 12 acquisition read-outs that were sent together as a single network transmission (the transmission package); latency: ~76 ms and total

transmission time: ~2.41 s resulting in ~35 MB/s. The lower throughput, as compared to the simple application's results, was caused by measuring the total time rather than the individual package transmissions. This included collection time of packages, which was dictated by the acquisition speed.

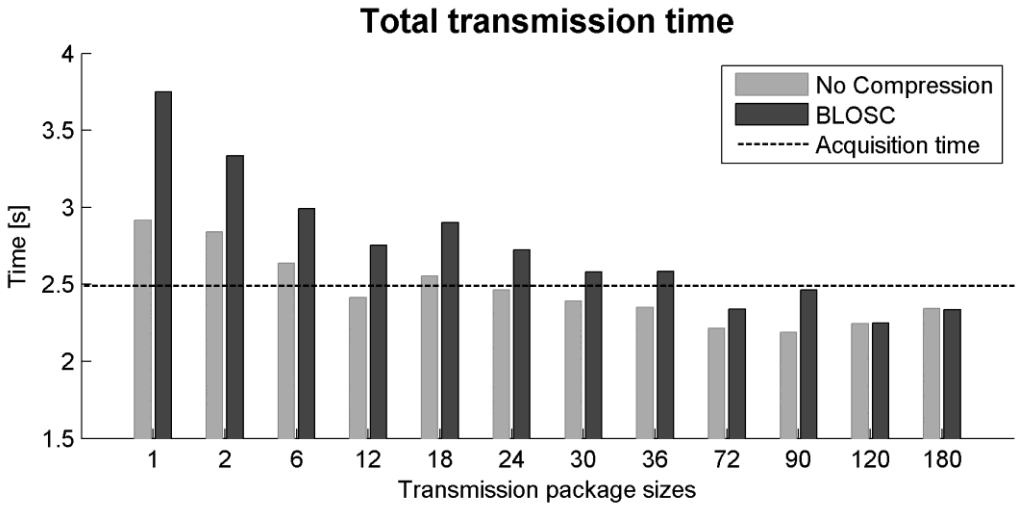

Fig. 3-6 Total transmission time test results.

Experimentally, I looked into data compression as some imaging protocols may produce more data per frame. For example, higher resolution spiral read-outs; 192x192 matrix, TR: ~8.28 ms, would generate: ~44 MB/s (assuming 16 spiral interleaves with 3812 read-out points per interleave and 12 receiver coils). This already exceeds the limits of the implemented system and needed to be addressed. The tests were repeated, this time applying data compression prior to a transmission. In the scope of this work, only one compression algorithm was tested; BLOSC - A blocking, shuffling and lossless compression library (58). Unfortunately, the tests showed that the achievable compression rate (~93 %) was not sufficient and better alternatives have to be sought. Low compression rate made it insufficient to make up for time required to run the compression, which added-up to the total transmission time. Possibly, reorganization of transmission stages to hide this extra time may mitigate this problem. However, more data intensive applications (i.e. 3D imaging) may require much higher compression rates to keep a transmission time below an acquisition time.

4. GPU reconstruction implementation

In this chapter, I present the initial implementation of the GPU based SENSE algorithm. This implementation was further improved and adopted in the further described projects. The work presented in the chapter was published in the following article:

> *Real-time flow with fast GPU reconstruction for continuous assessment of cardiac output*, GT Kowalik, JA Steeden, B Pandya, F Odille, D Atkinson, A Taylor and V Muthurangu; Journal of Magnetic Resonance Imaging, Volume 36, Issue 6, pages 1477–1482, December 2012 (DOI: 10.1002/jmri.23736).
>
> *Appendix 11.8*

4.1 Introduction

The networking framework described above enables fast transmission of data onto an external computer. In this chapter I will describe a GPU implementation of the SENSE algorithm. This implementation was meant to counterbalance the reconstruction time limitations incurred with the use of highly accelerated fast acquisition trajectories.

In cardiac MRI, cine data frames are acquired using either real-time or segmented k-space acquisitions. The segmented approach differs only in the organisation of data rather than the reconstruction process. This work concentrates on analysis of the real-time case, as this represents the more general example and is more demanding with respects to the time limitations.

A simple real-time imaging sequence constantly repeats the same trajectory. As it will be shown, the repetitive sampling strategy is very favourable for the GPU implementation, as groups of frames could be processed in parallel using the same data structures.

In this chapter, I will; i) present the profiling results of the original CPU implementation, which were calculated and used to identify the bottlenecks of the algorithm when run on the multi-core CPU; ii) discuss the GPU implementation for the repetitive real-time acquisition; and iii) present the GPU reconstruction for continuous real-time data, which was implemented within the previously described distributed reconstruction system (Chapter 3). The continuous assessment of cardiac output during exercise was used as an example of real-time acquisitions requiring prohibitively long reconstructions.

4.2 Conjugate gradient linear solver algorithm for the SENSE reconstruction

The SENSE reconstruction for data acquired on arbitrary trajectories uses the conjugate-gradient solver algorithm (Section 1.4.2). The algorithm is an iterative method for solving sparse systems of linear equations, which in this case is a very computationally intensive process. The appendix 11.3 presents a pseudo code of the solver, adapted to the reconstruction needs, based on the

equations 1-41, 1-42 and 1-43. On the basis of the equations and pseudo code the reconstruction can be divided into repeated matrix operations and dot-product calculations. The major difficulty of an implementation are multiplications with the E and E^H matrices. However, the multiplications can be broken down to three operations (see Section 1.4.2; equations 1-44 and 1-45); FFT, gridding and linear *matrix combinations* with coil sensitivity maps (Fig. 4-1).

The reformulation of the SENSE reconstruction in the form of matrix multiplications and additions makes it a perfect candidate for implementation on the GPU platform. GPUs as vector processors were designed to support matrix operations (Section 1.5). For example, the I, D and θ (i.e. Equation 1-43) are diagonal matrices and the left-multiplication with them simplifies to *element-wise multiplication* (scaling operations). This operation can be very efficiently implemented as a dedicated *kernel* (Appendix 11.4). Also, many basic linear algebra operations (i.e. dot-product) and more commonly used signal processing functions (i.e. FFT) are readily provided in different libraries.

The new GPU implementation was based on a previous (original) multi-core CPU implementation for arbitrary trajectories. When profiled (Tab. 4-1), it revealed that the gridding operations were the major bottleneck, accounting to over 80 % of each iteration time. It is obvious that these operations needed to be significantly sped-up to improve the overall performance of the reconstruction.

Gridding is the process by which a non-Cartesian signal is resampled onto a rectilinear grid, by use of convolution (Section 1.3.4). Intrinsically the operation has no data dependency. However the irregular data addressing pattern complicates a GPU implementation. A process of porting onto the GPU platform requires careful analyses and design of an algorithm that complements the GPU specific hardware (59). There has been a significant work in this area, which I summarise below.

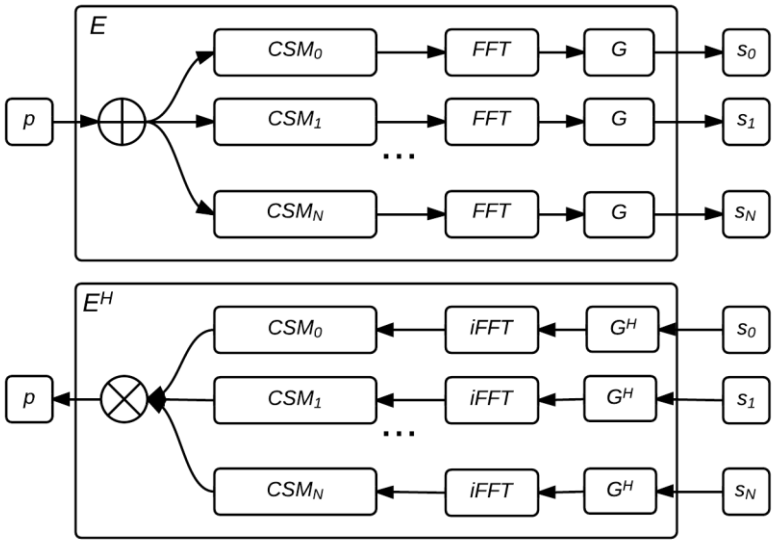

Fig. 4-1 Simplified block chart of the iterative SENSE algorithm.

4.3 Existing GPU gridding implementations

Conceptually, gridding is a very simple operation of combining the input data elements through the convolution operation. Convolution can be found in applications that need to process or analyse data resulting from some linear combination of another signal (sample data or, in general terms, functions). It is ubiquitous in digital signal processing, as it constitutes the basic step of many operations. Also, it is widely used in digital image processing (i.e. edge detection and calculation of image derivatives), digital data processing (i.e. Savitzky–Golay smoothing filters), physics (i.e. spectroscopy) and many other fields of science. The most common example is an image blurring or down-sampling (a convolution of an image with a Gaussian kernel).

A consequence of the widespread use of convolution is that it is crucial to optimise the operation in terms of speed and efficiency. Optimisation steps may depend on a targeted hardware. The procedure is intrinsically parallelisable, as in principle there are no dependencies between operations. Namely, each output result is not dependent on the other results.

For most applications convolution can be very efficiently implemented on the GPU (60-62), as most commonly the input and output are on rectilinear grids of the same or proportional sizes. This simplifies the implementation, as

each separate accumulation step does the same operation, but on a different subset of elements. The necessary kernel values are limited in number and can be pre-calculated or even predefined for specific applications. Nevertheless, the optimised implementation needs to take in account the specifics of GPU hardware. Most importantly these include coalesced memory read and write operations, as well as use of the shared on-chip memory, as software managed *cache* (43). Neglecting these features during the implementation process will have significant impact on the final performance.

A dedicated implementation for the transformation from an arbitrary trajectory onto a rectilinear grid is more difficult than the cases mentioned above. The input (or output in case of the inverse operation) may be of random (although always known) structure. This adds an additional step of calculating the kernel values, which can be very computationally intensive. Commonly, a table of kernel values is prepared only once in order to reduce a number of calculations. Next, the coefficients necessary for each convolution step are created through interpolation of the closest values from the table.

The lack of symmetry between input and output data has resulted in a few potential implementation strategies (50, 59) (Fig. 4-2). These can be categorised on the basis of how the work was assigned to executing thread or threads. The starting point is to evaluate *the Fine-grained input-driven* assignment (Fig. 4-2) in which the input trajectory points are assigned to a thread on a one-to-one basis. A thread's role is to identify output positions on a basis of a convolution kernel size. Next, kernel coefficients are calculated based on distances from the input point to the output points, and multiplied with the input value. The final step is to add the scaled values to values associated with each of the output positions.

This is a straight forward multi-threaded implementation for CPU like architectures. The implementation benefits architectures where threads are loosely coupled and which provide hardware managed *cache*. However, the GPU architecture does not allow *loose* execution of its treads. It is a direct result of the GPU design in which processing cores are tightly interconnected. This means that only the same operation can be executed by all of the cores (Section 1.6).

A modified *Coarse-grained* version (Fig. 4-2) can be designed to support this feature. This strategy has the optimal number of memory reads and has full benefit of coalesced memory read operations. However, the *input-driven assignment* entails a serious drawback for the GPU implementations, as the output regions are irregular causing non-coalesced memory access. Additionally, they can overlap each other. In consequence, the implementation requires *atomic* operations on the global memory. An operation (or set of operations) is *atomic* if its execution cannot be interrupted. Atomicity guarantees isolation from concurrent processes. This does not constitute a problem for CPU versions as the *atomic* operations, if not provided by the compiler, can be provided with specialised libraries or self-implemented.

These implementation challenges can be remedied with *the output-driven assignment* (Fig. 4-2). In this approach the output is split between threads. The algorithm assigns each of the output elements to a thread (Fig. 4-2 *the fine-grained* variant) or their group in a form of a region to individual block of threads (Fig. 4-2 *the coarse-grained* variant).

The coarse-grained output-driven assignment (Fig. 4-2) is preferable for GPU implementations, as the regular output area allows coalesced memory writes. However, in this case, the potentially irregular input structure poses a difficulty. Of course, the trajectory on which data was acquired is known. Hence, the data can be pre-sorted on the basis of the assignment to the output regions, allowing coalesced memory reads. However the input data must be shared between different output regions. This can be resolved by replicating the shared input data or padding of the output regions. Again the latter requires *atomic* operations on the global memory or some sort of post processing that combines all of the separate results into the final one. Additionally, a block of threads need to share the input data. For maximum efficiency the on-chip shared memory must be used as a programmatically controlled *cache* to avoid repeated read-outs from the global memory.

The analysis of the implementations of gridding for the GPU platform demonstrates it as not a simple task. To guarantee the maximum performance, the implementation would require long and tedious code development, problem specific data structures, tests and incremental optimisations. Additionally, the

final version would be a problem specific solution for a targeted hardware, which can be outdated in a short course of time.

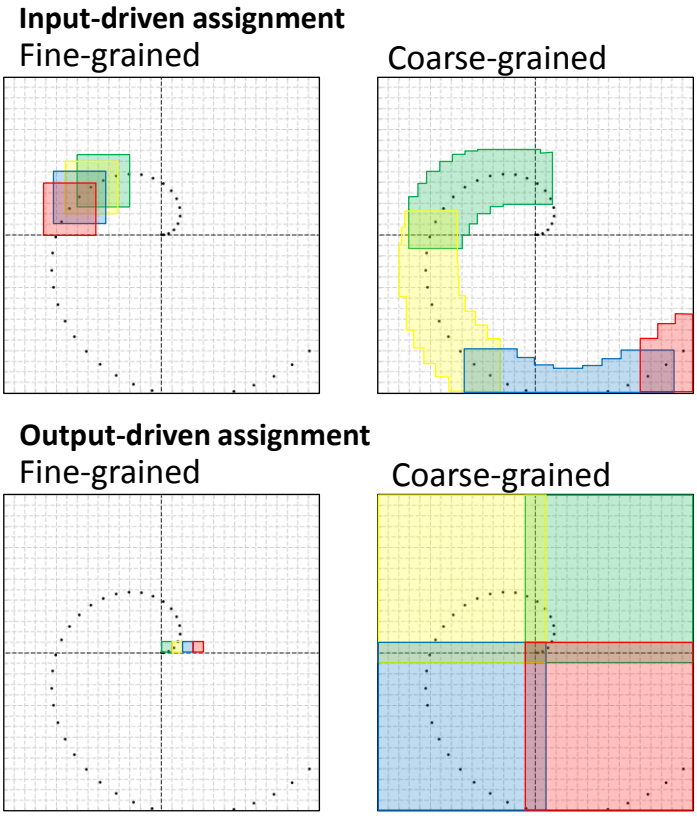

Fig. 4-2 Problem specific GPU gridding implementations (59).

4.4 The gridding operation as matrix multiplications

My work did not aim to create the fastest GPU implementation of the gridding operation; however the solution is proposed to avoid future re-implementations with new GPU architectures.

Gridding can be thought of as the weighted sum of all acquired k-space samples. However, since the kernel function is usually rapidly decaying, only a small fraction of k-space points have non-zero weights and a truncation can be applied. The specific points involved in the separate convolutions are thus determined by the sampling trajectory and size of the kernel. The weights for convolutions on each Cartesian coordinate can be organized as vectors, with the position in the vector referring to a different sampling trajectory position. Next, a matrix can be created by stacking the vectors in the form of rows, in an order relating to the index of Cartesian grid positions (Fig. 4-3). Now, gridding of

non-uniformly spaced data can be performed by multiplying the vector of acquired data with the matrix of weights.

$$G_{N,M}S_M^t = S_N^c \qquad 4\text{-}1$$

Equation 4-1 The gridding in form of matrix-vector multiplication.

This operation transforms a vector of M k-space data (S^t) samples onto a Cartesian grid (S^c) stored in a vector form of N points. Importantly, an inverse operation can be done by multiplying S^c with the conjugate-transpose of the gridding matrix ($G_{M,N}^H$) yielding data on the acquisition trajectory (S^t).

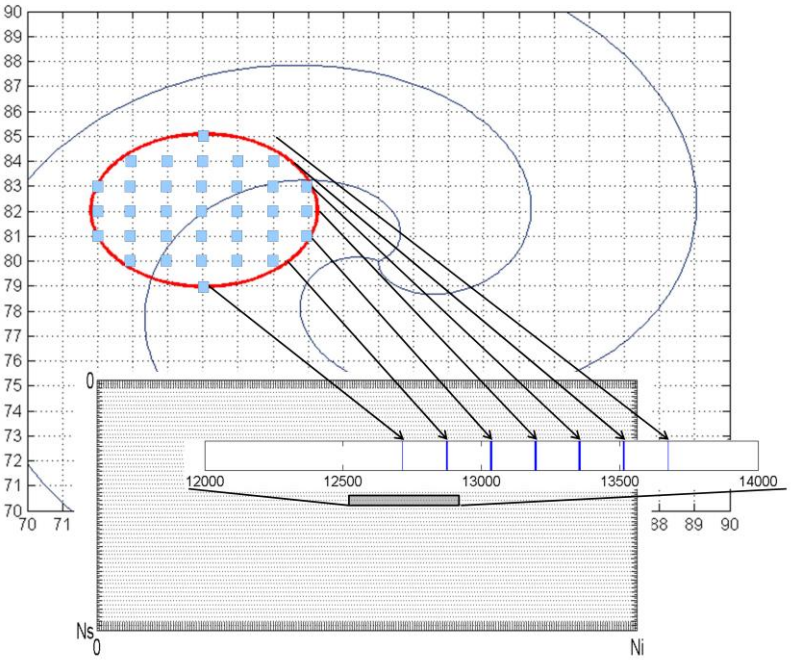

Fig. 4-3 Creation of the gridding matrix.

The arrows point storage positions of the kernel coefficients in the relevant row vector.

4.5 Batched gridding strategy for the repetitive trajectories

Further optimization of this strategy is possible when one considers the structure of raw multi-time frame MRI data. Firstly, k-space read-outs usually contain the same number of samples and therefore, the whole data can be kept in a single allocation organized in column-major matrix. In addition, where trajectories are repeated over consecutive frames, clustering of same trajectory into consecutive columns ($S_{M,T}^t$; T a number of read-outs) is possible. This

allows multiple simultaneous gridding operations in the form of matrix-matrix multiplication to be performed;

$$G_{N,M}S^{t}_{M,T} = S^{c}_{N,T} \quad \text{4-2}$$

Equation 4-2 Batched version of the gridding in form of matrix-matrix multiplication.

This operation produces a new matrix (S^c) of T data sets with N samples on a Cartesian grid. An additional step required by this approach is the calculation of a gridding matrix for each trajectory. In the case when only a single or very few matrices are necessary this step is trivial as they can be pre-calculated, stored as a sparse representation to minimize necessary memory usage, and reused when needed.

This method of optimization by batching (simultaneous execution of the same operation on multiple data) is one of the most widely used on GPU platforms. Launching a kernel requires preparation of parameters, their transmission onto a device and scheduling of the execution. The execution of the kernel happens asynchronously with respect to the scheduling thread but the preparation process needs to be repeated for each individual kernel call. Batching keeps the number of necessary kernel launches to absolute minimum.

4.6 Implementation specifics

This reformulation of gridding operations allowed me to leverage already existing and optimized libraries for linear algebra that are widely used in multiple scientific applications. The entire SENSE algorithm was ported onto the GPU, whilst keeping the number of CPU to GPU communications to the absolute minimum. The implementation used NVIDIA's CUDA 5.5 toolkit, which provided necessary libraries for sparse-dense matrix operations (*cusparse*), for simple linear vector, matrix operations, reduction and dot product calculations (*cublas*) and for 2D Fourier Transformations (*cufft*). Most importantly, all of them provide a batched version of their functions. Use of these libraries significantly reduced the time required for development. Moreover, the implementation benefits from continuous optimization with new releases of the toolkit, as well as their adoption to new GPU architectures. Some minor operators were not provided with the libraries (i.e. element-wise multiplications). These were

implemented in the form of in-house built kernels for execution on GPU. The implementation took full advantage of the continuous allocation and the batched approach was applied to them as well.

The implemented reconstruction was integrated into the distributed reconstruction system, as discussed in Section 3.4. The simplified data flow in the system for this reconstruction is presented on Fig. 4-4. The coil sensitivity maps, as well as the necessary preconditioning and regularisation maps, are calculated for each buffer (Section 3.4) prior to the SENSE reconstruction. After completion of the iterative SENSE reconstruction algorithm, Maxwell correction (63) and then standard PCMR subtraction are performed on the GPU. Execution of these steps on the external machine halves the size of result that need to be sent back; this way saving the network bandwidth. Upon completion of the external reconstruction, the resultant images are sent back into the scanner based reconstruction pipeline, for final conversion to the DICOM format and image viewing on the scanner console.

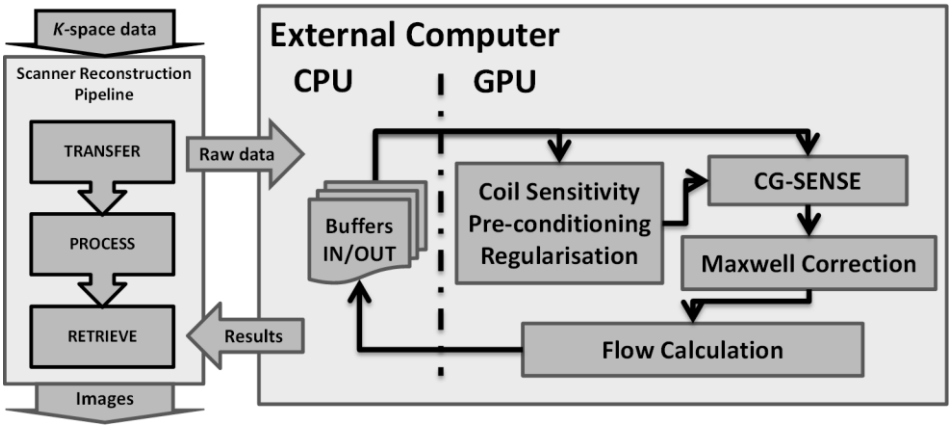

Fig. 4-4 Continuous real-time data processing with the distributed system.

Flow chart for the real-time PCMR data reconstruction process using a heterogeneous system incorporating an external computer equipped with a GPU card into the native image reconstruction system of a commercially available MR scanner.

4.7 Reconstruction tests

For optimal processing of a stream of real-time data an overlap between data acquisition and its reconstruction is required, as discussed in Section 3.4. The data transmission tests determined the maximum acquisition throughput supported by the system, as based on the maximum network

transmission speed. Correspondingly it was essential to ensure the processing of data would not be constrained by the external reconstruction time.

The reconstruction tests concentrated on the new gridding operation and tests of the final version of the GPU based SENSE reconstruction. The tests were run on the same system as presented in the networking tests (Section 3.5), using four separate data acquisition channels. The same uniform-density spiral PCMR sequence was used to acquire 60 flow frames; this was a size of reconstruction buffer. The imaging parameters were modified to match the desirable parameters in future assessments; FOV: 500x500 mm, matrix: 128x128, voxel size: 3.9x3.9x6 mm, TR/TE: 7.3/1.9 ms, flip angle: 25°, velocity encoding (VENC): 280 cm/s, complete k-space sampling: 12 interleaves. Data were acquired with four times acceleration, resulting in temporal resolution: ~44 ms. Neither water-only excitation, nor fat suppression pulses were used to minimize TR. The spiral interleaves were rotated with each frame to enable calculation of coil sensitivity maps. Under-sampled PCMR data was reconstructed using the new GPU SENSE reconstruction. Gridding was performed using a Kaiser-Bessel window function (19) (full-width: 8, oversampling factor: 1.25). This window size was chosen as a compromise between image quality and speed of reconstruction.

The Tab. 4-1 presents comparison of the reconstruction times between the CPU and GPU implementations. The gridding operations were found to be the major bottleneck per iteration, for the CPU implementation. They constituted over 80 % of each reconstruction iteration time. Each step in the iterative SENSE algorithm was faster on the GPU compared to the CPU. However, as expected, it was the reduction in the time taken for gridding (~46x quicker) that had the greatest effect on the total reconstruction time. Although, per iteration, the GPU was ~23x faster than the multithreaded CPU implementation, additional computational overheads (coil sensitivities, preconditioning, regularisation and other calculations) reduced the total speed-up to ~15x. Both CPU and GPU implementation required 8-9 iterations to converge. Most importantly the GPU reconstruction time for 60 frames (~1.6 s) was shorter than the acquisition time (~2.6 s). This meant that during online reconstruction of larger data sets, the processing would be fully overlapped with the acquisition.

		CPU [ms]	*GPU [ms]*	*CPU / GPU*
Per iteration	*FFT*	287.67	73.95	4
	Gridding	2674.75	58.48	46
	Matrix combination	245.69	4.24	58
	Preconditioning	52.95	1.00	53
	Total	3288.32	144.83	23
Per 60 frames	*Total*	24115.24	1581.67	15

Tab. 4-1 SENSE reconstruction time comparison.

Times of per iteration steps comprise total time required for both forward- and back-transformations between *k*-space and real domains.

The presented tests showed that the data transmission (Section 3.5) and reconstruction (Section 4.7) can be done faster than the acquisition. More in-depth system reconstruction tests required a more comprehensive test case. The next chapter describes online tests of the system to assess its suitability for the clinical use.

5. Real-time reconstruction for continuous acquisitions

In the chapter, an application of the developed system is presented on the example of continuous real-time assessment of blood flow. The work presented in the chapter was published in the article:

> *Real-time flow with fast GPU reconstruction for continuous assessment of cardiac output*, GT Kowalik, JA Steeden, B Pandya, F Odille, D Atkinson, A Taylor and V Muthurangu; Journal of Magnetic Resonance Imaging, Volume 36, Issue 6, pages 1477–1482, December 2012 (DOI: 10.1002/jmri.23736).

Appendix 11.8

5.1 Introduction

The continuous assessment of cardiac output during exercise may allow better understanding of the relationship between cardiac disease and exercise intolerance. Current clinical methods of continuous cardiac output assessment include; Doppler ultrasound, impedance cardiography and invasive measurements. However, they are impractical in the clinical environment and/or have been shown to have limited accuracy(64, 65). Flow quantification with PCMR sequences may provide a more suitable and accurate alternative (64-66). However, a high temporal resolution real-time PCMR sequence is needed to assess dynamic changes in cardiac output during an exercise. One possible implementation is spiral real-time PCMR (17). Unfortunately as discussed (Chapter 2) the reconstruction of under-sampled spiral data is very computationally intensive. Consequently, to truly make the assessment feasible the reconstruction of acquired data must be fast enough to enable multiple assessments and to not impede the clinical workflow. In this chapter it will be shown that our original online (CPU) reconstruction would take over one hour to reconstruct 10 minutes of continuously acquired real-time data. These long reconstruction times made continuous assessment of cardiac output with PCMR impractical in the clinical environment.

The purpose of this chapter was to demonstrate the potential of the described reconstruction system. This work aimed to; i) provide a quantitative validation of data provided with the new GPU reconstruction against the original CPU reconstruction, ii) demonstrate the significant improvement in reconstruction time on a demanding real-life application example; and most importantly, iii) present translation of this clinically important assessment protocol into an everyday examination tool, which was otherwise impractical. Therefore, the continuous cardiac output assessment during active exercise was selected as the test case. The study supplements the test results from the previous chapters by stress-testing the system with a 10 minutes real-time scanning protocol.

5.2 Methods

5.2.1 Study Population

Twenty healthy volunteers (9 Male: 11 Female) were recruited between August and September 2011. The median age was 31.5 years (range 25-51 years). Exclusion criteria were: i) Cardiovascular disease (assessed by clinical history); ii) Illness that prevented exercise (i.e. joint disease); iii) Contraindications for MR such as MR-incompatible implants, or pregnancy. The local research ethics committee approved the study and written consent was obtained from all volunteers.

5.2.2 Data acquisition and processing

All imaging was performed on a 1.5 Tesla (T) MR scanner (Avanto, Siemens Medical Solutions, Erlangen, Germany) using two six-element body-matrix coils. To reduce amount of acquired data the coils were set to combine mode. This is a feature of the scanner in which clusters of three coils are behaving as a circularly polarised coil read out through a single receiver. This resulted in four separate data acquisition channels.

The uniform density spiral PCMR sequence previously developed in our unit (17) was used to acquire the flow data. The imaging parameters were the same as in the reconstruction tests (Section 4.7); FOV: 500x500 mm, matrix: 128x128, voxel size: 3.9x3.9x6 mm, TR/TE: 7.3/1.9 ms, flip angle: 25°, velocity encoding (VENC): 280 cm/s, complete k-space sampling: 12 interleaves. Data were acquired with four times acceleration, resulting in temporal resolution: ~44 ms. To minimize TR, neither water-only excitation nor fat suppression pulses were used. To enable calculation of coil sensitivity maps the spiral interleaves were rotated with each frame. The described networking framework (Chapter 3) was used to enable the overlapping data transmission and external reconstruction. The raw k-space lines were sent using the client application implemented on the scanner side to the reconstruction server on the external workstation for processing (Section 3.4). The tests were run on the same hardware as presented in the networking tests (Section 3.5). The data processing was done as described in Section 4.7. Importantly, incoming data were buffered on CPU memory and only sent to GPU memory when a whole

packet had been collected. Data were processed in packets of 60 flow frames to ensure sufficient SNR for the coil sensitivity calculations and reduce motion within a packet.

Under-sampled PCMR data were reconstructed using the new GPU SENSE reconstruction (Chapter 4). Gridding was performed using a Kaiser-Bessel window function (19) (full-width: 8, oversampling factor: 1.25). This window size was chosen as a compromise between image quality and speed of reconstruction.

5.2.3 In-vivo validation of GPU reconstruction

As the reconstruction of the exercise data was not possible without major changes to our existing multicore CPU implementation (due to the data size), a smaller data set was used for validation and comparison of GPU and CPU reconstructions. Similarly to the reconstruction tests (Section 4.7), the SENSE reconstruction was run on a data set of 60 frames; equivalent to a single reconstruction buffer or the length of one packet in the continuous acquisition. This small set was reconstructed offline using both the GPU reconstruction and the original CPU implementation.

5.2.4 Vascular response to exercise

The exercise was performed with an MR-compatible ergometer (Lode, Groningen. Netherlands), the participants were placed supine in the MR scanner, with their feet strapped into the pedals and the upper leg strapped to supports of the ergometer, prior to the scan. The exercise consisted of an up- and downward motion of the pedals. This type of exercise is designed to minimize motion artefacts as motion is restricted to the lower legs. The 10 minute exercise protocol consisted of 1 minute of rest, 8 minutes of ramped exercise (starting at 2W and increasing by 2W every minute) and 1 minute of recovery. During the entirety of the exercise protocol, real-time flow data was acquired using the described sequence resulting in 13980 frames (~612 s) of PCMR data.

5.2.5 Image analysis

All images were processed using in-house plug-ins for the open-source software OsiriX (the OsiriX Foundation, Geneva, Switzerland) (67) performed on a multicore workstation (12 core, Mac Pro, Apple, CA, USA). The magnitude flow images were segmented semi-automatically using a registration based algorithm (68). This requires the user to select and segment the aorta in one reference frame, and the plug-in performs subsequent propagation of this region of interest (ROI). For the CPU/GPU comparison the CPU magnitude images were segmented and the same ROIs were used to quantify flow in both the CPU and GPU phase images.

The original segmentation plan had to be modified for the exercise data set due to its size. The original algorithm registers a single frame from a set to all of the others. In the case of long exercise data sets this was not desired and slow. Thus, I implemented a new split segmentation approach. The task was parallelized across multiple-cores by automatically dividing the full data set (13980 frames) into 12-15 equally sized subsets, each processed by a separate CPU thread. Subsets were presented within separate windows (Fig. 5-1) allowing the initial segmentation of the aorta in each of them. Next, a CPU thread was assigned to each subset and the original registration algorithm was run in parallel on each of them. Resultant ROIs were visually assessed and if necessary subsets were individually re-segmented to improve accuracy. The final ROIs were copied onto the phase data for flow quantification (Fig. 5-2).

5.2.6 Statistical Analysis

All aortic flow results were expressed as the mean ± standard deviation (SD). Measurements of agreement between the CPU and the GPU reconstruction were performed using Bland-Altman analysis, as well as calculation of correlation coefficients.

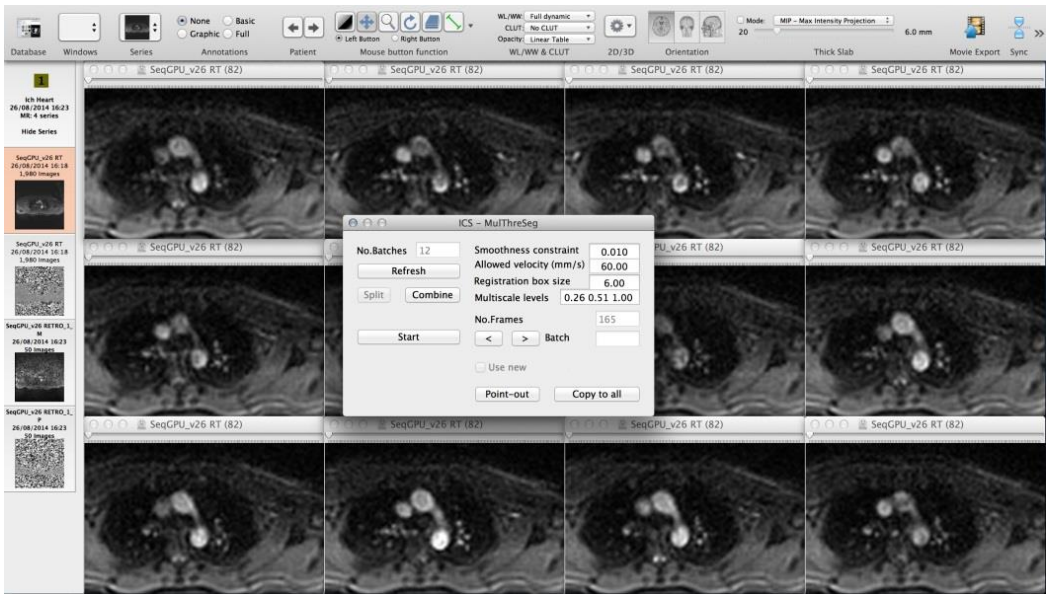

Fig. 5-1 Multi-threaded segmentation plug-in.

A series of magnitude images was split into 12 sub-sections that were processed separately. Each window allowed visualisation of a sub-section to enable selection of a reference frame to which the rest of images were registered. After finished segmentation the plug-in enables combination of all sub-sets back into the single series in the original order. The prepared ROIs were preserved and propagated on the series for visualisation and further processing.

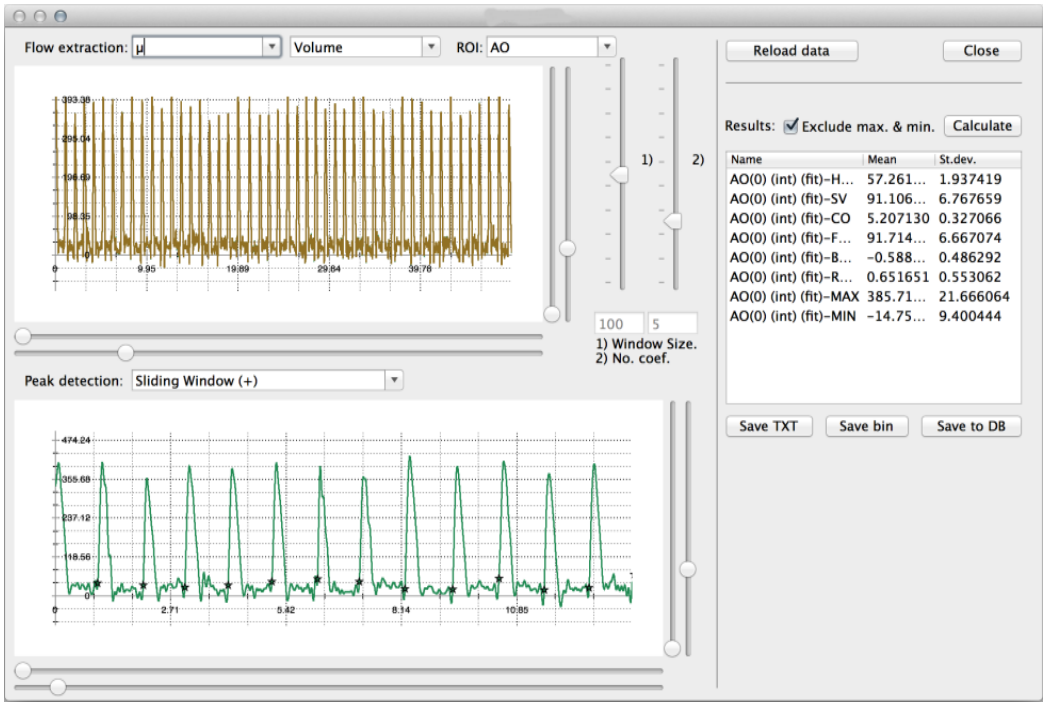

Fig. 5-2 Flow analyse plug-in.

Flow data were extracted on the basis of ROIs and were presented in the top plot. The peak detection algorithm was used to detect individual R-R intervals. These were used to calculate heart-rate, stroke volume, cardiac output, forward and backward flows. The averaged numerical values were presented in the table on the right hand side. The plug-in allowed exporting the results for further processing.

5.3 Results

5.3.1 Reconstruction validation

There was no observable difference in image quality between the CPU and GPU reconstructions (Fig. 5-3).

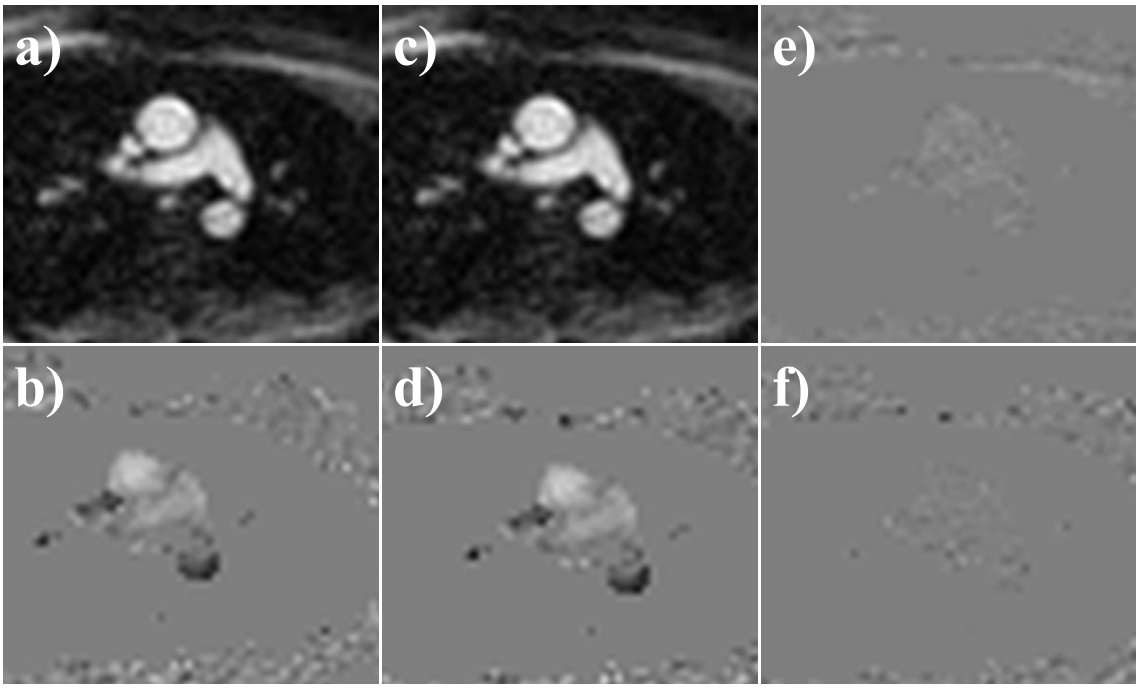

Fig. 5-3 Image quality comparison.

The first column – CPU reconstruction results: a) magnitude and b) phase images. The second column – GPU reconstruction results: c) magnitude and d) phase images. The third column – difference images of e) the magnitude and f) phase images. Images (a, c, e) and (b, d, f) are presented with the same window width. The phase images (b, d, f) were masked to remove the low SNR pixels.

Bland-Altman and correlation analysis (Fig. 5-4) demonstrated a negligible bias (~0.4 ml) and excellent agreement (limits of agreement: -1.9 to 1.2 ml, r = 0.998, $P < 0.05$) in aortic stroke volumes measured using the CPU and GPU reconstructions of the 60 frames data set.

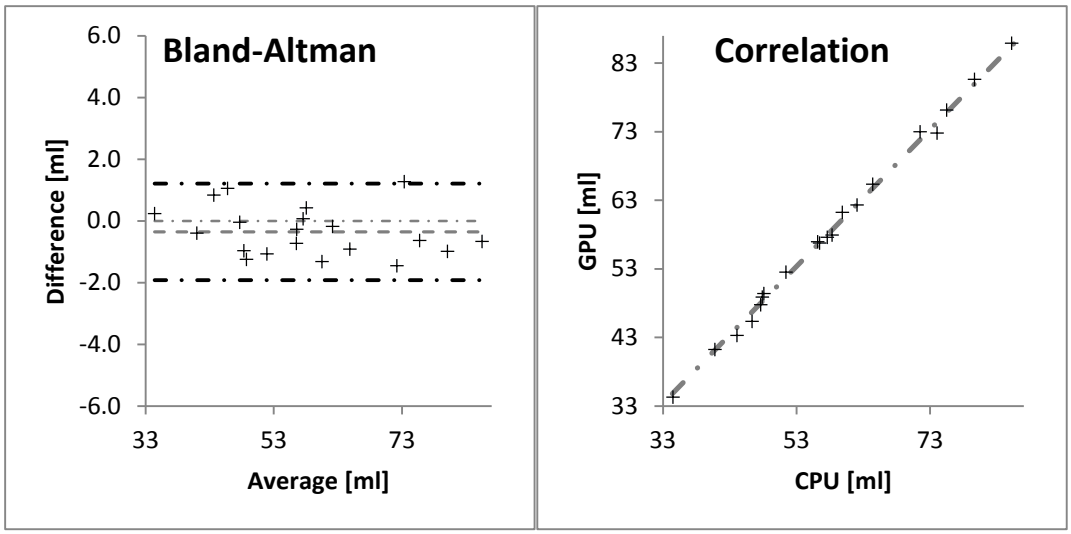

Fig. 5-4 Flow quantification validation

The figure presents plots of Bland-Altman and correlation analysis for the set of aortic flow data from 20 volunteers, reconstructed with the CPU and the new GPU reconstructions.

5.3.2 Reconstruction times

The reconstruction times of a single data packet (Tab. 4-1) were presented in the previous chapter (Section 4.7). The same results were found in this study with data acquired over much longer time period. Consequently, the external reconstruction times of each data packet were shorter than the acquisition and transmission times of the packet (Fig. 5-5).

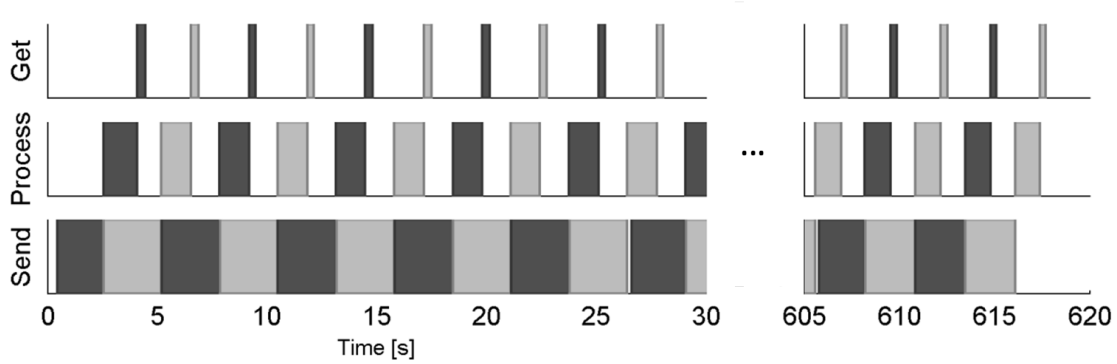

Fig. 5-5 System workload during the continuous flow assessment.

The alternating colours identify rotating processing between two buffers.

As mentioned, the complete exercise studies were not run through the CPU implementation. Consequently, an estimated CPU reconstruction time was calculated (mean reconstruction time for 60 frames multiplied by the total

number of packets), which equalled to ~93 minutes. Thus, data would theoretically be available ~83 minutes after the acquisition finished.

The estimated speed ups related to performing the reconstruction on the GPU for this application are shown in Tab. 5-1. On average, total reconstruction time with the external GPU reconstruction for the exercise data was ~629 s (data length: 13980 frames, acquisition time ~620 s). This included external reconstruction, data transmission/buffering to and from the external computer and results storage on the scanner, and resulted in all the data being available ~9 s after the scan finished. This represents a ~556x speed-up if measured from the end of acquisition. Of course, this is not a practical comparison, as the speed up value depends on a size of the problem. A better way of classifying algorithms is by *big-O* notation (Landau's symbol). $O(\phi)$ can be used to classify an algorithm's processing time (f) by specifying it's limiting behaviour (ϕ) with respect to an input size (n); $f(n) < A\phi(n)$. A is a constant. Applying this notation to the waiting period after the acquisition finishing as a metric of improvement yields $O(n)$ (linear complexity) for the CPU and $O(1)$ (constant complexity) for the GPU reconstructions. This shift in the algorithm's classification was the crucial improvement that removed the reconstruction time as a restricting factor and enabled the real-time assessment studies.

Also, the reconstructed data were collected throughout the acquisition time, which made it possible to view images during acquisition (with a slight delay).

		CPU [s]	*GPU [s]*	*CPU / GPU*
Per 13980 frames	*Total*	5618.85*	629.24	9
(buffered recon.)	*From acquisition end*	4998.61*	9.00	556

Tab. 5-1 Continuous flow assessment reconstruction time comparison.

* CPU times for the entire dataset of 13980 frames were estimated from the CPU reconstruction time required for a subset of 60 frames.

5.3.3 Continuous cardiac output monitoring

All participants successfully completed the exercise protocol and MR flow data were measured successfully on all subjects. Flow curves collected during a whole exercise study and selected portions are shown in Fig. 5-6.

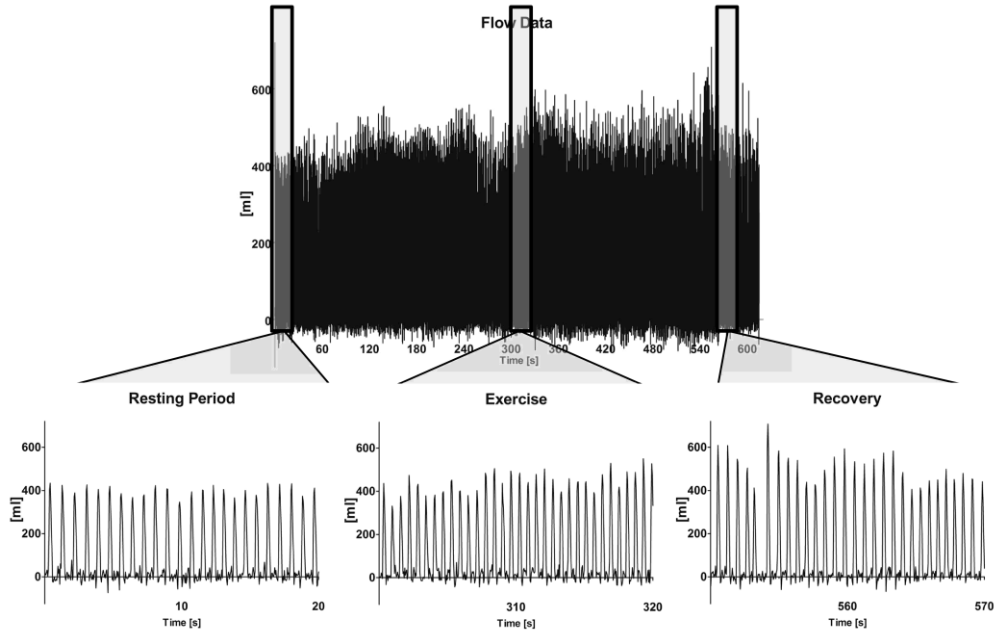

Fig. 5-6 An example of flow data acquired with continuous real-time PCMR during exercise. Bottom graphs show 20 s sections of the total data taken at rest, mid exercise and recovery.

Increased heart rate during exercise is observable in this raw flow curve data. Fig. 5-7 shows the average measured responses to exercise, which demonstrate the expected behaviour (17). Cardiac output increased throughout the exercise protocol, particularly in the first minute of exercise. This was assigned to the increase in heart rate as stroke volume slightly declined during the 10 minutes of exercise. During recovery, heart rate fell dramatically although not to baseline, while stroke volume increased back to baseline. This led to a greater cardiac output in the first minute of recovery compared to baseline.

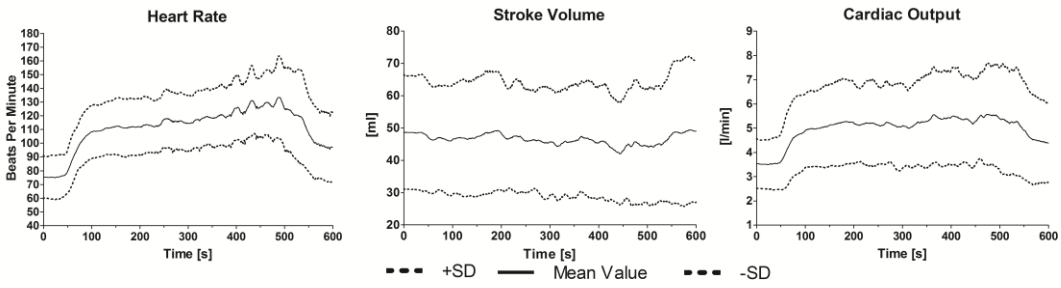

Fig. 5-7 Exercise data results.

The figure presents average and standard deviations of heart rate, aortic stroke volume and cardiac output based on flow data from the population of 20 volunteers.

5.4 Discussion

The study showed the continuous MR assessment to be impractical with the CPU reconstruction. The CPU reconstruction was estimated to take over 80 minutes to reconstruct ~10 minutes of real-time data. The gridding was the bottleneck of the CPU reconstruction accounting for majority of its time. However, as described (Section 4.3) the gridding problem was well studied and the new developed GPU version (Section 4.4) was shown to provide significant speed-up for the iterative SENSE algorithm (Section 4.7).

The GPU implementation made the reconstruction faster than the acquisition. This when combined with efficient data transmission (Section 3.5) enabled the buffered reconstruction process. Consequently, a single packet of continuous data was reconstructed and transmitted back to the scanner before the next packet was fully acquired. This allowed monitoring of the results and potential re-starting of the scan if necessary (i.e. if a patient moves) without waiting till the end of the acquisition. More importantly, the time between the acquisition finishing and the whole data set being available for viewing was independent of the scan length (~9 s in this study). This means that much longer acquisitions can be acquired without a significant effect on the clinical workflow. Furthermore, the described distributed reconstruction system preserved the existing clinical workflow; for example the simple transfer of final image data (including patient biometrics stored in the DICOM headers) to processing and storage nodes stayed unchanged. This was essential for the assessment protocols to be effectively translated into the clinical environment.

The main objective of this study was to enable and validate the cardiac output measurement during exercise. First, the validation tests proved the new reconstruction equivalent to the original one and consequently it can be used instead. Next, the exercise results were in keeping with the type of supine exercise performed (69). To make the continuous real-time assessments feasible in the busy clinical environment the bottleneck in the form of prohibitively slow reconstructions was removed. However, a new bottleneck was uncovered; slow post-processing of the reconstructed data. Although segmentation was parallelised over the multiple CPU cores, the requirement for some manual correction led to relatively long post processing times (~1 hour).

Thus, more accurate segmentation algorithms better suited to such large data sets are required. Nevertheless, the study as a proof of principle opened up many novel areas in diagnostic and research cardiovascular MR that are currently impeded by long reconstruction times. For instance, cardiac output data could be combined with oxygen consumption to fully assess the cardio-respiratory response to exercise (70). Furthermore, long-term beat-to-beat analysis could be performed (similar to beat-to-beat heart rate analysis) providing new insight into cardiovascular control mechanisms in health and disease (71).

6. High temporal resolution real-time acquisitions with temporal encoding

In the chapter, I demonstrate the flexibility of the system to accommodate multiple of reconstruction algorithms. The UFOLD technique was developed and introduced to improve acquisition speed and/or image quality. The work presented in the chapter was published in the following article:

> *Assessment of cardiac time intervals using high temporal resolution real-time spiral phase contrast with UNFOLDed-SENSE*, GT Kowalik, DS Knight, JA Steeden, O Tann, F Odille, D Atkinson, A Taylor and V Muthurangu; Magnetic Resonance in Medicine, Volume 73, Issue 2, pages 749–756, February 2015 (DOI: 10.1002/mrm.25183).

Appendix 11.9

6.1 Introduction

One of principal goals of my work was to promote flexibility through development of robust *building blocks* and preparation of a computationally efficient translation environment. The first steps (Chapters 3) presented the distributed architecture as a scalable framework capable of accommodating the demanding acquisition sequences and reconstructions. Next, the GPU version of the SENSE algorithm was shown to very significantly boost the reconstruction's performance, which elevated the algorithm's utility in the clinical environment (Chapter 4). These components, in combination with the previously developed spiral PCMR sequence, were shown to have a sufficient impact to open new research paths (Chapter 5).

This section describes the next step, which was to illustrate that the framework was not limited in the range of MR techniques which can be built within it. Other existing MR components can be integrated into the system and benefit from its seamless and robust reconstruction process. The combination of the temporal encoding technique (UNFOLD) with SENSE was selected as the first example. The presentation of the system's expandability was not the only reason behind this choice. More importantly it provided an incremental improvement in the range of available assessment protocols. The temporal encoding technique is used to double the acquisition speed. However this can be exploited in two ways; improving image quality through higher spatial resolution or better artefact suppression, while preserving a temporal resolution, or doubling the temporal resolution, while sustaining the image quality. Of course the choice depends on the application.

The problem of assessing cardiac time intervals with PCMR was used as the target application. This allowed me to present; i) the scalability of system, ii) the incremental improvement in quality of assessments; and iii) actual real-life applicability. Cardiac time intervals (i.e. isovolumic times and ejection time) can aid the assessment of integrated myocardial function (72). Usually the time intervals are assessed by Doppler echocardiographic measurement of ventricular in- and out-flow patterns, but they could also be assessed with PCMR. However, PCMR is conventionally cardiac-gated and this introduces two

major problems. Firstly, flow patterns produced by gated PCMR may be distorted by inter-beat variation in stroke volume and heart rate. This has little effect on quantification of velocity, but may affect the reliability of time interval measurement. Secondly, acquiring gated data with sufficiently high temporal resolution takes several minutes, limiting its utility in the clinical environment. An alternative option is to use real-time PCMR, which additionally can be used during an exercise. Unfortunately, assessment of cardiac time intervals requires very high temporal resolution, as the time intervals can be as short as ~30 ms. This is significantly lower than the sampling rate enabled by the combination of efficient k-space filling (i.e. spiral acquisition trajectory) and parallel imaging (i.e. SENSE), which typically results in 40-50 ms sampling rate.

In the case of multiple frame acquisitions the temporal domain can be utilised for further acceleration. Previous work has shown development of *k-t* BLAST and *k-t* SENSE (28) techniques, which allow higher acceleration factors for multi frame imaging (73). These techniques require acquisition of low spatial and high temporal resolution training data. These data are intrinsically present in under-sampled radial data due to the oversampling of the central portion of *k*-space. This is in contrast with spiral acquisitions. The rapid data sampling of the trajectory would need to be compromised to enable the simultaneous acquisition of the training data with the variable density spirals. Consequently, it would reduce the achievable sampling rate and increase imaging artefacts (due to longer read outs). A better approach for spiral imaging may be the previously described temporal encoding/filtering technique – UNFOLD (Section 1.4.3), which in combination with SENSE (27, 29) should allow the acquisition of real-time data with high enough temporal resolution to assess cardiac time intervals.

Therefore, the high temporal resolution real-time spiral PCMR sequence that combined UNFOLD and SENSE reconstructions (UNFOLDed-SENSE) was implemented within the distributed image reconstruction system. The implementation process fully reused the previously developed accelerated real-time spiral PCMR sequence (17), overlapping data transmission-execution framework (Section 3.4), and GPU based SENSE reconstruction (Chapter 4). A new trajectory ordering pattern was applied to allow the temporal encoding for UNFOLD and self-referencing for creation of coil sensitivity maps. The major

implementation work concentrated on adding a new reconstruction step in the form of a filter designed to remove the temporal under-sampling. The new technique was validated in-silico, in-vitro and in-vivo to assess its suitability for the measurement of time intervals. Experimentally, the method was used to evaluate the changes in cardiac time intervals with exercise.

6.2 Methods

6.2.1 Data acquisition and processing

All imaging was performed on a 1.5 Tesla MR scanner (Avanto, Siemens Medical Solutions, Erlangen, Germany) using six-element spine and body-matrix coils (total of twelve elements used in acquisition).

The same type of real-time uniform density spiral PCMR sequence as in the continuous cardiac output assessment was used (Chapter 5). The imaging parameters were; FOV: 450x450 mm, matrix: 128x128, voxel size: 3.5x3.5x7 mm, TR/TE: 7.4/2.0 ms, flip angle: 20° and VENC: 150 cm/s, complete k-space sampling: 10 interleaves. In order to minimize TR, neither water-only excitation nor fat suppression pulses were used. The trajectory acquisition patterns had to be modified to achieve high sampling resolution (<15 ms) and allow combination of the UNFOLD and SENSE reconstructions. The data acceleration factor was split into spatial ($SENSE$) and temporal accelerations (temporal encoding). The temporal acceleration for the UNFOLD technique was fixed: 2x, leaving the spatial acceleration to be determined depending on the desired temporal resolution. In this study we optimised the parameters for imaging of cardiac time intervals. A single interleave was acquired per flow frame, this resulted in 10x under-sampling ($SENSE = 5$). The novel acquisition pattern was implemented to fulfil the temporal encoding criteria (Section 1.4.3) and enable the self-referencing SENSE approach for the calculation of coil sensitivity maps (27). The UNFOLD reconstruction was accommodated by acquiring alternate lines in consecutive frames, while the full coverage of k-space was enabled by rotating this pair of alternating lines every N frames; referred to as an acquisition block (Fig. 6-1). In this study N was set to 20. This allowed the maps used in the reconstruction (Section 1.4.2) to be calculated by combining data from $N * SENSE = 100$ frames, with the resultant maps having $N/2 = 10$ signal

averages (Fig. 6-1). This data acquisition plan required the total number of frames to be divisible by a multiple of the size of acquisition block and the spatial acceleration ($N * SENSE$), which equalled to 100 for the presented parameters. In the validation studies a total of 700 frames were acquired resulting in ~10.37 s of scanning time.

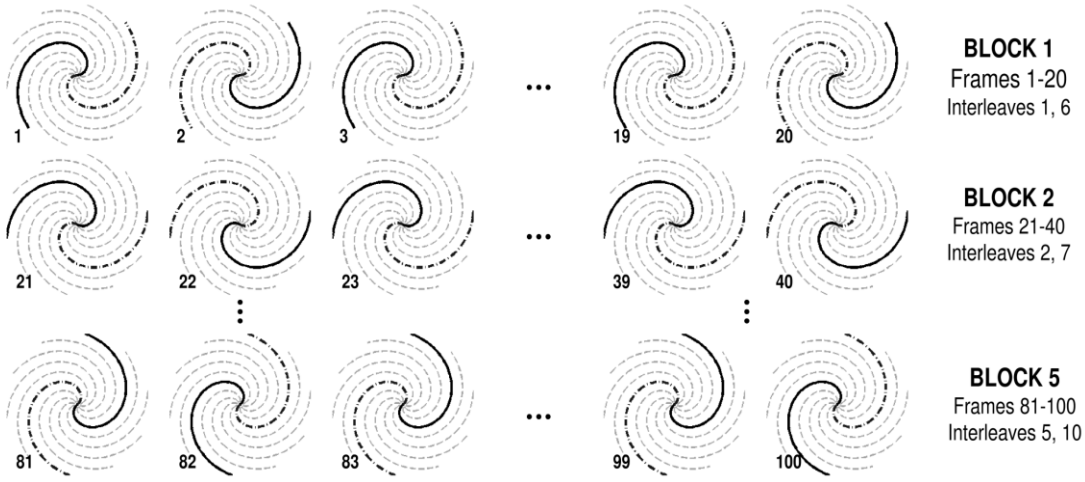

Fig. 6-1 Sampling trajectory pattern used in the 10x accelerated UNFOLDed-SENSE reconstruction.

A batch of 100 frames is organised into five blocks of 20 frames (3 blocks are shown in the figure). Each row shows the alternating acquisition pattern within a block, with the acquired interleave in each frame (numbered) in continuous line. Each block uses a different pair of interleaves, thus data acquired in five adjacent blocks fully covers the whole k-space and can be used to calculate the coil sensitivity maps.

The temporal filtering (Section 1.4.3) can be done in either image space or k-space (74) providing the following condition is met;

$$\begin{gathered}\rho(r,t) = \mathcal{F}_t^{-1}[\rho(r,\omega)] \\ \vec{S}(k,t) = \mathcal{F}_t^{-1}\left[\vec{S}(k,\omega)\right] \\ \vec{S}(k,\omega)f(\omega) = \rho(r,\omega)f(\omega) \Leftrightarrow \forall_{\omega_a \in \mathbb{R}} \vec{S}(k,\omega_a) \overset{\mathcal{H}}{\leftrightarrow} \rho(r,\omega_a) \\ \mathcal{F}_r^{-1}\left[\vec{S}(k,\omega)f(\omega)\right] = \rho(r,\omega)f(\omega)\end{gathered} \qquad 6\text{-}1$$

Equation 6-1 Filter reciprocity condition

In this implementation UNFOLD was performed before the SENSE reconstruction. Primarily, the reason was to improve the initial conditions and consequently the convergence rate of the subsequent iterative SENSE solving process. This assumption was based on the fact that the UNFOLD filtering

process, besides reducing the initial under-sampling (Fig. 6-2), removes high frequency noise and in this way improve SNR (33).

In equation 6-1, $\varkappa$ denotes a transformation between image space and k-space that does not depend on any temporal information about the imaged object. As demonstrated Fourier transform is such an operation. However, it can be only directly applied to Cartesian trajectories. For non-uniformly spaced accelerated trajectories, reconstructions like SENSE play the role of the transformation operator. Unfortunately, these reconstructions do not always conform to this condition. For instance, regularisation used to constrain the noise amplification, can alter the solution and have a negative impact on the oscillations in image space. Also, if coil sensitivity maps are not calculated correctly for each frame the accuracy of the result can vary through time introducing a temporal component. Presumably, these have negligible effect on the performance of UNFOLD post transformation into image space, as conducting temporal filtering in image space is the most common approach (29, 33, 40).

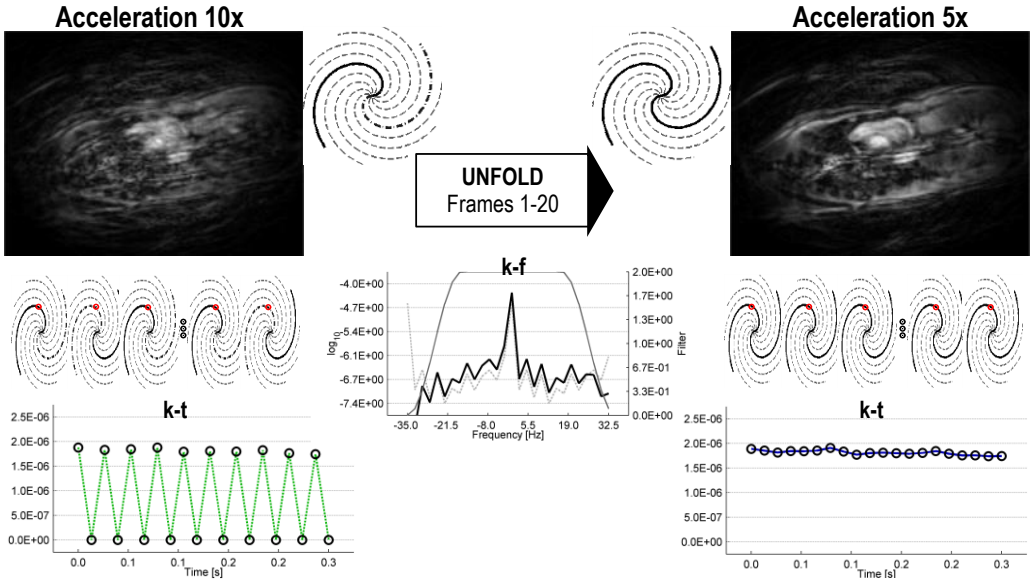

Fig. 6-2 Schematic visualisation of the temporal encoding used in the UNFOLDed-SENSE.

The figure presents reduction in acceleration/under-sampling/aliasing with the UNFOLD technique applied directly to k-space data. On the left, a series of 20 frames acquired with alternating 10x accelerated trajectories has 10 spatial aliases (the top left image). Also, each acquired position is two times under-sampled through time (the bottom left plot). This results in aliasing in the temporal frequency space (the middle plot) which can be removed with an adequate filter. On the right, resultant data of the filtering process is fully sampled through time (the bottom right plot). This is equivalent to halving of the number of spatial aliases (the top right image).

A schematic visualisation of the UNFOLDed-SENSE reconstruction steps are presented in Fig. 6-3. UNFOLD was performed separately on each acquisition block. As each k-space position in the acquired interleaves, was 2x under-sampled in time (Fig. 6-2), Fourier transformation along time resulted in aliasing at the Nyquist temporal frequency. The aliases were removed using a low-pass temporal frequency filter and after inverse Fourier transformation, every k-space position in the processed interleave was fully sampled in time (Fig. 6-2).

This resulted in each k-space frame containing two interleaves, a reduction in under-sampling from 10x to 5x. The reconstructed signal may suffer from *ringing* artefacts (38). This can happen when a temporal filter is too *abrupt* and/or a *jump discontinuity* is present between the beginning and end of a filtered signal – similarly to Gibbs artefacts. To address this problem, prior to UNFOLD, each acquisition block was extended by four frames in either direction (by replicating the first and last two frames) (Fig. 6-8). Therefore, potential ringing introduced by the filtering process was *pushed* into these additional frames, which were then discarded prior to the SENSE reconstruction.

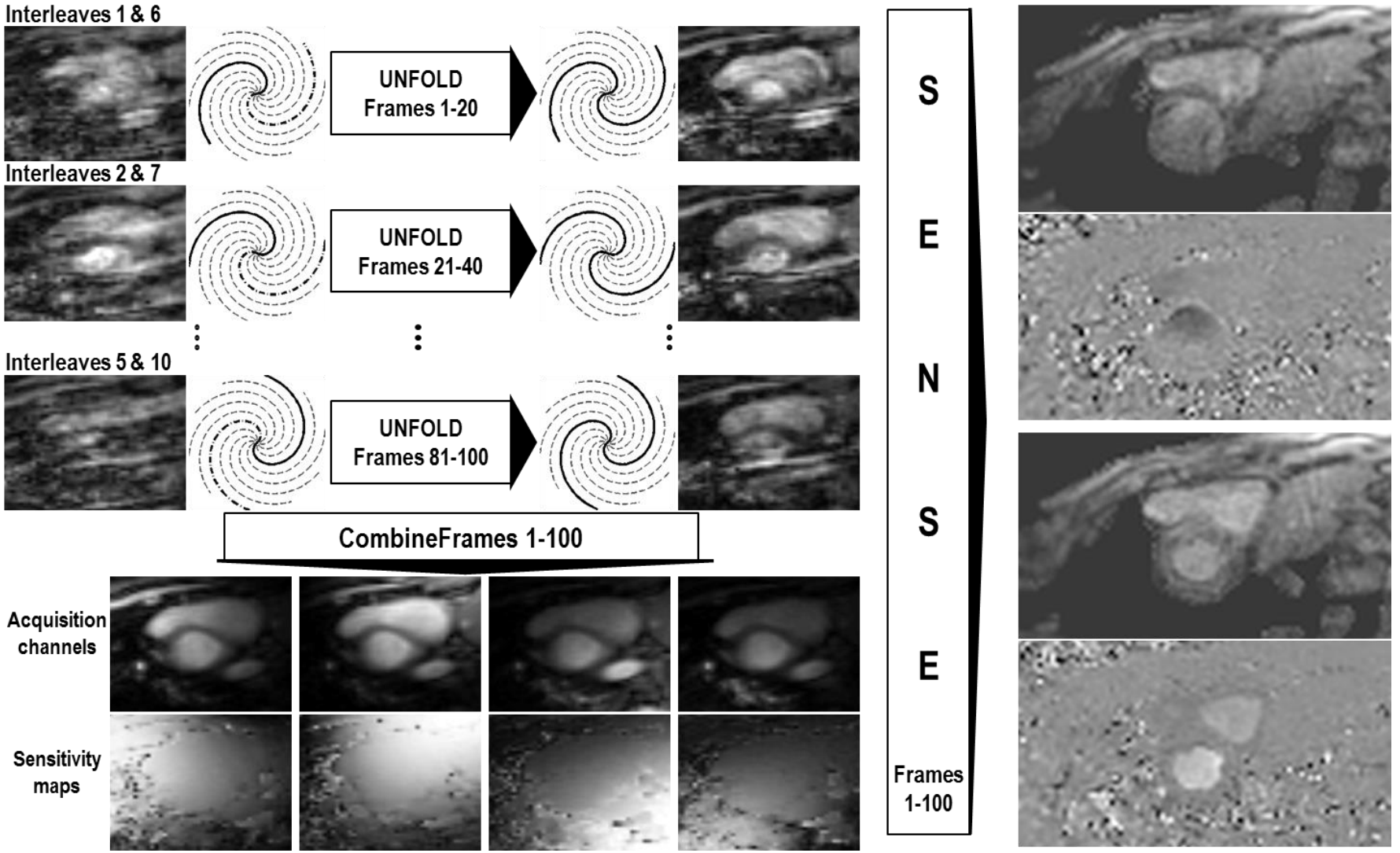

Fig. 6-3 UNFOLDed-SENSE reconstruction process.

The implementation of UNFOLDed-SENSE made full use of the previously described distributed reconstruction system (Fig. 6-4). All real-time reconstructions were performed online. The networking framework (Chapter 3) was used to enable the overlapping data transmission and external reconstruction of the acquired real-time data. A new temporal filtering step was added to the reconstruction process prior the SENSE reconstruction. This step allowed flexible definition of the filter parameters required for the optimisation tests. The resultant PCMR data, after temporal filtering (still 5x under-sampled in the spatial domain), were next reconstructed using the implemented SENSE reconstruction (Chapter 4) on the external machine (Workstation Specialists, Two Intel Xeon E5645 2.4 GHz, 24 GB DDR3) equipped with the GPU (NVIDIA Tesla C2075 1.2 GHz, 6 GB DDR5). Gridding was performed using a Kaiser-Bessel window function (19); full-width: 5, oversampling factor: 1.25. The parameters were chosen to maximise reconstruction speed. Coil sensitivity maps were calculated by combining the original velocity compensated interleaves from 100 consecutive frames (the minimum number of frames required to ensure complete k-space filling). These coil sensitivity maps were used in the iterative non-Cartesian SENSE reconstruction on the same 100 frames after they have undergone the UNFOLD reconstruction.

Intrinsically, UNFOLD is a temporal filtering technique that can introduce blurring and reduce the temporal resolution of data. The optimal

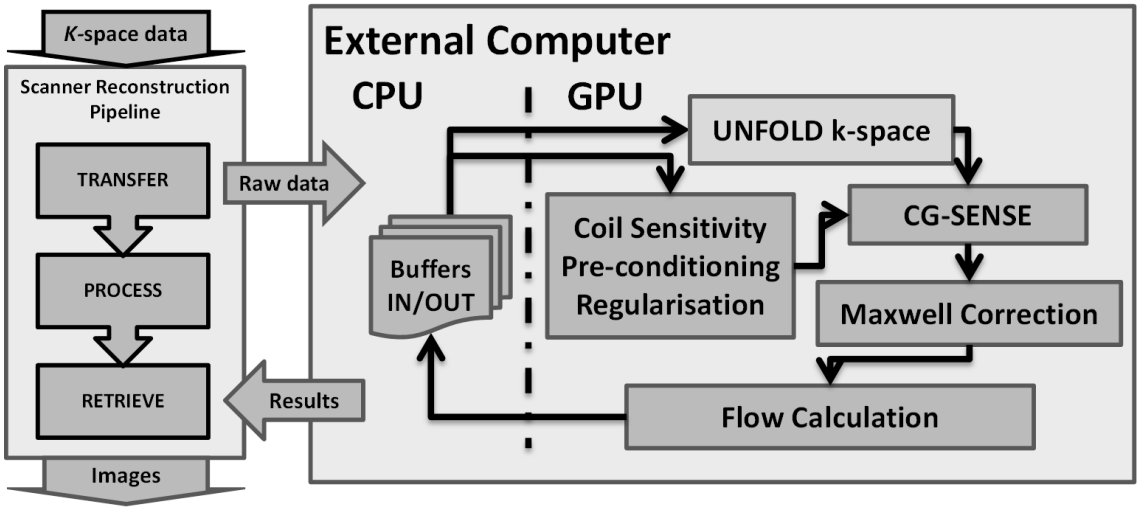

Fig. 6-4 Modified continuous real-time data processing for UNFOLDed-SENSE.

The processing flow chart was extended with the temporal filtering step prior to the SENSE reconstruction.

selection of the filter is crucial for good performance of the technique. This depends on the temporal frequency spectrum of the imaged object. To find optimal filter parameters for flow data acquired in the assessment of cardiac time intervals, an in-silico experiment was designed as described below.

To validate the necessity of the combination of UNFOLD with SENSE in the assessment of cardiac time intervals two additional reconstructions were run retrospectively on the acquired real-time data. Both can be seen as a form of sliding window reconstruction (Fig. 6-5). Firstly, the data was treated as if no temporal encoding was applied. This meant twice lower sampling resolution (~30 ms) (Sliding Windows with Low Temporal Resolution - SW-LTR). This reconstruction can be seen as an equivalent of the previously presented continuous real-time PCMR assessment. It was used to evaluate potential improvements with the higher temporal resolution acquisitions. Secondly, the data from adjacent frames were used to create combined frames with halved under-sampling. This is an equivalent of convolving the data with a box kernel of width of two. Consequently, it is a temporal filter of fixed broad window characteristic. This combination of k-space data produced the same sampling resolution as UNFOLDed-SENSE (Sliding Windows with High Temporal Resolution - SW-HTR) and was used to assess how the *simpler* form of temporal filtering compares with the tailored approach in UNFOLD. Similarly to UNFOLDed-SENSE, both sliding windows reconstructions reduced the

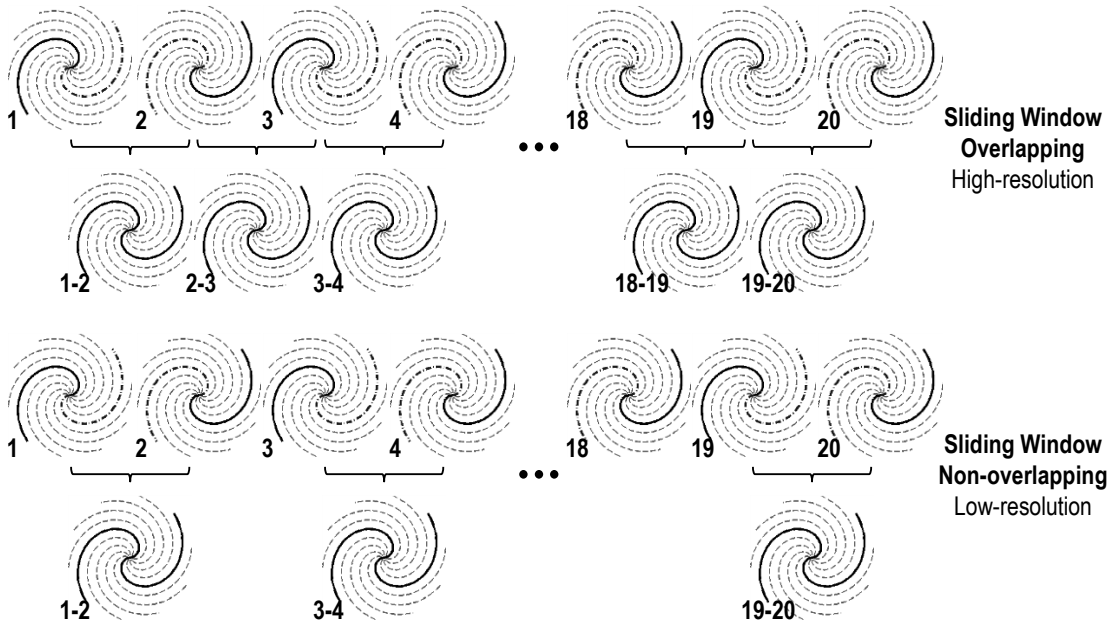

Fig. 6-5 Two sliding window reconstructions used in validation.

acceleration from 10x to 5x, and required the secondary step in form of SENSE. However, none of them needed the newly added temporal filtering step which was omitted or replaced with the necessary data exchange between k-space frames. The rest of the reconstruction was done as presented on Fig. 6-4.

6.2.2 In-Silico simulation

An in-silico simulation was used to determine the optimal filter characteristics for the UNFOLD reconstruction. A high resolution (matrix: 256x256) 2D in-silico model was developed, consisting of a high intensity border representing subcutaneous fat, and an internal medium intensity ellipse representing the ventricular short axis at the mitral valve level (Fig. 6-6 c). Respiratory motion (Fig. 6-6 e) was modelled using a function consisting of expansion (inhalation), a brief pause and contraction (exhalation). The respiratory rate was ~0.22 Hz, with the outer border increasing by ~11 % of its original size. Blood flow velocity at the mitral valve orifice was based on a real mitral valve inflow (MVI) trace acquired using high temporal resolution Doppler echocardiography (Fig. 6-6 a) (SC2000 cardiac ultrasound system, Siemens Healthcare, Erlangen, Germany, pulsed-wave Doppler frequency 1.75 MHz,

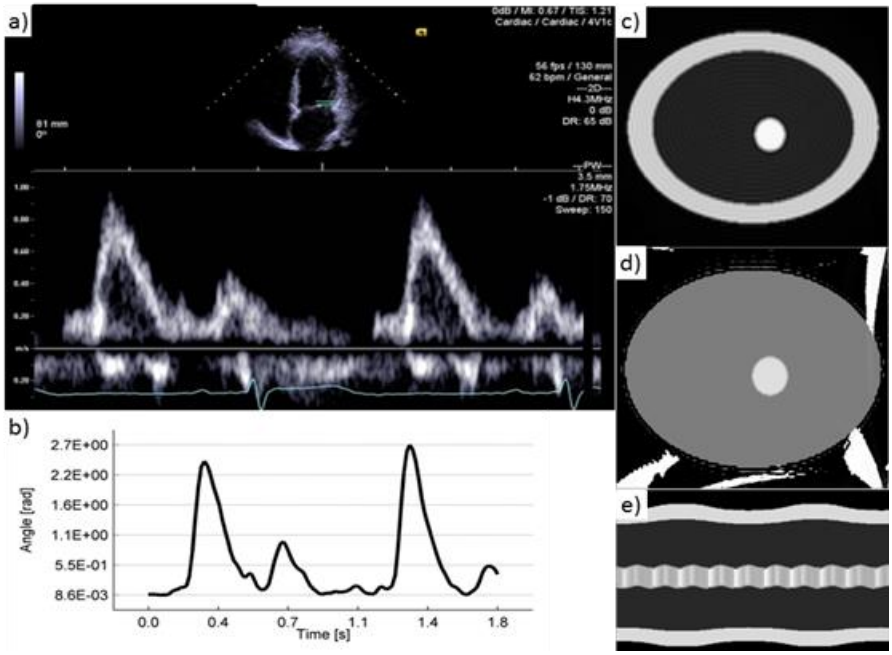

Fig. 6-6 In-silico model design.

a) The captured Doppler echocardiography result of MVI acquisition was used to synthesize b) the new flow curve. The model's simulated mitral valve orifice phase (d) was varied proportionally to this curve. Visual representations of the in silico model; c) a magnitude, d) a phase and e) a cross-section time representation to display motion are presented.

sweep speed: 100–150 mm/s). Next, a temporal frequency spectrum (up to ~76 Hz) of the curve was extracted and used to synthesize a new flow curve of 2 ms temporal resolution (Fig. 6-6 b). The phase and magnitude of the simulated mitral valve orifice varied proportionally to the synthesized flow trace with a maximum phase of ±0.9 π rad (Fig. 6-6 d). Cardiac motion was simulated by sinusoidal translation of the internal ellipse along a diagonal trajectory. The simulated heart rate was matched with the extracted curve period (~1 Hz) and the amplitude of translation was of ~30 % of the internal ellipse size.

This high resolution model (matrix: 256x256, sampling resolution: 2 ms) was used to simulate k-space data acquisitions. The model was down sampled to match the MR acquisition resolution (14.8 ms) and spatial frequency data were extracted on 10 spiral interleaves with a matrix of 128x128 points. The extracted fully sampled data, reconstructed by gridding and inverse Fourier transformation, were used as the reference standard. For the temporal filtering tests, the simulated k-space data were two times under-sampled, rotating between even and odd interleaves. I choose not to simulate the total under-sampling in the sequence as the in-silico model was designed to select the optimal UNFOLD filter. As mentioned, the subsequent SENSE reconstruction's performance may vary depending on the initial conditions. This might have led to sub-optimal filter choice; for example due to imperfections of coil sensitivity simulations.

The under-sampled k-space data underwent the UNFOLD reconstruction with a wide range of temporal filters, along with the sliding window reconstructions for comparison. The filters were defined using a modified version of Tukey filter, which allowed control of the beginning of the cosine lobe as well as its end point. Two filter characteristics were varied: the passband (20-100 % of temporal frequency, in increments of 5 %) and the position of the stopband corners (passband edge – Nyquist frequency, in increments of 5 %). The resultant data were reconstructed by gridding and Fourier transformed into image space. The average phase in the simulated mitral valve orifice for each filter was compared to the equivalent phase data extracted from the reference data. The optimal filter was the one with the lowest normalised root-mean-square error (NRMSE). Also, to assess the level of

temporal blurring introduced by each technique, the up-slop starting points of the two highest flow peaks were calculated and compared.

6.2.3 In-vitro validation study

The in-vitro phantom consisted of a PVC pipe (length: ~12.5 m, internal diameter: 2 cm) surrounded by water and oil bottles (width 400 mm; height 200 mm; Fig. 6-10). Fifteen experiments were performed with different stroke volumes (range: 25 to 45 ml) and heart rates (range: 60 to 130 beats-per-minute) using a pulsatile flow pump (Harvard Apparatus, Holliston, USA). These values were chosen to produce peak mean velocities and ejection times in the normal range seen in humans. Ejection times were compared to simultaneously acquired pressure curves (the reference standard) measured at the same position as the imaging planes using MR compatible pressure transducers (Datex-Ohmeda, GE healthcare, Helsinki, Finland) with 1 kHz sampling frequency. Peak mean velocities measured using the three real-time reconstructions were compared to a reference standard Cartesian PCMR sequence (FOV: 400x300 mm, matrix: 192x144, voxel size: 2.1x2.1x5 mm, TR/TE: 10.7/2.5 ms, temporal resolution: ~10.7 ms, flip angle: 20°). The peak mean velocity and ejection time of the flow curve produced by the pump were analysed in the same way as described below.

6.2.4 In-vivo validation study

Fifteen healthy volunteers (6 male and 9 female) were recruited for this study. The median age was 43 (range 27–66 years). Exclusion criteria were: i) Cardiovascular disease (assessed by clinical history), ii) Contraindications to MR, iii) Cardiac rhythm abnormalities including heart block. The local research ethics committee approved the study and written consent was obtained from all volunteers.

An image plane at the base of the heart was selected so that both the MVI and left ventricular outflow tract (LVOT) were imaged in the short axis. Flow data in this plane were collected using the real-time UNFOLDed-SENSE PCMR (which was also reconstructed using the sliding window reconstructions), and a high-resolution Cartesian gated scan (same parameters as for the in-vitro

study). The optimal temporal filter found in the in-silico tests was used in the UNFOLD step.

For comparison (the reference standard) transthoracic echocardiography was performed immediately prior to or after MR examination (same system as for the in-silico experiment). Pulsed-wave Doppler recordings were acquired separately in the LVOT (apical 5-chamber view) and at the MVI in the apical 4-chamber view.

6.2.5 Exercise study

The imaging protocol was altered to increase the data sampling rate for the exercise study. The imaging parameters were; FOV: 500x500 mm, matrix: 128x128, voxel size: 3.9x3.9x7 mm, TR/TE: 6.58/1.97 ms, flip angle: 15°, complete k-space sampling: 12 interleaves. Consequently, to keep the acquisition rate high with a single interleave per frame, the total acceleration factor was increased to 12x resulting in the sampling rate of ~13 ms. The acquisition block size was kept at 20 frames with six blocks constituting the full k-space coverage needed for the self-referencing SENSE reconstruction. In this case the total number of frames needed to be a multiple of 120 and hence, 480 frames were acquired in ~6.32 s. The same type of temporal filter was used as in the in-vivo validation study.

Ten healthy volunteers (all male) were recruited for this study. The median age was 31.5 (range 21–44 years). Exclusion criteria were: i) Cardiovascular disease (assessed by clinical history); ii) Illness that prevented exercise; iii) Contraindications to MR. The local research ethics committee approved the study and written consent was obtained from all volunteers.

Subjects were scanned prone to facilitate the alternating weighted knee flexion exercise (bilateral 500 g ankle weights) used in this study. Real-time flow data were acquired using the UNFOLDed-SENSE spiral PCMR sequence at rest and during active exercise as soon as the heart rate increased by more than 20 % (as measured using plethysmography). The imaging plane was placed at the base of the heart such that both the MVI and LVOT were imaged in the short axis.

6.2.6 Image analysis

All PCMR data (in-vitro and in-vivo) were segmented using a registration-based algorithm (68) with manual user correction. Mean velocity curves were extracted using in-house plug-ins that I developed, for the OsiriX software (the OsiriX Foundation, Geneva, Switzerland (67)). The velocity curves were sinc interpolated to ~1 ms temporal resolution. MVI mean velocity curves contained early (E wave) and late diastolic waves (A wave), while LVOT velocity curves contained systolic ejection wave (S wave). To calculate the timing intervals the start and end of each wave had to be found. For this task I designed and implemented an automated peak-detection algorithm as a part of the image processing plug-ins (Fig. 6-7). An example of the velocity curves processing is shown on Fig. 6-11. The start and end of each wave were defined by tangent lines calculated at the inflection points of ascending and descending slopes of the wave. The S wave peak was used to divide the trace into separate R-R intervals, which were processed independently. The following cardiac time intervals were measured; ejection time (ET) was the length of the S wave, isovolumic contraction time (ICT) was the time between the end of the A wave and the start of the S wave, isovolumic relaxation time (IRT) was the time between the end of the S wave and the start of the E wave. E and A wave peak

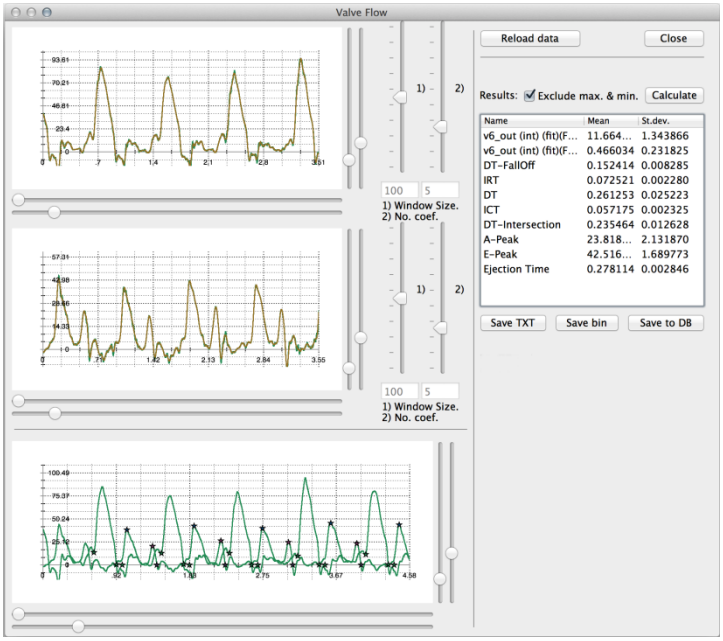

Fig. 6-7 Data processing plug-in for calculation of cardiac time intervals.

The two upper plots allow visualisation of LVOT and MVI extracted from images and used in processing. The bottom plot presents the results; the velocity curves with markers placed at the start, end and peaks of each wave form used in calculations. Numerical values are gathered in the table on the right hand side.

mean-velocities were measured using the automated peak detection and E/A ratio was calculated from these data. Additionally, Tei index (index of myocardial performance; (IRT + ICT)/ET) was calculated.

Doppler echocardiographic data were manually processed and the time intervals were calculated as previously described (75) using both the LVOT and MVI Doppler traces and the concurrently acquired ECG (76).

Estimation of SNR and velocity-to-noise ratio (VNR) in the in-vivo validation data was performed as previously described (56, 77). A region-of-interest (ROI) was drawn in stationary tissue, and estimated noise was calculated as the average standard-deviation of the pixel intensity or velocity through all time frames. Final estimates of SNR were made from the mean signal intensity, and VNR from the mean velocity, within a ROI drawn in the vessel during peak systole, divided by their noise estimates.

6.2.7 Statistical analysis

Results of in-silico tests were compared using the normalised root mean square error (NRMSE – Equation 6-2) metric normalised with respect to the reference data. This allowed a direct comparison and selection of the optimal filter.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(a_i - b_i)^2}{N}} : a_i \in A, b_i \in B$$
$$NRMSE = \frac{RMSE}{b_{max} - b_{min}} * 100[\%] \qquad 6\text{-}2$$

Equation 6-2 Definition of normalised root mean square error used in the in-silico test.

For real-time data, each cardiac cycle was analysed separately and then averaged to produce the final result that was compared to the Cartesian gated PCMR, both in-vivo and in-vitro. Time intervals, peak mean velocities, SNR and VNR were expressed as mean ± standard deviation. Measurements of agreement were performed using Bland–Altman and correlation analysis and differences between means was tested using ANOVA with post-hoc Bonferroni testing. The significance of changes in time intervals and peak mean velocities in response to exercise were tested using paired Student's t-test.

6.3 Results

6.3.1 In-silico tests

The filter comparison results varied from 3.1 maximum to 0.33 % minimum NRMSE. The optimal temporal filter had a passband of 48 % and stopband corners at the Nyquist frequency ~33.7 Hz (–3 dB cut-off frequency at ~25 Hz resulting in ~20 ms temporal resolution) (Fig. 6-8). The NRMSE of this optimal filter was 0.33 %, compared to the SW-HTR NRMSE of 0.66 % and SW-LFR NRMSE of 1.22 %. Fig. 6-9 shows the velocity curves generated by the three reconstructions with the SW-HTR and SW-LTR curves exhibiting temporal blurring that affected their ability to accurately reproduce the starting point of the simulated E waves.

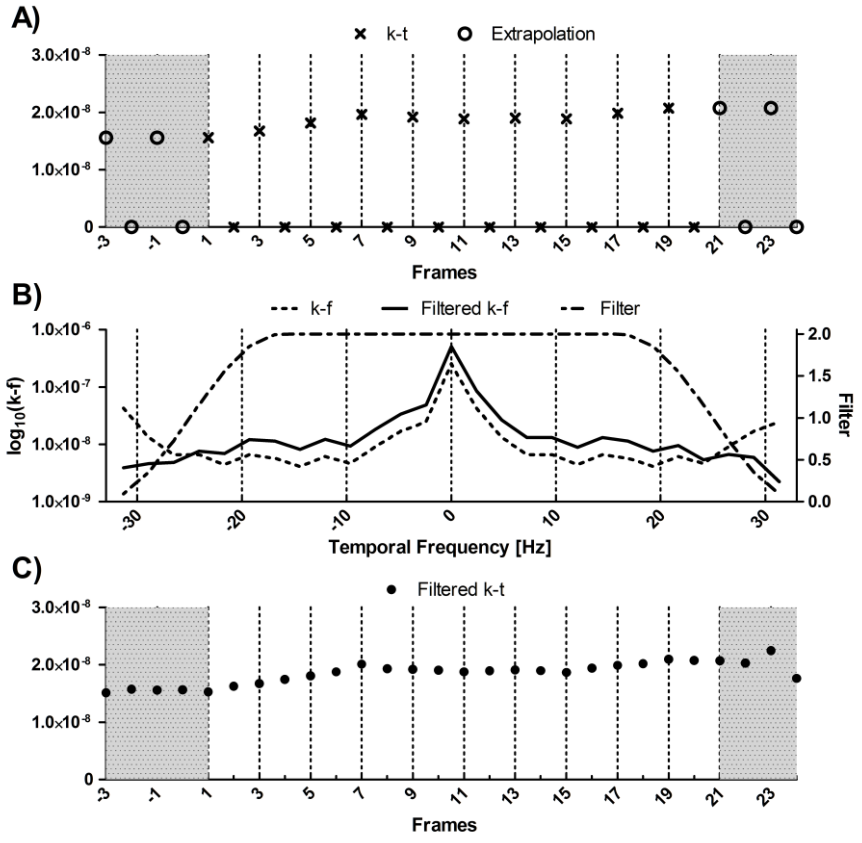

Fig. 6-8 Example of *k*-space temporal filtering for accelerated spiral read-out

A) plot of the magnitude of a *k*-space sample through time (note the two-fold under-sampling); B) temporal frequency of the sample before and after filtering; C) plot of the magnitude of the *k*-space sample through time after the UNFOLD filtering (note that the data is now fully sampled through time). Grey area indicates extrapolated data, which were discarded after UNFOLD to suppress possible ringing artefact.

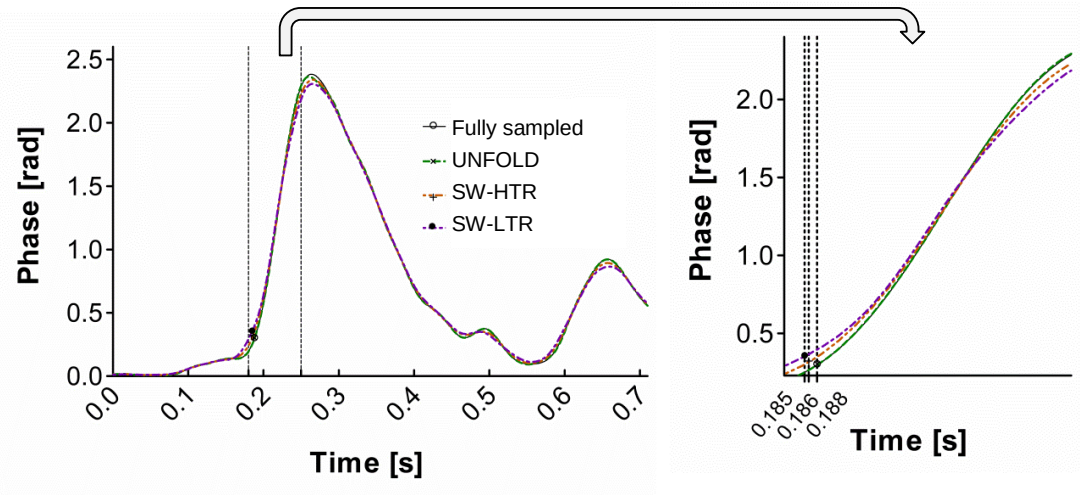

Fig. 6-9 In-silico results.

The average phase curves extracted from simulated mitral valve orifice for all real-time reconstructions. On the right the close-up of the ascending slope of the simulated E wave showing displacement of its starting point at 0.188 s (calculated as the x-intercept of a tangent line through the inflection point) in the SW-HTR (0.186 s) and SW-LTR reconstructions (0.185 s) due to temporal blurring.

6.3.2 In-vitro validation

The results for in-vitro experiments are shown in Tab. 6-1 and Fig. 6-10. The ejection times calculated using the UNFOLDed-SENSE and SW-HTR data were similar to the reference standard pressure measurement, with negligible biases, narrow limits of agreement and excellent correlations. Both the SW-LTR and the gated Cartesian data overestimated the ejection time (positive bias) with wider limits of agreement and poorer correlation. Peak mean velocities from three real-time reconstructions agreed well with the reference standard gated Cartesian PCMR sequence. Nevertheless, UNFOLDed-SENSE performed marginally better. A residual spiral artefact can be seen on the phantom image (Fig. 6-10 a) possibly due to sub-optimal distribution of receiver coils. These spatial artefacts seemed to have no or very little effect on quantitative assessment of flow or time intervals.

6.3.3 In-vivo study

The imaging results are presented in Fig. 6-11, whilst the numerical results are collected in Tab. 6-1 and summarised in Fig. 6-12. There was good agreement for all time intervals (including Tei index) between the UNFOLDed-SENSE results and Doppler echocardiography with negligible biases,

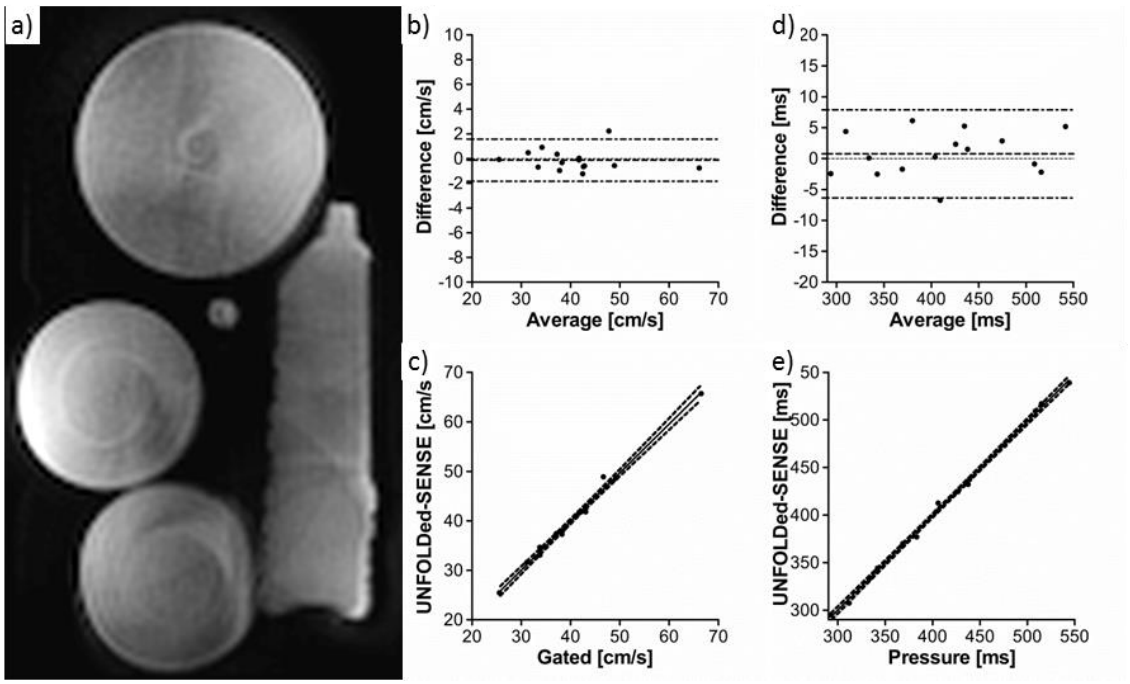

Fig. 6-10 In-vitro validation results.

a) In vitro phantom reconstructed with the UNFOLDed-SENSE reconstruction. The phantom consisted of three doped water bottles, a bottle of oil, and a nondistensible polyvinyl chloride (PVC) pipe placed in middle. b) Bland–Altman and c) scatter plots of UNFOLDed-SENSE mean velocity measurements against the reference standard Cartesian gated results. d) Bland–Altman and e) scatter plots of UNFOLDed-SENSE measured time intervals against the reference standard pressure measurements.

reasonable limits of agreement and good correlations. The SW-LTR and SW-HTR reconstructions performed less well, with increasing biases, and generally wider limits of agreement. Cartesian PCMR performed worst of all with clinically significant biases, wide limits of agreement and poor correlation compared to echocardiography. For assessment of E/A ratio, all the real-time reconstruction performed reasonably well with similar biases, limits of agreement and correlations. However, Cartesian PCMR performed better with negligible bias and narrower limits of agreement and better correlation.

There was no statistically significant ($P > 0.6$) difference between the UNFOLDed-SENSE, SW-HTR or SW-LTR reconstructions in terms of SNR (27.4 ± 7.3, 26.8 ± 8.4 and 28.5 ± 7.6 respectively) or VNR (21.7 ± 5.7, 23.9 ± 6.5 and 23.2 ± 6.0 respectively). SNR and VNR were highest for the Cartesian gated sequence (40.4 ± 8.5 and 32.0 ± 11.8 respectively) and were significantly different from the real-time reconstructions ($P < 0.05$).

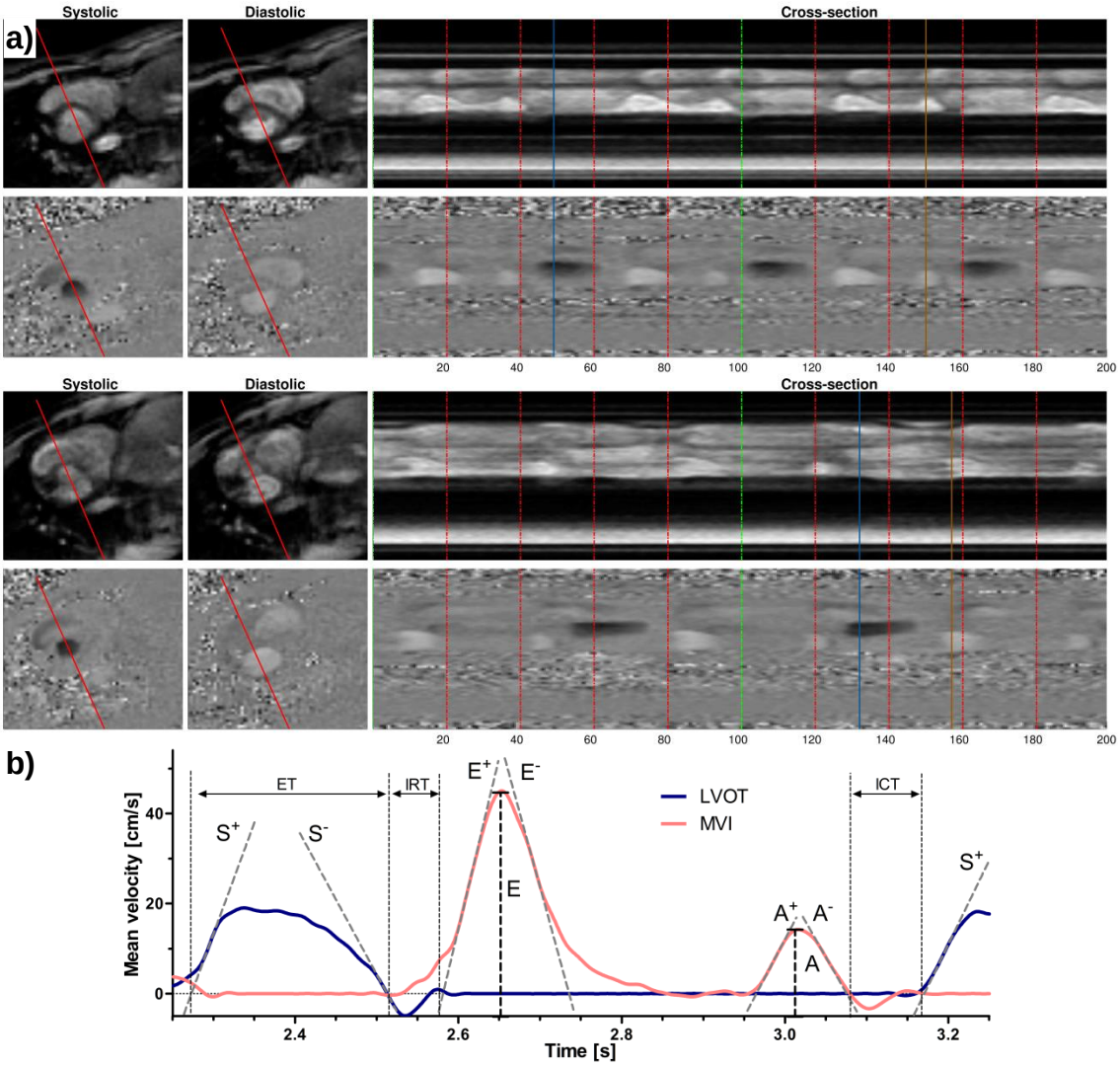

Fig. 6-11 In-vivo imaging results.

a) Examples from two volunteers of magnitude and phase images reconstructed with the UNFOLDed-SENSE reconstruction, at systole, diastole and represented as a cross section (as shown by red line) against time image. The red dotted lines represent transitions between blocks that underwent UNFOLD and the green dotted lines represent the transition between 100 frame sets that underwent SENSE. b) Plot of left ventricular outflow tract (LVOT - blue line) and mitral valve inflow (MVI - pink line) velocity curves. The start and end of the S, E and A waves are delineated by the horizontal axis intercepts of tangent lines drawn on the ascending and descending slopes of the respective waves. These are then used to calculate: isovolumic relaxation time (IRT), isovolumic contraction time (ICT) and ejection time (ET) as shown.

Mean velocity	**Mean [cm/s]**	**Bias[cm/s]**	**Limits [cm/s]**	**Correlation**
Cartesian Gated	40.86±9.07	---	---	---
UNFOLDed-SENSE	40.73±8.96	-0.13±0.84	-1.79 : 1.52	r=0.996
SW-HTR	40.44±8.78	-0.42±0.89	-2.16 : 1.32	r=0.996
SW-LTR	40.37±8.51	-0.49±1.46	-3.34 : 2.36	r=0.988
Time intervals	**Mean [ms]**	**Bias [ms]**	**Limits [ms]**	**Correlation**
Pressure curves	412.59±73.55	---	---	---
Cartesian Gated†	404.94±68.22	7.65±11.15*	-14.22 : 29.51	r=0.990
UNFOLDed-SENSE	411.83±72.98	0.76±3.51	-6.12 : 7.64	r=0.999
SW-HTR	411.90±74.39	0.69±6.83	-12.69 : 14.08	r=0.996
SW-LTR	410.24±69.07	2.35±10.58	-18.38 : 23.09	r=0.991
ICT	**Mean [ms]**	**Bias [ms]**	**Limits [ms]**	**Correlation**
Echo	46.42±12.64	---	---	---
Cartesian Gated†	39.58±12.28	-6.84±8.96*	-24.39 : 10.71	r=0.759
UNFOLDed-SENSE	45.02±11.57	-1.40±5.55	-12.67 : 9.87	r=0.898
SW-HTR	42.45±11.63	-3.97±6.94*	-17.58 : 9.63	r=0.850
SW-LTR	41.82±13.24	-4.60±8.62	-21.49 : 12.29	r=0.794
IRT	**Mean [ms]**	**Bias [ms]**	**Limits [ms]**	**Correlation**
Echo	73.80±14.71	---	---	---
Cartesian Gated†	56.73±12.54	-17.06±13.76*	-44.03 : 9.09	r=0.534
UNFOLDed-SENSE	74.13±9.26	0.33±9.61	-18.51 : 19.17	r=0.793
SW-HTR	71.53±9.33	-2.26±10.58	-23.00 : 18.47	r=0.724
SW-LTR	70.48±9.20	-3.32±11.44	-25.74 : 19.11	r=0.661
Ejection Time	**Mean [ms]**	**Bias [ms]**	**Limits [ms]**	**Correlation**
Echo	301.09±21.43	---	---	---
Cartesian Gated†	310.33±20.89	9.24±17.81	-25.67 : 44.16	r=0.669
UNFOLDed-SENSE	305.99±19.02	4.90±11.73	-18.10 : 27.90	r=0.849
SW-HTR	308.07±19.13	6.98±10.77*	-14.12 : 28.08	r=0.875
SW-LTR†	309.49±19.24	8.40±11.35*	-13.86 : 30.65	r=0.860
Tei	**Mean**	**Bias**	**Limits**	**Correlation**
Echo	0.40±0.06	---	---	---
Cartesian Gated†	0.31±0.05	-0.088±0.069*	-0.226 : 0.047	r=0.378*
UNFOLDed-SENSE	0.39±0.04	-0.010±0.046	-0.099 : 0.080	r=0.721
SW-HTR	0.37±0.05	-0.029±0.053	-0.133 : 0.075	r=0.610
SW-LTR†	0.36±0.05	-0.036±0.058*	-0.150 : 0.077	r=0.523
E/A	**Mean**	**Bias**	**Limits**	**Correlation**
Echo	1.42±0.29	---	---	---
Cartesian Gated	1.47±0.35	0.047±0.245	-0.433 : 0.528	r=0.745
UNFOLDed-SENSE	1.55±0.39	0.128±0.288	-0.435 : 0.692	r=0.698
SW-HTR†	1.57±0.38	0.145±0.283	-0.410 : 0.699	r=0.697
SW-LTR	1.55±0.39	0.127±0.299	-0.459 : 0.713	r=0.681

Tab. 6-1 Combined in-vitro and in-vivo results of Bland–Altman and correlation analyses.

* –fixed bias or insignificant correlation (95 % confidence).
† - significant difference in Bonferroni's Multiple Comparison Test (95 % confidence) against the reference standard measurement.

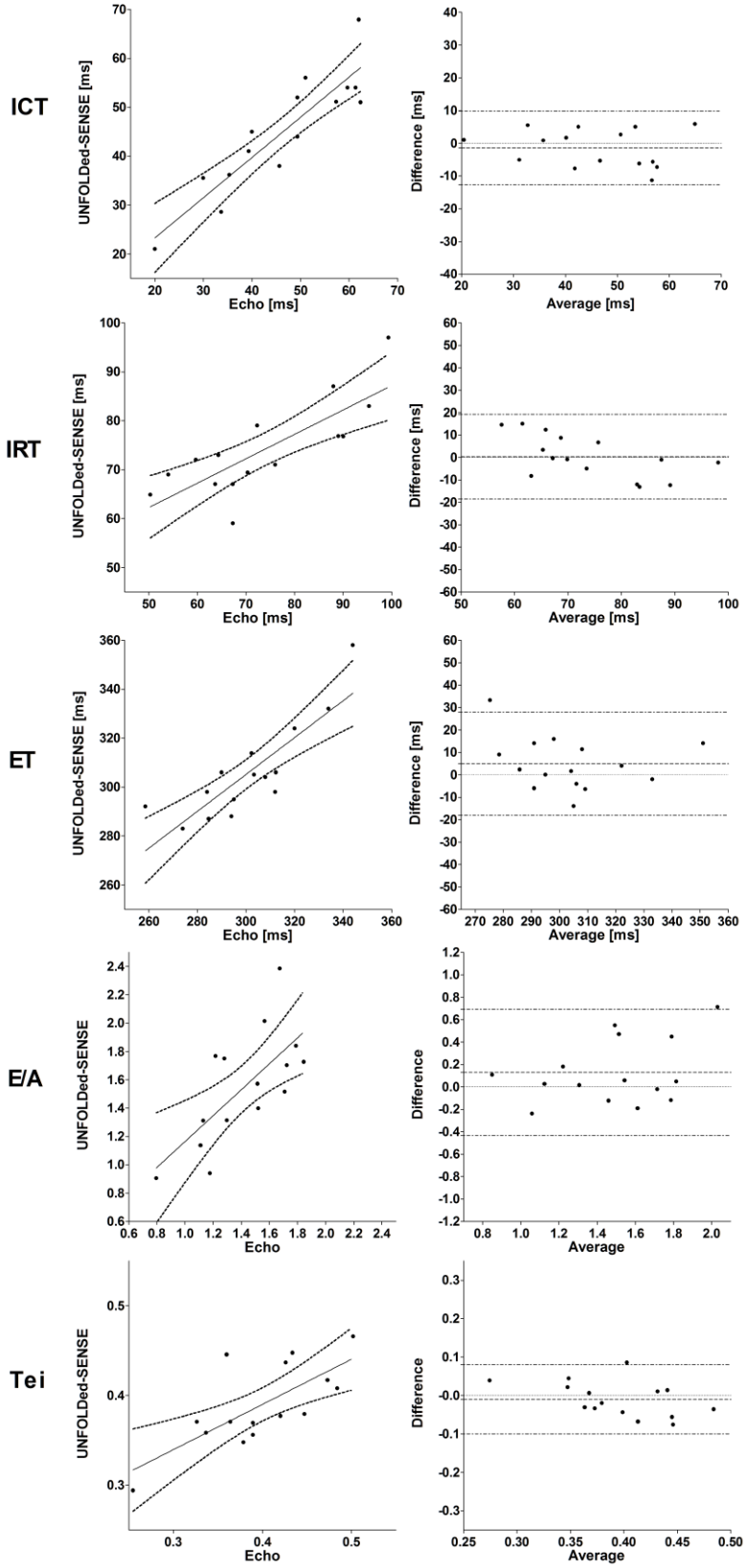

Fig. 6-12 In-vivo validation results.

Bland-Altman and scatter plots of UNFOLDed-SENSE against Doppler echocardiography. All of the measured ventricular function parameters had a good agreement with all negligible biases and acceptable limits of agreement, as well as strong correlations.

6.3.4 Exercise study

All of the volunteers successfully finished the exercise protocol with a mean increase in heart rate of approximately 30 %. MVI and LVOT mean velocity curves allowed calculation of cardiac time intervals in all subjects. All time intervals were significantly lower ($P < 0.05$) during the exercise (Tab. 6-2). However, Tei index was also lower during exercise, demonstrating that cardiac time intervals do not fall proportionally with exercise. Both E and A wave velocities significantly increased ($P < 0.05$) during exercise, although E/A ratio fell ($P < 0.05$) demonstrating an increased reliance on A wave filling.

	HR [bpm]	IRT [ms]	ICT [ms]	ET [ms]	E/A	Tei Index
Rest	66.9±11.3	66.8±8.1	55.9±21.3	276.8±20.1	2.61±0.87	0.45±0.08
Stress	87.6±11.3	57.9±9.5	42.6±19.5	264.8±21.4	1.95±0.58	0.38±0.09
P	2.8E-05	0.0213	0.0307	0.0067	0.0017	0.0339

Tab. 6-2 In-vivo exercise results.

Results (mean ± standard deviation and *P* values of paired two-tailed t-test) of the cardiac time intervals assessment measured before and during the exercise in population of 10 male volunteers: HR – heart rate, IRT – isovolumic relaxation time, ICT – isovolumic contraction time, ET – ejection time, E/A – ratio of E- to A-wave maximum mean-velocity and Tei index.

6.4 Discussion

Currently cardiac time intervals are not routinely assessed using MRI. This is because of the lack of robust and reliable measurement technique. For instance, commonly used cardiac gated PCMR sequences with sufficient temporal resolution would be too time consuming. This study showed that a more suitable approach is to measure the cardiac time intervals using real-time PCMR, and that this approach is superior to gated PCMR. Namely, gated PCMR data is produced by averaging over multiple heartbeats, consequently inter-beat variability results in temporal blurring of flow data and errors during processing. Using real-time PCMR, each heartbeat is processed separately and therefore the raw measurements are unaffected by inter-beat variability. In contrast to processing of the gated PCMR data the averaging of resulting timing intervals from individual R-R waves is advantageous as it strengthens confidence in the result. Also, by using real-time imaging it was possible to acquire data within a few seconds. This strongly limits variation in the results due to the beat-to-beat variability. This was confirmed with the strong

agreement between the UNFOLDed-SENSE and Doppler echocardiography results.

Further benefits of the real-time approach are the ability to use it during physiological interventions such as exercise or forced respiratory manoeuvres, and possibility of extending the assessment time to analyse the beat-to-beat variability due to different physiological processes.

Accurate measurement of cardiac time intervals requires high temporal resolution data. Real-time PCMR imaging with sufficient resolution requires significant acceleration, beyond the level at which SENSE can produce artefact free images. This was the rationale behind expanding the acceleration to the temporal domain. Specifically, the temporal encoding doubles the achievable acceleration, which increased the sampling rate to 13-15 ms (67-77 frames/sec). This is a significantly higher resolution than previously described for real-time PCMR (47, 78, 79) and is a prerequisite for assessment of cardiac time intervals during rest and exercise. Of course this required additional reconstruction steps to combat the temporal under-sampling.

In this study, I designed the comparative experiment to demonstrate the necessity for the high temporal resolution data and increased accuracy with the optimised temporal filtering technique (UNFOLD) in the measurement of cardiac timing intervals. The importance of high temporal resolution can be appreciated by evaluating the results of SW-LTR reconstruction, which was the equivalent of acquiring data with no temporal acceleration (2x lower sampling resolution). The velocity curve from the in-silico test reconstructed with SW-LTR exhibited notable blurring and had the worst NRMSE score from all the tested reconstructions. This translated into blurring of velocity curves and lengthening of ejection times in the in-vitro experiment. Correspondingly, velocity curves extracted in the in-vivo validation using this reconstruction demonstrated significant temporal blurring of the E, A and S waves; resulting in shorter ICT's and IRT's and longer ET's. These results suggested that a higher temporal resolution is required, which could be achieved with the sliding window reconstruction (SW-HTR), which doubled the sampling rate (number of frames) through convolution with the predefined narrow kernel. The simplicity of the operation was its biggest advantage; however its temporal frequency

characteristic of the equivalent filter was inadequate to the frequency spectrum of the imaged object. Consequently, it resulted in temporal blurring visible in the in-silico experiment. Nevertheless, SW-HTR did perform better than the SW-LTR reconstruction. However the best results and closest match with the reference curve were achieved with the optimised temporal filter characteristic enabled with the UNFOLD technique.

The next steps were to robustly combine the optimised UNFOLD technique with the SENSE reconstruction and validate its performance. Although UNFOLD had previously been combined with both SENSE and spiral imaging, I developed a novel implementation that was optimized for the specific needs of real-time acquisitions. Namely, the batched acquisition/reconstruction scheme was applied to enable the coil sensitivity maps to be created from the data itself, while still allowing the temporal encoding. A simpler acquisition plan could be used if the coil sensitivity maps were acquired separately. However, due to possible patient movement and respiratory motion, separate acquisitions are not optimal. Specifically this feature of the acquisition pattern meant UNFOLD was performed independently on consecutive blocks of data. This had two important benefits. Firstly, data undergoing UNFOLD were acquired over a short period of time (263-296 ms) compared to the respiratory period (4–6 s). This limited potential impact of the respiratory motion (39). Secondly, it allowed different interleaves to be acquired in each block and enabled the coil sensitivity maps to be created by combining frames from adjacent blocks. Consequently, the coil sensitivity maps were updated every ~1.5 s (every five blocks) resulting in adaptation and some resistance to patient motion.

It should be noted that the abrupt transitions between the blocks undergoing UNFOLD or the adjacent blocks undergoing SENSE could lead to temporal discontinuities. Although neither was seen in the study, methods could be employed to reduce the impact of these transitions. For instance, the coil sensitivity maps could be continuously updated using a sliding window approach (this would have an impact on reconstruction time).

A less conventional aspect of the reconstruction was performing temporal filtering on the raw k-space data, prior to the SENSE reconstruction. The main reason for this was to better condition the iterative SENSE

reconstruction by removing high frequency noise and improving SNR (33). However, it should be noted that this does require formal testing. The main drawback is that it is not possible to use the support region technique to further improve image quality in static areas of an image (32). Nevertheless, aliasing introduced with 10x under-sampling would cover the whole image space, invalidating the support region technique.

The final reconstruction (UNFOLDed-SENSE) had the highest resolution (sampling resolution: ~14.8 ms, effective resolution due to filtering: ~20 ms) of all the tested reconstructions, which translated into accurate assessment of in-vivo and in-vitro timing intervals with negligible temporal blurring. One of the main limitations of this technique was the lack of fat suppression or water selective excitations, which can lead to image blurring at the tissue interfaces. This was not obvious in the acquired data. A further limitation was the significant reduction in SNR due to high levels of under-sampling, although this did not affect the accuracy of time interval assessment. Potentially, the technique could be improved by using an algebraic reconstruction (80) rather than temporal filtering. This could further improve artefact suppression, particularly potential *ghosting* due to respiratory motion.

Importantly, I showed that this sequence can accurately quantify cardiac time intervals and peak velocity in phantom models and in-vivo. The in-vivo validation results were in keeping with previously published literature (81, 82). Furthermore, the experimental study showed that it is possible to measure the change in these parameters in response to exercise. However, more work is needed to design a better exercise protocol for the evaluation of changes in timing intervals. Nevertheless, the developed imaging technique was a valuable precedence and future protocols can be built on its example.

In conclusion, the modular approach (to the development of reconstructions) allowed reuse of the optimised GPU SENSE implementation in all three tested reconstructions. The incremental development concentrated solely on the *pre-processing* of k-space data in different forms of temporal filtering. The subsequent handling of data stayed the same. The project demonstrated the scalability of the system with the pioneering example of calculating cardiac time intervals with PCMR, for the first time using MRI.

Moreover, the technique allowed performing the comparative exercise study, which laid out the basis for more complex assessment studies. Also, it could provide a valuable tool of assessment of ventricular walls stiffness in elderly patients.

7. GPU reconstruction generalisation

In the chapter, I describe the further development of the GPU implementation of SENSE algorithm. The work resulted in the new version capable to support a wider variety of MR sequences. The work was published in the article:

> *Implementation of a generalized heterogeneous image reconstruction system for clinical magnetic resonance*, GT Kowalik, JA Steeden and V Muthurangu; Concurrency and Computation: Practice and Experience (Special Issue) Volume 27, Issue 6, pages 1603–1611, 25 April 2015 (DOI: 10.1002/cpe.3349);
>
> *Appendix 11.6*

and the proceeding of 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013;

> *Implementation of a Heterogeneous Image Reconstruction System for Clinical Magnetic Resonance*, GT Kowalik, JA Steeden, D Atkinson, A Taylor, V Muthurangu; Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science Volume 8384, 2014, pp 469-479.
>
> *Appendix 11.7*

7.1 Introduction

My previous developments, projects and tests (Chapters 3-5) aimed to remove limitations connected with the fast real-time data acquisition for cardiac MR, for which the long reconstruction times were the most severe. In Chapter 4 the fast GPU based reconstruction of MR data was introduced. Next, its positive impact on the type of protocols available in the clinical and research settings was presented, and evaluated (Chapter 5 and 6).

The developed distributed image reconstruction system proved to enable continuous real-time data reconstructions for data acquired with repeating trajectories. The significant reduction in reconstruction time was achieved by executing the computationally intensive parts of the reconstruction on a GPU. As it was shown, the repetitive nature of the used *k*-space acquisition strategy enabled processing of multiple frames in a single run. This feature of the used sequences was greatly beneficial for the adoption to GPU architecture. However, some applications may require non-repeating sampling strategies. A good example is multi-frame golden angle (137.5°) spiral imaging (56). This acquisition pattern is attractive as it allows reconstruction from the same data with different temporal resolutions. Unfortunately, as no two trajectories are the same, the data structures necessary for reconstruction cannot be reused. This complicates a GPU implementation of these reconstructions. Specifically, the optimisation steps that were used in the initial implementation are rendered inadequate if applied to this more general case of acquisitions with non-repeating trajectories.

It was clear that the developed GPU reconstruction cannot be efficiently applied to all data acquisition patterns without some degree of alterations. An important question was to determine if the benefits of the implementation (most importantly the batched processing and the use of optimised libraries) can be preserved.

In this chapter, I answer this question by evaluating different implementation approaches to solve the problem. In consequence, it led to development of a generalized GPU reconstruction suitable for repetitive and non-repetitive trajectories. I present; i) the further evaluation of the

implementation of SENSE algorithm on a GPU, ii) the necessary modifications to the GPU reconstruction required to support repetitive and non-repetitive trajectories, and iii) the profiling of the final optimized GPU reconstruction, which was implemented within the previously described distributed reconstruction system. The original implementation (Chapter 4) for the repetitive real-time data acquisition was used as the starting point in the optimisation process. Also, the golden angle spiral SENSE reconstruction was selected as a test case, as it represents one of the most challenging examples.

The tests were run on the previously described system (Section 3.4 and 4.6). The same spiral PCMR sequence, as in the transmission tests (Section 3.5), was used to acquire multiple data sets (28, 44, 60, 76 and 92 frame sets). These were used in the tests to assess scalability of different approaches.

The final optimised version of the GPU based SENSE reconstruction was re-evaluated to assess its usability. For this purpose the continuous online reconstruction of accelerated golden angle spiral PCMR sequence was implemented and assessed.

Experimentally, the tests were repeated on different GPU enabled machines to assess performance across different hardware. The computers specifications are presented in Tab. 7-1. The Laptop was used as the development machine, while the Desktop and Work-station machines were also used as production machines in different projects.

	Laptop	*Desktop*	*Work-station*	*Native*
Name	Apple MacBookPro10,1	DELL Alienware Aurora	Workstation Specialists	1.5 T Siemens Avanto
CPU	Intel i7-3820QM	Intel i7-920	Two Intel Xeon E5645	2x Intel Xeon E5440
---- *Clock*	3.7 GHz	2.7 GHz	2.4 GHz	2.8 GHz
---- *Cores*	4	4	2 x 6	2 x 4
---- *Threads*	8	8	2 x 12	2 x 4
---- *Cache*	8 MB	8 MB	2 x 12 MB	2 x 12 MB
Memory	16 GB DDR3	9 GB DDR3	24 GB DDR3	16 GB DDR3
GPU (NVIDIA)	GeForce GT 650M	GeForce GTX 480	Tesla C2075	–
---- *Processor Clock*	0.9 GHz	1.4 GHz	1.2 GHz	–
---- *Memory*	1 GB DDR5	1.5 GB DDR5	6 GB DDR5	–
---- *CUDA cores*	384	480	448	–

Tab. 7-1 Tested hardware specification.

Specifications of computers used during different stages of development and tests of the system, and ones used as image reconstructors during data acquisition.

7.2 Gridding optimisation steps

The major implementation difference, entailed with the transition from repeating to non-repeating trajectories, is in the organisation and preparation of gridding operations. Specifically, the batched optimization (Section 4.5) of gridding is not applicable in situations where the trajectories are non-repeating. Thus, to create a generalised GPU implementation of the iterative SENSE reconstruction, it was foremost necessary to optimize gridding for non-repeating trajectories. In order to do this, a series of tests was run on data acquired with the golden angle acquisition pattern. This acquisition strategy prevented gridding operations from being scheduled as a single batched call across multiple frames. Additionally, the gridding tests were done with the equivalent repeating trajectories for comparison.

This section concentrates on the description of the optimisation steps that led to the final gridding implementation, and were based on results from the production computer (Tab. 7-1 Work-station). The results from all tested machines and comparisons are presented in the Appendix 11.5.

7.2.1 Initial assessment

An important step of the process was to define the acceptable execution time boundaries and resulting from them, the maximum achievable

speed-up with the GPU gridding. This laid out a context within which the new implementations were assessed. The maximum time (*Upper-limit*) was defined as the time it took for the native scanner multi-core CPU to grid frames acquired using the non-repeating trajectory. The minimum time (the *Lower-limit*) was defined as the time needed for the GPU to grid the same amount of data, but acquired on a repetitive trajectory (the initial batched implementation of gridding). I considered these were valid assumptions, as it was unlikely that the implementation for non-repetitive trajectories would be quicker than the previously described GPU implementation. Also, execution times longer than the existing on-scanner implementation would be unacceptable.

The measured boundaries were found to be linearly related to the number of frames within the tested range. This allowed calculation of the averaged time (boundaries) per data frame. On the scanner, a single gridding operation for the non-repeating trajectories (the *Upper-limit*) took approximately 17.76±1.06 ms. The external GPU implementation needed 0.95±0.00 ms to grid a single frame from the repeating set of trajectories (the *Lower-limit*). Consequently, the maximum attainable speed-up for the gridding of the non-repeating trajectories on the GPU was estimated as ~19x, as compared to the *Upper-limit*. Alternatively, when compared to the *Acquisition time* (the time needed to acquire the whole set of frames) the achievable speed-up was estimated as ~43x.

The *Acquisition time* and the *Upper-limit* were the same in all tests, as they are related to the sequence and scanner reconstructor performance.

The initial timing tests showed that the sequential scheduling of individual gridding operations yields the same results (0.96±0.00 ms to grid a frame) as the batched gridding (the *Lower-limit*), providing the whole data structures were already available in the GPU memory.

Tab. 7-2 presents GPU memory requirements for the gridding operation. ~11 MB was needed to accommodate a single gridding operation, from which ~32 % was needed for a gridding matrix. However, in the worst case of non-repeating trajectories the matrix cannot be re-used and the ~32 % overhead quickly becomes problematic. This is especially important when

considering buffered real-time reconstruction, as a service for multiple incoming requests.

Receiver Coils		*12*					
Acceleration		*4*					
Matrix		*128x128*					
Read-out samples		*2300*					
PCMR encodings		*2*					
Frames		*1*	*28*	*44*	*60*	*76*	*92*
Spiral Trajectory	*[MB]*	***1.33***	*37.16*	*58.39*	*79.63*	*100.86*	*122.09*
Cartesian Grid	*[MB]*	***5.90***	*165.15*	*259.52*	*353.89*	*448.27*	*542.64*
Gridding Matrix	*[MB]*	***3.40***	*95.24*	*149.66*	*204.08*	*258.51*	*312.93*
Total	***[MB]***	***10.63***	***297.55***	***467.58***	***637.61***	***807.63***	***977.66***

Tab. 7-2 GPU memory requirement of the gridding operation.

The table presents how much GPU memory is needed to enable the gridding operation. The results were calculated for spiral PCMR data acquired with the presented parameters. The total value is a sum of space needed to store the data on the spiral trajectory, the result in form of Cartesian grid and the gridding matrix stored in the sparse format.

This initial assessment confirmed the original GPU implementation as applicable to the non-repeating trajectories, but impractical due to excessive GPU memory use. Consequently, a better data structures management was needed to reduce memory usage without performance loss.

7.2.2 Sequential approach

My first step was to evaluate a simple sequential strategy where a single CPU control thread creates gridding matrices and then calls the multi-threaded gridding in form of matrix-matrix multiplications on the GPU (Fig. 7-1). In this *naive* implementation no parallelism on the CPU side is exposed and the control thread waits for the gridding operation to finish. To reduce the memory consumption, the sequential approaches allowed only a single gridding matrix to be stored on the GPU. The algorithm sequentially created a matrix in CPU memory related to each frame, which then was used to replace the one in GPU memory.

A possible optimization of this *naive* approach is to allow the creation of the next gridding matrix to occur while gridding of the current data is being executed. Additionally, declaring more than one gridding matrix object and rotating between them allows an overlap between creation and execution. Of course, execution on the GPU is always asynchronous and the CPU needs only

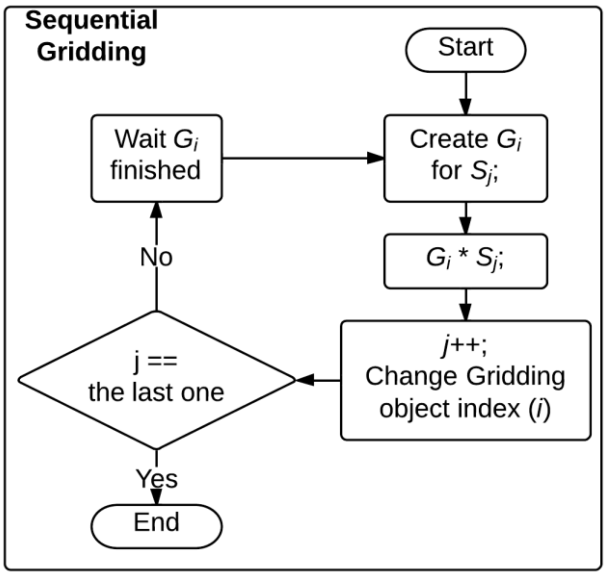

Fig. 7-1 Sequential approach to gridding of data from non-repeating trajectories.

The single CPU control thread in sequential fashion goes through all of the data frames (S_j) repeating the creation of a gridding matrix for that data set (G_i) and scheduling of multiplication. The algorithm rotates between available gridding objects to allow the overlap between the creation of a matrix for the presently processed data and the multiplication with the previous data.

to wait until the gridding matrix is transferred before starting calculation of the next one. In this case, the overlap can be used to hide the transmission time.

These simplistic implementations did not perform well; however, it allowed assessment of the average times of the basic operations constituting the GPU gridding. These times were fixed across the test sets (linear relation to a number of frames), as only the number of frames varied rather than the type of a trajectory. These operations are; *Creation* of a matrix (32.05±0.81 ms), single frame *Gridding* operation as a matrix multiplication on GPU (1.03±0.00 ms) and *Copy* – transfer of data structures on a GPU (0.71±0.05 ms). The average times were calculated from all the *sequential gridding* tests results.

The tests revealed a noticeable difference between these estimated gridding times of a single frame (~1.03 ms) and those from the initial assessment (~0.96 ms). It suggests that the adjacent gridding operations (as in contrast to ones interleaved with data transmissions – the *sequential gridding* tests) were able to achieve some degree of overlap, which levelled the cumulative time with the *Lower-limit*. After correction for these gridding times the maximum achievable speed-up changed to ~17x.

The overlapping approach is most efficient if both stages take the same amount of time. Specifically, unless creation of the gridding matrix is faster or equal to data transmission onto the GPU, there will be a delay between consecutive matrix multiplications. For the overlapping version to make a difference the ratio between the creation and transmission times would need to be very close to one. Unfortunately, the creation was up to ~45x slower in the tests.

The *naive* strategy showed very poor results which were ~33x slower than expected (as compared with the *Lower-limit*). The overlapping version was only marginally better. The improvement was connected to a slight shift in the creation of matrix times with which the final timing results were driven. In all cases the serial matrix creation took >98 % of the total time.

Consequently, the total processing time was reduced to the time needed to create all of the matrices in serial fashion. In fact, my tests showed that optimization by overlapping had no or negligible impact on the sequential version of the GPU gridding. This was because the data transmission time was short enough to be hidden with an overhead introduced with scheduling of the consecutive GPU matrix multiplication calls.

In order to gain more insight into the performance of the sequential scheduling of operations onto a GPU the tests were repeated using pre-calculated matrices stored in the CPU memory. The goal was to find out if the data transfer could be overlapped with the longer matrix multiplications on GPU. To facilitate the overlapping execution I made use of *CUDA streams*. These can be seen as processing pipelines within which tasks are run sequentially. However, tasks across different *streams* can overlap providing a GPU has enough hardware resources. The process involved four tasks; three data transmissions and one matrix multiplication for each gridding. This was because in this version the non-zero values and their row, column indices for sparse matrices were stored separately.

The timing results were ~0.4 ms/frame longer than the batched gridding (the *Lower-limit*) resulting in ~1.4x slow-down. Comparison of the results with the transmission times (~0.7 ms) revealed that only a portion of the transmission time was overlapped with the matrix multiplication operation. This

could be caused by an overhead time due to the GPU kernel launch scheduling or the partitioned transmission of each gridding matrix. The former cause would be connected with the CPU time needed to arrange consecutive operations. This can be an issue if a GPU kernel call takes longer than its execution on a GPU. The latter would suggest only the adjacent operations could be overlapped, as the total transmission time was shorter than the gridding operation. Further addressing of this issue and a consequent removal of the time difference would level the total time with the *Lower-limit*. However, the time necessary for the pre-calculation of matrices would need to be added to the final result, which would make it longer than the previously discussed overlapping version.

These tests revealed that creation of matrices is the major limitation of the new gridding approach for data on non-repeating trajectories. The execution times were dictated by the speed of CPU, as the GPU execution was short enough to be hidden by it. In fact, the sequential gridding was only marginally (~1.3x) quicker than the acquisition of data and significantly (~1.8x) slower than the Upper-limit.

7.2.3 Threaded approach

Creation of a gridding matrix requires preparation of weights assigned to each k-space sample and connected with them indices describing the matrix in sparse format. Preparation of the indices is done in a serial fashion, as they depend on the number of points included in each convolution, which vary depending on the position of the sample. However, creations of individual matrices are independent of each other and can be run in parallel. Consequently, a straightforward optimization is to spread gridding matrix creations between N threads, so that each thread performs the two stages considered on a different frame, thus allowing the parallel processing of N frames (Fig. 7-2). The speed-up would be equivalent to the number of concurrently running calculations. This is strictly hardware dependent, as there may be not enough resources for each thread to run without interruptions.

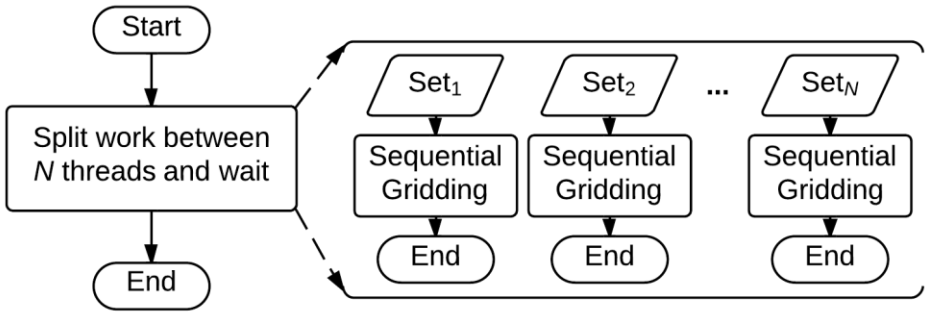

Fig. 7-2 Threaded approach to gridding of data from non-repeating trajectories.

A single CPU control thread splits the work between N sequential gridding tasks that are run in parallel.

For this implementation, the optimal number of concurrent matrix creations is equal to the ratio of the time needed to create a matrix to the time needed to execute gridding on the GPU (~31x). Providing enough computing resources were available the number of concurrently created matrices could be brought to the level sufficing to fully occupy the GPU with work. This way a set of matrices would be prepared while the previous one was used. Consequently, both the CPU and GPU would be fully utilized and the final processing time would equal the total GPU time plus initial creation of the first set of matrices. However this condition was impossible to meet as the tested hardware (Tab. 7-1) was limited to 12 physical processors.

In the tests, apart from using different number of frames, the number of CPU threads was varied as well. This was done to find the optimal partitioning of the task for the tested hardware.

The tested CPU was capable of hyper-threading (mapping of two virtual processors into one physical); however the tests showed the achieved speed up (~9x) was significantly lower than expected and would not exceed the number of physical processors. Also, this was reflected in an increase of the average matrix creation time (~45.1 ms). Collected log files showed that some runs were able to achieve the creation times similar to those found in the sequential approach, but the rest required significantly longer time (about double the time). This behaviour was observed regardless of the number of concurrently scheduled threads and increased with total number of matrix creations. Presumably, execution of some threads assigned to the same physical processor caused an interrupted execution due to insufficient computing

resources. Consequently, the total time needed to create all of the matrices was greater than expected based on the total number of threads involved in the task.

Finally, I assessed if concurrent scheduling of gridding on the GPU with pre-calculated matrices would perform better than the equivalent sequential approach. As before, the tests were repeated using pre-calculated matrices and the resultant times were in perfect agreement with the estimated total time of gridding (difference: 0.00±0.03 ms, ratio: ~1.0x). The average gridding times (found in the *sequential* gridding test) were used as a reference to estimate the expected processing time (the single gridding operation time on GPU multiplied by a number of repetitions) in this test.

The *threaded* results suggested that all the data transmission operations were almost completely hidden with the longer matrix multiplication operations, as in contrast to the *sequential* test. Presumably, the multithreaded scheduling of GPU operations gave rise to more beneficial ordering, resulting in better utilisation of the GPU.

However, comparing with the batched execution (the *Lower-limit*) the difference in time was 0.09±0.02 ms resulting in ~1.1x slow down. This can be related to the difference between the gridding times per frame found for the batched gridding (~0.95 ms) and one acquired in the *sequential* gridding tests (~1.03 ms).

For non-repeating trajectories the gridding operations on the GPU had to be interleaved with the data transmissions, as in contrast to the batched gridding or the initial assessment from the *sequential* tests – where the data structures were pre-stored on the GPU. Consequently, even if fully overlapped with the data transmission operations, the total gridding time could not achieve its minimum. Assuming the data structures were available in the CPU memory (neglecting the creation time), the *threaded* approach would achieve almost the maximum desirable speed-up (~17x).

7.3 Hybrid CPU/GPU implementation

For the threaded optimization strategy the limiting factor was the hardware on which the algorithm runs. Providing a sufficient number of CPU

cores capable of uninterrupted work (31 in the tests) were available, full overlap between the creation of matrices and execution of gridding could be achieved. In the tests this was not possible and resulted in suboptimal processing time. As shown, providing the gridding matrices were readily available the processing time would be the closest to the batched execution. Gridding matrices (gridding kernel coefficients) can be *hard coded* into the application if the sampling trajectory type and order of acquired lines were known; removing the need of their calculation. However, to build a tool for generalized reconstruction, no prior knowledge of the application can be assumed. Effectively, the necessity of re-creating the gridding matrices must be assumed. Nevertheless, the image reconstruction works in an iterative way in which the gridding operations are repeated multiple times for a set of reconstructed images. Thus, matrices created during the first iteration can be stored and reused in following iterations, assuming there is sufficient space. Consequently, I decided to build both versions of the threaded approach into the new iterative SENSE reconstruction. The new version differed only in the way gridding operations were organised. Depending on the iteration index the control thread launched a set of worker threads that carried out the gridding operations in parallel by creating the necessary gridding matrices or reusing the one stored in CPU's memory. The rest of the SENSE reconstruction stayed as previously described (Section 4.6).

The new algorithm forces partitioning of reconstructions for which data structures cannot fit into memory. Separate reconstruction of sets of frames could extend the reconstruction time even if the reconstruction time per frame stayed unchanged. Consequently, the set size (number of reconstructed frames) needed to be optimised to keep the reconstruction time below the acquisition time. The new implementation was tested with different set sizes to find out if the reconstruction times were faster than the frames acquisition and to assess the relation of the reconstruction time to the set size.

Tab. 7-3 presents the comparison of the new GPU reconstruction with the original CPU version run on the native scanner reconstructor. The table comprises timing results of different steps of the iterative algorithm; Fast Fourier Transformations (FFT), combination with coil sensitivity maps (CSM), gridding and *other* (it refers to element-wise operations, preconditioning and regularisation) were distinguished. Also, the averaged total reconstruction and

iteration times are presented. The results were divided into timings per set of frames and the average per frame. The *Ratio* marks the achieved speed-up as compared to the native image reconstructor. The Laptop machine was unable to run tests with 76 and 92 frames due to an insufficient amount of memory.

All tests showed a linear relation with the size of the reconstructed data. Proportions between the reconstruction steps were preserved across all of the data sets. As presented, the combination with coil sensitivity maps and gridding operations were the bottlenecks of the original CPU reconstruction. These accounted to ~81 % of each iteration time.

Not surprisingly, of the external reconstructors, the Laptop performed the worst achieving a total ~5x speed up. This was more than two times slower than the Desktop (~12x speed up) and almost three times slower than the Work-station machine (~15x speed up).

The reconstruction times, calculated per frame, were compared with the acquisition rate. For the Work-station and Desktop machines, the very high acceleration meant the reconstruction time constituted only ~55 % and ~68 % of the acquisition time, respectively.

The time balance between the reconstruction stages was shifted with the GPU reconstructions, as compared to the CPU reconstruction. All of them were significantly sped up, with the most visible improvement in the CSM operations (~52x, ~55x and ~8x for the Work-station, the Desktop and the Laptop respectively) which clearly demonstrates GPU's efficiency in running multiple simple arithmetic operations. The gridding was also significantly sped up ~15x, ~12x and ~5x for the Work-station, the Desktop and the Laptop respectively. However this was not sufficient to reduce its impact on the total time, which similarly to CPU's was 36-42 %. Nevertheless, the aggregate ratio of the CSM and gridding was reduced to ~50 %, ~53 % and ~67 % for the Work-station, the Desktop and the Laptop respectively.

		Total[ms]/ Frame[ms]/ [%]	*Ratio [%]*	*Total[ms]/ Frame[ms]/ [%]*	*Ratio [%]*	*Total[ms]/ Frame[ms]/ [%]*	*Ratio [%]*	*Total[ms]/ Frame[ms]/ [%]*	*Ratio [%]*	*Total[ms]/ Frame[ms]/ [%]*	*Ratio [%]*	Average
Number of frames		28		44		60		76		92		*Frame[ms]/[%]* ***Ratio [%]***
CSM	Native	647.6/23.1/44		1004.8/22.8/46	-	1357.6/22.6/46	-	1712.2/22.5/46	-	2065.1/22.4/46	-	22.7±0.2 / 45.6±0.9 -
	Work-station	12.3/0.4/13	**52**	19.3/0.4/13	**52**	26.3/0.4/13	**52**	33.3/0.4/13	**51**	40.3/0.4/13	**51**	0.4±0.0 / 13.3±0.1 **51.8±0.4**
	Desktop	11.6/0.4/12	**56**	18.1/0.4/12	**55**	24.7/0.4/11	**55**	31.2/0.4/11	**55**	37.7/0.4/11	**55**	0.4±0.0 / 11.1±0.4 **55.2±0.4**
	Laptop	79.6/2.8/29	**8**	124.0/2.8/29	**8**	168.5/2.8/29	**8**	-/-/-	-	-/-/-	-	2.8±0.0 / 29.5±0.1 **8.1±0.0**
FFT	Native	200.1/7.1/14		295.3/6.7/14	-	410.0/6.8/14	-	496.1/6.5/13	-	597.6/6.5/13		6.7±0.2 / 13.5±0.2 -
	Work-station	32.7/1.2/35	**6**	51.5/1.2/35	**6**	70.5/1.2/36	**6**	89.7/1.2/36	**6**	109.1/1.2/36	**5**	1.2±0.0 / 35.6±0.5 **5.7±0.2**
	Desktop	19.3/0.7/19	**10**	30.4/0.7/19	**10**	41.4/0.7/18	**10**	53.1/0.7/19	**9**	64.6/0.7/18	**9**	0.7±0.0 / 18.7±0.6 **9.7±0.4**
	Laptop	52.2/1.9/19	**4**	81.6/1.9/19	**4**	111.3/1.9/19	**4**	-/-/-	-	-/-/-	-	1.9±0.0 / 19.4±0.1 **3.7±0.1**
Gridding	Native	555.1/19.8/38		744.4/16.9/34	-	1040.9/17.3/35	-	1300.9/17.1/35	-	1616.4/17.6/36	-	17.8±1.1 / 35.6±1.1 -
	Work-station	33.5/1.2/36	**17**	52.9/1.2/36	**14**	71.4/1.2/36	**15**	92.8/1.2/38	**14**	112.7/1.2/37	**14**	1.2±0.0 / 36.6±0.6 **14.7±1.0**
	Desktop	42.2/1.5/43	**13**	67.0/1.5/42	**11**	91.3/1.5/40	**11**	117.2/1.5/41	**11**	144.3/1.6/41	**11**	1.5±0.0 / 41.4±1.1 **11.6±0.8**
	Laptop	103.0/3.7/38	**5**	162.0/3.7/38	**5**	221.0/3.7/39	**5**	-/-/-	-	-/-/-	-	3.7±0.0 / 38.4±0.2 **4.9±0.3**
Other	Native	48.9/1.7/3		77.0/1.8/4	-	104.8/1.7/4	-	133.0/1.7/4	-	160.9/1.7/4	-	1.7±0.0 / 3.5±0.1 -
	Work-station	2.7/0.1/3	**18**	4.3/0.1/3	**18**	5.8/0.1/3	**18**	7.4/0.1/3	**18**	8.9/0.1/3	**18**	0.1±0.0 / 2.9±0.0 **18.0±0.0**
	Desktop	3.9/0.1/4	**13**	7.0/0.2/4	**11**	10.6/0.2/5	**10**	14.5/0.2/5	**9**	19.7/0.2/6	**8**	0.2±0.0 / 4.7±0.6 **10.2±1.5**
	Laptop	7.1/0.3/3	**7**	11.1/0.3/3	**7**	15.4/0.3/3	**7**	-/-/-	-	-/-/-	-	0.3±0.0 / 2.6±0.0 **6.9±0.0**
Iteration	Native	1475.7/52.7/-		2162.8/49.2/-	-	2967.4/49.5/-	-	3709.9/48.8/-	-	4520.2/49.1/-	-	49.9±1.4 / - -
	Work-station	93.6/3.3/-	**16**	145.6/3.3/-	**15**	197.6/3.3/-	**15**	247.0/3.3/-	**15**	304.2/3.3/-	**15**	3.3±0.0 / - **15.1±0.3**
	Desktop	99.1/3.5/-	**15**	157.6/3.6/-	**14**	230.4/3.8/-	**13**	283.3/3.7/-	**13**	353.6/3.8/-	**13**	3.7±0.1 / - **13.5±0.8**
	Laptop	270.0/9.6/-	**5**	422.0/9.6/-	**5**	571.5/9.5/-	**5**	-/-/-	-	-/-/-	-	9.6±0.0 / - **5.3±0.1**
Total time	Native	10335.8/369.1/-		15148.9/344.3/-	-	20782.9/346.4/-	-	25983.3/341.9/-	-	31657.7/344.1/-	-	349.2±10.1 / - -
	Work-station	639.6/22.8/-	**16**	1029.6/23.4/-	**15**	1372.8/22.9/-	**15**	1716.0/22.6/-	**15**	2090.4/22.7/-	**15**	22.9±0.3 / - **15.3±0.5**
	Desktop	806.2/28.8/-	**13**	1170.2/26.6/-	**13**	1702.5/28.4/-	**12**	2079.5/27.4/-	**12**	2745.6/29.8/-	**12**	28.2±1.1 / - **12.4±0.5**
	Laptop	1932.6/69.0/-	**5**	2803.1/63.7/-	**5**	3801.2/63.4/-	**5**	-/-/-	-	-/-/-	-	65.4±2.6 / - **5.4±0.0**

Tab. 7-3 Iterative SENSE GPU reconstruction test results.

Laptop was unable to run tests with set of 76 and 92 frames due to not sufficient memory space.

7.4 System workload tests

The early tests (Section 3.5 and 4.7) showed that the data transmission and reconstruction can be done faster than data acquisition. This was confirmed with the online continuous assessment reconstruction tests (Section 5.3.2). As stated, the use of the batched gridding strategy was highly dependent on the application, as it can be only used if all, or the majority of read-outs, are on repeating trajectories. Also, these are ideal situations and to test the system in a more general, demanding condition, a continuous acquisition with all non-repeating trajectories needed to be assessed.

The new set of tests for the generalised reconstruction (Section 7.2 and 7.3) showed the same dependency between the reconstruction and acquisition times. The last test was to assess suitability of the new online GPU reconstruction for the use in clinical/research setting. As in the previous tests, the real-time spiral PCMR sequence was used to acquire data using a trajectory continuously rotated by the golden angle. The new reconstruction was installed in the distributed reconstruction system. All three external reconstructors (Tab. 7-1) were connected with the scanner and tested using the same implementation (Section 3.4) of the GPU based SENSE reconstruction (Chapter 4) service. The simplified data flow in the system was presented in Fig. 4-4 and was discussed in Section 4.6. The processing of the continuous stream of data did not change and was done as presented in Section 5.2.2. To allow the overlapping acquisition and reconstruction, the incoming stream of continuous data was divided into blocks that can be transferred and processed separately. The aim was to find out if the waiting time after the acquisition finishes and data being available on the scanner would increase with the length of acquisition or stay constant as in the previous test (Section 5.3.2). In the latter case, as long as the transmission and reconstruction time of a block were faster than its acquisition, it would not matter how many blocks needed to be processed. On the other hand, if they proved slower, the tests with small number of blocks would suffice to ascertain it. Consequently, the number of continuously acquired frames was set to five times the size of a block (reconstruction buffer), in each test.

As in the previous tests, five sizes (28, 44, 60, 76 and 92) of data blocks were used in tests to determine the impact on performance. The timing results are presented in Tab. 7-4. As in the conjugate gradient tests (Section 7.3), the size of data proved problematic not only to the Laptop but also to the Desktop computer. Reconstructions for blocks of 76 and 92 failed to run on the Laptop due to in-sufficient memory for all the necessary data structures. The 60 frames block test for the Laptop and the 76 and 92 frame block tests for the Desktop did run, but did not allow fully unimpeded buffered reconstruction. One or both buffers were limited in available space for temporal processing structures, restricting some operations. Most notably the FFT on the GPU was forced to switch from batched to serial processing resulting in unacceptable extension of the reconstruction time.

Timing results were collected in the form of time-stamps, recorded on the client side. These were used in assessment of the system work-load across different block sizes and external reconstructors (Fig. 7-3). In all cases, external reconstructions of the first two blocks proved to be significantly longer than the last three. This is explained with the necessary initialisation and memory allocations that must be done once per each reconstruction buffer. These are time consuming operations that caused ~1.3x to ~3.6x increase in the expected reconstruction time (as calculated in Tab. 7-3). Fortunately, the subsequent runs with the initialised buffers showed to be more reliable in execution time. This can be observed in the time plots (Fig. 7-3) and numerical values (Tab. 7-4). However, even then the measured external reconstruction time was on average ~31 % longer for the Work-station and Desktop machines, and ~13 % for the Laptop, as compared to the time needed to run the SENSE reconstruction (Tab. 7-3). This additional time on top of the SENSE reconstruction, is spent on the preparation of coil sensitivity, preconditioning and regularisation maps, as well as organisation of data. This is not an insignificant amount of time which may become a bottleneck of the reconstruction for more demanding applications. The Laptop's performance was already shown insufficient to allow full overlap with the acquisition, which can be seen in the plots (Fig. 7-3). Even with the additional overhead, the Work-station and Desktop's averaged reconstruction times were still faster than the acquisition time.

For tests with the Work-station and Desktop machines, the transmission speed exceeded the acquisition speed (~32 MB/s) as expected, which resulted in a block transmission time being faster than or equal to the corresponding data acquisition. For the Laptop tests, the transmission time measurements showed poorer than expected performance. This was secondary to the slow reconstruction consuming the processing time.

Overall, it can be seen that the continuous assessments, which are un-restricted by acquisition time, would be possible with the described GPU reconstruction within the online distributed system using either the Work-station or Desktop computers. The initial delay, due to the initialisations, would resolve over time as both the transmission and external reconstructions were much faster than the acquisition. This would result in the waiting period measured from the end of acquisition to become constant and depend solely on the image resolution parameters and the block size.

Based on these final test conclusions, I showed that the assessment of strategies for generalised gridding (Section 7.2) resulted in the implementation that had the least impact on the original implementation and connected with it performance benefits (Section 7.3).

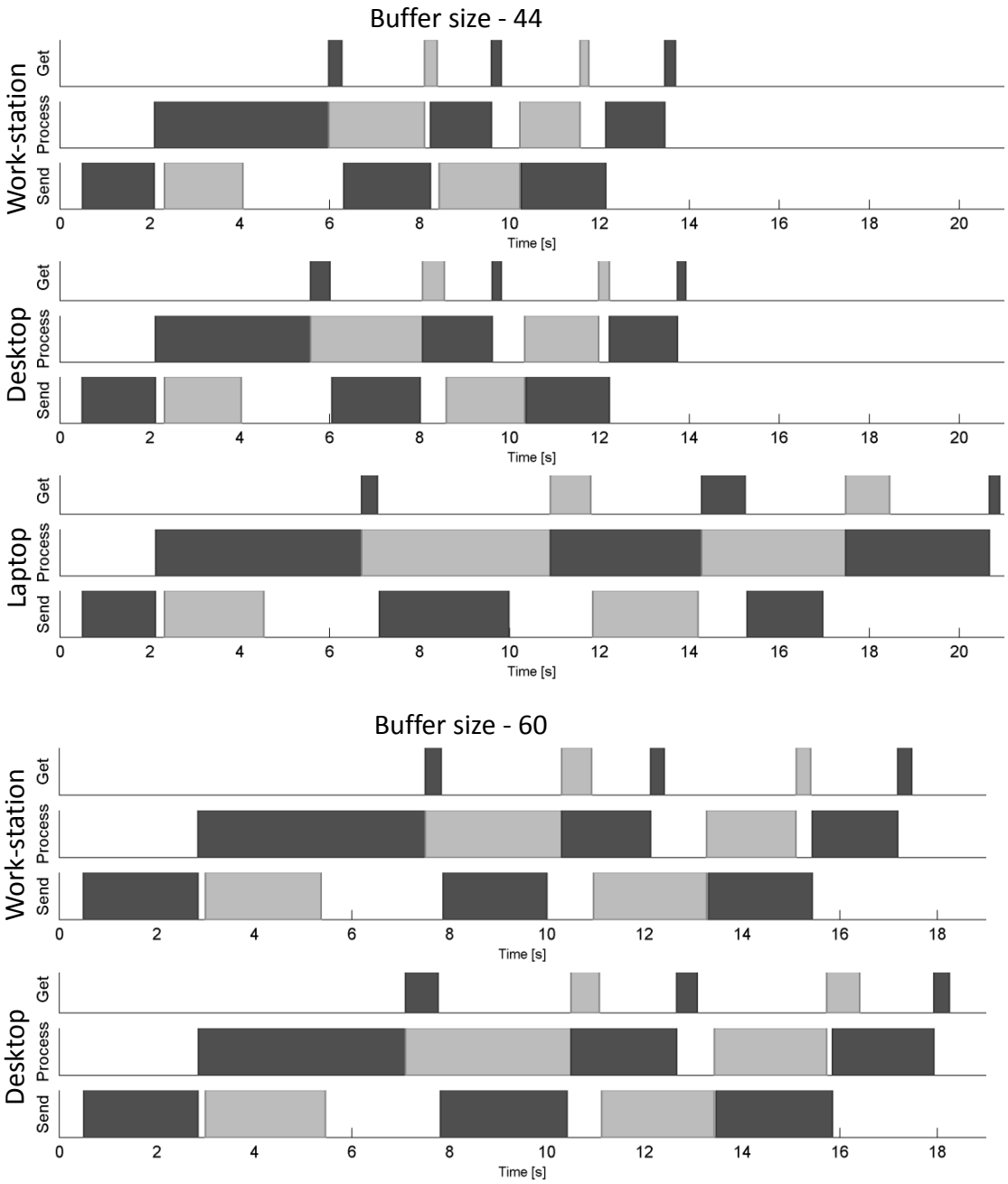

Fig. 7-3 Examples of system workload charts.

Timing results were brought together in Gantt chart like representations of work flow in the distributed system. The alternating colours identify the rotating processing with two buffers. The upper charts compare performance between all the three external reconstructors; the Tesla, Desktop and Laptop, for the buffer size of 44. The lower charts present only comparison between the Laptop and Desktop for the buffer size of 60, as the Laptop had not enough resources for this reconstruction.

Receiver Coils		12		*Flow encodings*	2		
Acceleration		4					
Matrix		128x128					
Frames			*28*	*44*	*60*	*76*	*92*
Work-station							
Send		*[s] (MB/s)*	1.03±0.05(35)	1.79±0.13(32)	2.26±0.12(34)	3.01±0.11(33)	3.71±0.09(32)
Get		*[s] (MB/s)*	0.17±0.06(44)	0.25±0.04(47)	0.36±0.14(44)	0.58±0.39(34)	0.57±0.18(42)
Process	*1st*	*[s]*	3.13	3.87	4.66	5.24	5.98
	2nd	*[s]*	1.35	2.13	2.79	3.48	4.35
	Rest	*[s]*	0.90±0.06	1.33±0.02	1.81±0.05	2.19±0.05	2.68±0.05
Overhead		***[%]***	**40**	**29**	**32**	**27**	**28**
Desktop							
Send		*[s] (MB/s)*	1.09±0.11(33)	1.78±0.13(32)	2.42±0.12(32)	-	-
Get		*[s] (MB/s)*	0.19±0.05(40)	0.31±0.14(37)	0.54±0.16(29)	-	-
Process	*1st*	*[s]*	3.08	3.44	4.25	-	-
	2nd	*[s]*	1.26	2.49	3.39	-	-
	Rest	*[s]*	1.05±0.10	1.57±0.07	2.18±0.11	-	-
Overhead		***[%]***	**30**	**34**	**28**	**-**	**-**
Laptop							
Send		*[s] (MB/s)*	1.17±0.22(31)	2.15±0.51(27)	-	-	-
Get		*[s] (MB/s)*	0.36±0.20(21)	0.69±0.36(17)	-	-	-
Process	*1st*	*[s]*	5.06	4.58	-	-	-
	2nd	*[s]*	2.19	4.20	-	-	-
	Rest	*[s]*	2.14±0.15	3.25±0.09	-	-	-
Overhead		***[%]***	**11**	**16**	**-**	**-**	**-**

Tab. 7-4 Work-load timing results.

The continuous acquisition was split into five reconstruction packets and run for different sizes of the packet (28, 44, 60, 76 and 92). Lack of sufficient memory prevented the buffered reconstruction to be run for a packet size larger than 60 for the Desktop and 44 for the Laptop. The measurements were done on the client side (scanner). The averaged transmission time of the whole packet and transmission time of image results (with the achieved transmission speed) are presented. The processing time was divided into time needed to run the first initial reconstruction on each of two reconstruction buffers and the average of the subsequent reconstruction times. The overhead was calculated as a ratio of the averaged measured time to the averaged SENSE reconstruction time (Tab. 7-3) minus one.

8. Discussion

In this work I describe some of the challenges underlying translation of advanced MRI protocols into the clinical environment. These were addressed with the development and implementation of an external, heterogeneous image reconstructor integrated into the scanner system. In this implementation, distributed client-server architecture was applied to create the flexible, modular platform that can span multiple different MRI systems and reconstruction hardware.

Throughout this work the iterative SENSE reconstruction implemented for GPU was optimised. The generalized GPU implementation reduced the main bottlenecks, the element-wise matrix operations and the gridding steps. However, it was only by speeding up each part of the iterative SENSE reconstruction using the GPU that make it possible to perform reconstruction quicker than acquisition.

Nevertheless, the GPU implementation must be integrated into the scanner's reconstruction pipeline to make the external reconstruction invisible to the end user, which is essential for clinical translation. This introduced a middle step in the form of data transmission that could become the new bottleneck of the reconstruction. Therefore, I implemented a data management scheme that allowed overlap between all three parts of the reconstruction; acquisition, transmission and execution. Also, the reconstruction was optimised to accommodate arbitrary data acquisition patterns, which ensured its suitability for a wide range of applications.

Next, I showed that this reconstruction methodology can be used to translate advanced MR sequences into clinical environment. The tests and validations concentrated on challenging examples of real-time PCMR data acquisitions. One of which was the continuous acquisition with the non-repeating trajectories (the golden angle acquisition). In which a separate gridding matrix had to be created for each trajectory. This prevented optimisation of GPU execution with the batched gridding on GPU. Importantly, it was a valuable example presenting the importance of proper organisation of execution of CPU and GPU tasks. The key step was to keep both CPU and GPU utilised to maximise the time gains from parallel execution.

The presented work not only validated the developed online image reconstruction process, but showed that it is possible to run continuous, real-time acquisitions in an unrestricted by reconstruction time fashion.

Of course, the system has its limitations. The tests identified the weak spots that need to be considered in future developments. The reconstruction algorithm was shown to be very memory intensive. This was limiting for workstations with low available memory. This would be particularly important in bigger systems encompassing multiple clients. Also, the network transmission capacity would need to be carefully considered; including potential data compression as an additional speed up. However, the presented data compression tests were unsatisfactory. Presumably, the MR data exhibits too high entropy and more advanced, algorithms are needed which may be dedicated for real signals.

The devised networking framework allowed flexible organisation of the system, which can be extended with new processing nodes (or replaced) without impacting on applications that run within it. A typical MR system has only a single scanner (client), as scanner systems are not designed to be interconnected by network. To reduce development time the client and server applications implemented only the basic functionalities necessary for them to run in the system. Nevertheless, they could be modified and expanded to allow multiple simultaneous reconstructions for different requesting clients (as discussed in Section 3.4.3)

Also, in systems encompassing multiple active nodes, the adopted distinction into clients and servers could prove limiting. Specifically when considering the active supervision and management of tasks within the distributed system. Presently, task assignment is fixed to a single coupling between server and client. For dynamic load-balancing, a more flexible approach, similar to peer-to-peer (P2P) networks, may be needed. For example, a class of network objects could be capable of identifying themselves and dynamically sharing the processing load within the network.

Similar works toward offloading image reconstruction from the scanner have previously been described (83). These works concentrated on interventional MRI where resultant images were presented on a separate

viewing station. Also, worth of mentioning is a recent work toward an open source platform for implementing and sharing online medical image reconstruction algorithms (84).

In conclusion, all of the project goals were achieved by integrating a GPU-based image reconstructor into the scanner system. I developed and described the novel distributed image reconstruction system dedicated for the clinical MRI. The system allowed integration of existing MR components for the seamless reconstruction process necessary to make a difference in a busy clinical service. Also, it provided a scalable platform for the translation of advanced MRI algorithms. The presented work laid out bases for further developments and improvements, which in consequence has the potential to revolutionize the type of sequences that can be performed on patients.

9. Future work

My work succeeded in creating a flexible system that can be used to translate multiple MRI sequences into the clinical environment. In this final chapter I discuss MRI techniques that I would like to investigate in the near future; i) implementing retrospectively gated sequences with fast GPU reconstructions, ii) image and non-image based navigators for the retrospective gating, iii) researching modified spiral trajectories for non-image based respiration navigators, and iv) expanding and testing the developed system to multiple acquisition and reconstruction nodes.

9.1 Retrospectively gated reconstruction

Chapter 6 presented the modularity of the reconstruction system. However, the system design provides the same flexibility on the client side. The client (or scanner side) is not fixed to purely real-time protocols and the developed *gateway* between computers can be used to provide the fast reconstruction for other types of acquisitions. In keeping with the projects described in this thesis, the real-life example of retrospective cardiac gating will be considered. Cardiac gated PCMR sequences are one of the most often used in clinical cardiac MR. They provide a very high quality and reliable imaging technique. However, they are very time consuming. Alternatively, the spiral sequence could be used (48, 56) providing rapid data acquisition that can be performed within a very short breath-hold.

To date, I have modified the spiral PCMR sequence used in chapter 5 and 6 to support cardiac gated acquisitions. The real-time acquisition pattern used in chapter 6 (Fig. 6-1) was adapted for segmented, retrospectively gated acquisitions. The sequence was prepared to acquire a set of alternating interleaves per heartbeat (read out spiral interleaves per heartbeat). The set of interleaves was changed with each new heartbeat, covering the whole *k*-space over time. The acquisition finished when the last spiral interleave (segment) was acquired. To speed-up the acquisition process, parallel imaging (SENSE) was introduced, which reduces the number of acquired interleaves. Also, temporal encoding (UNFOLD) was implemented to allow two times under-sampling through time.

This acquisition pattern is exactly the same as the one presented on Fig. 6-1, with the exception of the number of read-out lines within each acquisition block. In the real-time approach the number of lines was predefined and equal in each of acquisition blocks. In retrospective acquisitions the number of lines that *fits* into a heartbeat is unknown and varies between heartbeats. A physiological signal (i.e. electrocardiogram - ECG) was used as a trigger to change an acquisition block (sub-set of interleaves).

The self-referencing approach to creation of coil sensitivity maps is impractical for cardiac gated spiral acquisitions, thus a fully sampled set of *k*-space data was acquired at the end of each acquisition.

On the reconstruction side, the whole networking framework and remote reconstruction were left unchanged, as in the real-time assessments. However, an additional functionality was built into the scanner reconstruction pipeline, prior to the data transmission step (Fig. 9-1). Namely, resampling along the time domain of the acquired data was introduced. In retrospective fashion, using the ECG signal for synchronization, the data were divided into separate sections – cardiac intervals. Next, each interval was divided into equal number of cardiac phases. The data from all intervals were realigned with each other by resampling to the beginning of each phase. The Lanczos resampling algorithm was implemented for this purpose. If the acquired data was temporally encoded, the resampling step was preceded with the UNFOLD filtering of

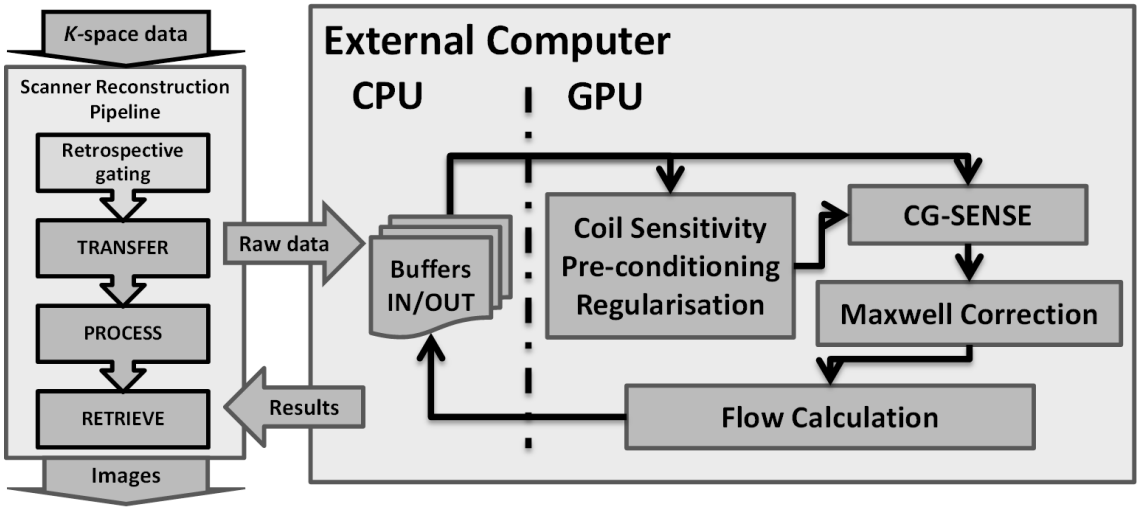

Fig. 9-1 Modified continuous data processing for accelerated gated PCMR data.

The processing flow chart was extended with the resampling step prior to the collection, transmission and remote execution blocks.

the *k*-space data (the same technique as in Chapter 6).

The resampled *k*-space data were then treated as if they originated from a single packet in the continuous real-time acquisition (Chapter 5). The external reconstruction block size was set to the number of cardiac phases which the data was resampled into. Also, the fully sampled data for the coil sensitivity calculations were sent to the external computer; where they were used in the iterative SENSE reconstruction.

Fig. 9-2 presents the first results of retrospectively cardiac gated aortic flow data, which were reconstructed on-line using the distributed reconstruction system (Chapter 3). The imaging parameters were: FOV: 400x400 mm, matrix: 256x256, voxel size: 1.6x1.6x6 mm, TR/TE: 5.1/1.93 ms, flip angle: 20° and VENC: 150 cm/s, complete *k*-space sampling: 80 interleaves. 5x spatial acceleration and 2x temporal encoding was used to minimise the acquisition time. The acquisition required 10 heartbeats, which resulted in data being acquired in ~8 s, as the measured average heart rate was ~86 beats/min. This

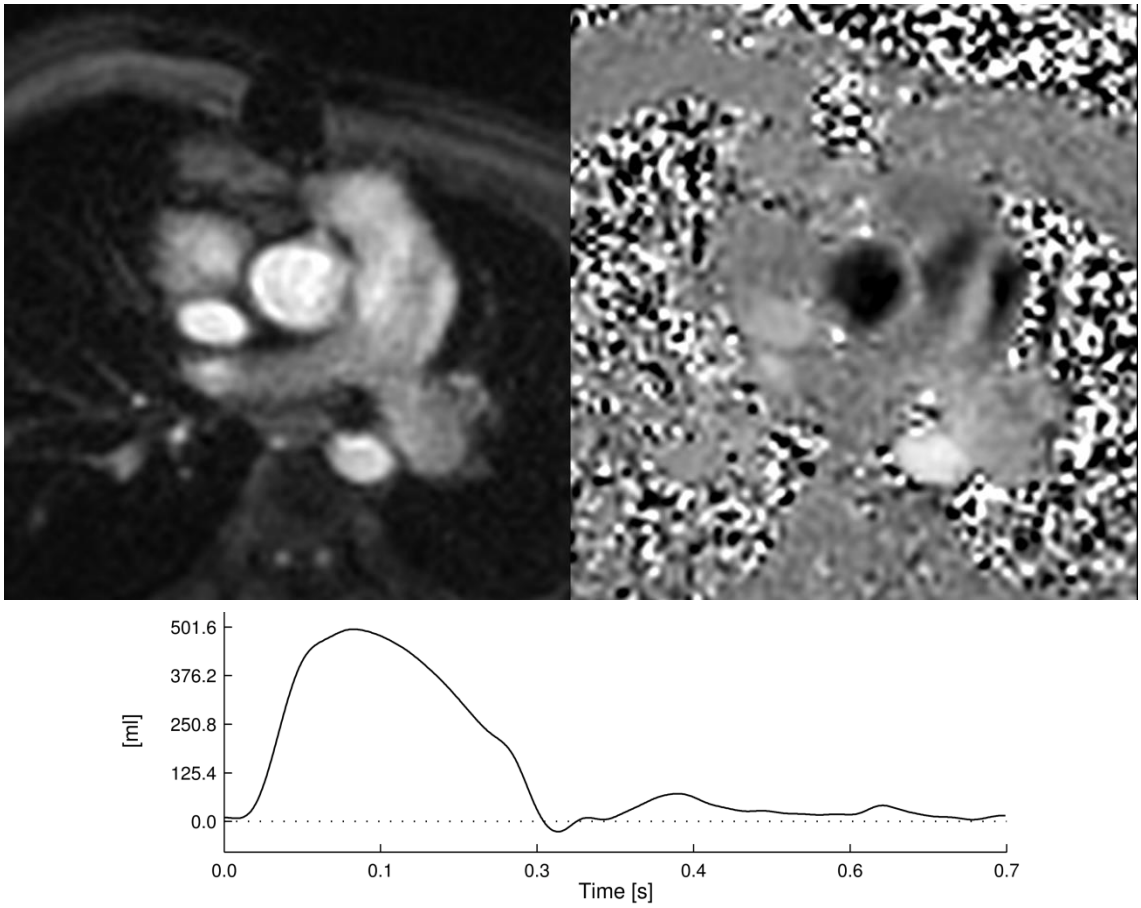

Fig. 9-2 Initial results for accelerated gated PCMR sequence with on-line GPU reconstruction.

allowed the acquisition to be done within a short breath hold.

With minimal work the fast on-line GPU reconstruction of accelerated gated PCMR data was enabled for clinical and research studies. However, the new technique still needs to be thoroughly validated, which is planned for the near future.

9.2 Fast reconstruction of image based self-navigator

The developed fast on-line real-time reconstruction can be used to aid the reconstruction process of self-navigated golden-angle spiral PCMR sequence for free-breathing acquisitions (56). The reconstruction calculates the image-based respiration signal. This is subsequently combined with the simultaneously acquired ECG signal for retrospective segregation of *k*-space read-outs into the cardiac phases. The respiration signal is prepared on the base of lower temporal resolution real-time images. This is enabled by rotating read-out interleaves by the golden-angle, with each new frame. Consequently, the adjacent read-outs can be combined into the real-time series of desired temporal resolution. The real-time data need to be reconstructed with the SENSE reconstruction before the separation into cardiac phases can start. This reconstruction process is the limiting aspect of the algorithm. Usually, the data are acquired over a period of ~5 minutes resulting in a long reconstruction process; up to 40 minutes (using the multi-threaded CPU version of the SENSE algorithm run on the native image reconstructor).

In future work, I would like to use the techniques developed in my work to speed-up the real-time data reconstruction for the image-based navigator calculations. The real-time data acquired with the retrospectively gated spiral PCMR sequence are equivalent to the data acquired in the presented continuous real-time assessment protocols (Chapter 7). Consequently, the same external GPU reconstruction can be used to improve the reconstruction speed.

9.3 Modified spiral acquisition for self-navigating

The calculation of the respiration navigator enables the free-breathing gated acquisitions. The Cartesian and radial trajectories benefit from possibility of self-navigating based on a single *k*-space read-out line. The idea is to acquire a 1D signal in the anterior-posterior (front to back) direction allowing monitoring of the chest wall motion. This strategy is in contradiction with the spiral trajectory, in which data are acquired on spiral interleaves. Alternatively, the creation of image based self-navigator signal can be done; however this is a very time consuming process (56).

In future work, I would like to explore a new concept of self-navigating for the spiral acquisitions using the gradient *re-winding* data. At the end of each *k*-space acquisition, the encoding spatial information linear gradients have to be *ramped-down* to zero or *re-winded* back to the centre of *k*-space. In our sequences, this is a *dead* time when no sampling is performed, at present. This linear traversing of *k*-space may be used to acquire additional 1D data. Providing no information regarding the phase of data were necessary, reading out only a half of the line should suffice to create a reliable navigator. A modification to the spiral trajectory can be applied allowing linear read-outs of the same *k*-space portion. Namely, after reading-out each spiral interleave, the read-out position could be moved to the same location of the outer-portion of *k*-space (Fig. 9-3). This would be the starting point of the navigator read-outs, which would end in the centre of the *k*-space; at the same time *re-winding* the gradients. Of course, the trajectory errors due to long read-outs can affect the ending point of the spiral interleaves. However, I suspect that the exact alignment of the consecutive *navigators* may not be so important to some extent, as long as the directionality of each read-out is preserved. However, this will need a detailed evaluation and study.

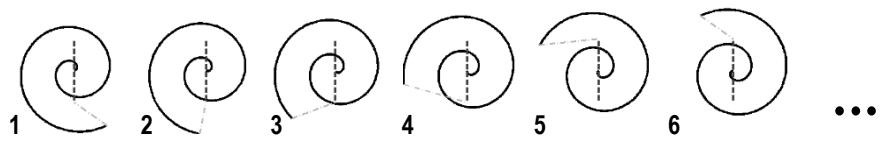

Fig. 9-3 An example of modified spiral trajectory including additional navigator read-outs.

The figure presents a series of rotating spiral read-outs (the black thick lines) followed with a repositioning of a read-out (the dashed grey line) and a navigator data read-out (the dashed black line).

9.4 MRI as a web service

The work described in this thesis provides flexibility in introducing new components to existing MR systems. The fast external reconstruction with hardware accelerators was efficiently integrated into the MR system. This was done without compromise in the existing clinical/research framework. The next step will be to expand the system onto more MR scanners, which will benefit from the fast reconstruction process when connected into the system. Ultimately, the idea would be expanded into a grid (cloud) connecting not only on-site resources, but also on-web available processing nodes. This will need further development and research in the scope of applied middleware providing identification of resources, load balancing and security to the system.

Presently, the implemented distributed reconstruction system was a minimal version of the presented concept (Chapter 3). The implementation was prepared to meet the needs imposed by the on-site hardware distribution. Only a single external computer was connected to a scanner, as connecting multiple scanners to the same external computer was *logistically* difficult. Consequently, to not overcomplicate the implementation, the server applications were prepared to maintain a single reconstruction process, as only a single scanning protocol can run on a scanner at a time. Nevertheless, I consider the many-to-many assignment as a very important to develop in future work. This will allow more optimal use of the available resources. To clarify, the present implementation was not *hardwired* to the specific networked hardware, but only restricted in the implemented functionality. Namely, the identification provided from the client was reduced to indexing of external buffers for which the processing request was issued. Similarly, the server side was prepared to maintain reconstruction objects (buffers) only for a single client. When expanding the system, simple measures can be undertaken to address these issues in new versions of servers; i.e. assigning a unique identification to each client and adding a client identification step on the server side. However, this would only be the first step to increase the *awareness* of the components. In future work, I would like to research a more robust and intelligent management of resources, allowing the real-time load balancing. This is crucial for big systems containing multiple processing nodes. The work would involve replacing or enriching the naming service used to identify the distributed

components. The new version would be an active part of the system monitoring the workload, rather than a static record of system's components.

In this new setting, the adopted client-server architecture with the division into two kinds of network objects; client and server, can be seen as a limitation. Self-discovering and collaborating objects could be a better approach. This could be introduced with peer-to-peer (p2p) network architecture. In p2p networks each object is equally privileged being at the same time a supplier and consumer of resources.

10. References

1. Purcell EM, Torrey HC, Pound RV. Resonance Absorption by Nuclear Magnetic Moments in a Solid. Physical Review. 1946;69(1-2):37-8.

2. Bloch F, Hansen WW, Packard M. Nuclear Induction. Physical Review. 1946;69(3-4):127-.

3. Rabi II, Zacharias JR, Millman S, Kusch P. A New Method of Measuring Nuclear Magnetic Moment. Physical Review. 1938;53(4):318-.

4. Zeeman P. XXXII. On the influence of magnetism on the nature of the light emitted by a substance. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 1897;43(262):226-39.

5. Haacke EM, Brown R, Thompson M, Venkatesan R. Magnetic resonance imaging: physical principles and sequence design. New York: John Wiley and Sons, Inc; 1999.

6. Cooley JW, Tukey JW. An algorithm for the machine calculation of complex Fourier series. Mathematics of computation. 1965;19(90):297-301.

7. McGibney G, Smith MR, Nichols ST, Crawley A. Quantitative evaluation of several partial Fourier reconstruction algorithms used in MRI. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 1993;30(1):51-9. Epub 1993/07/01.

8. Feinberg DA, Hale JD, Watts JC, Kaufman L, Mark A. Halving MR imaging time by conjugation: demonstration at 3.5 kG. Radiology. 1986;161(2):527-31. Epub 1986/11/01.

9. Margosian P, Schmitt F, Purdy D. Faster MR imaging: imaging with half the data. Health Care Instrum. 1986;1(6):195.

10. Noll DC, Nishimura DG, Macovski A. Homodyne detection in magnetic resonance imaging. IEEE transactions on medical imaging. 1991;10(2):154-63. Epub 1991/01/01.

11. Cuppen J, van Est A. Reducing MR imaging time by one-sided reconstruction. Magnetic Resonance Imaging. 1987;5(6):526-7.

12. Haacke E, Lindskogj E, Lin W. A fast, iterative, partial-Fourier technique capable of local phase recovery. Journal of Magnetic Resonance (1969). 1991;92(1):126-45.

13. Mansfield P. Multi-planar image formation using NMR spin echoes. Journal of Physics C: Solid State Physics. 1977;10(3):L55.

14. Ahn CB, Kim JH, Cho ZH. High-speed spiral-scan echo planar NMR imaging-I. IEEE transactions on medical imaging. 1986;5(1):2-7. Epub 1986/01/01.

15. Yudilevich E, Stark H. Interpolation from samples on a linear spiral scan. IEEE transactions on medical imaging. 1987;6(3):193-200. Epub 1987/01/01.

16. Rasche V, Holz D, Schepper W. Radial turbo spin echo imaging. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 1994;32(5):629-38. Epub 1994/11/01.

17. Steeden JA, Atkinson D, Taylor AM, Muthurangu V. Assessing vascular response to exercise using a combination of real-time spiral phase contrast MR and noninvasive blood pressure measurements. Journal of magnetic resonance imaging : JMRI. 2010;31(4):997-1003.

18. Brouw W. Aperture synthesis. Image Processing Techniques in Astronomy: Springer; 1975. p. 301-7.

19. O'Sullivan JD. A fast sinc function gridding algorithm for fourier inversion in computer tomography. IEEE transactions on medical imaging. 1985;4(4):200-7. Epub 1985/01/01.

20. Schomberg H, Timmer J. The gridding method for image reconstruction by Fourier transformation. IEEE transactions on medical imaging. 1995;14(3):596-607. Epub 1995/01/01.

21. Jackson JI, Meyer CH, Nishimura DG, Macovski A. Selection of a convolution function for Fourier inversion using gridding [computerised tomography application]. IEEE transactions on medical imaging. 1991;10(3):473-8. Epub 1991/01/01.

22. Carlson J, Minemura T. Imaging time reduction through multiple receiver coil data acquisition and image reconstruction. Magnetic Resonance in Medicine. 1993;29(5):681-7.

23. Sodickson DK, Manning WJ. Simultaneous acquisition of spatial harmonics (SMASH): fast imaging with radiofrequency coil arrays. Magnetic Resonance in Medicine. 1997;38(4):591-603.

24. Pruessmann KP, Weiger M, Scheidegger MB, Boesiger P. SENSE: sensitivity encoding for fast MRI. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 1999;42(5):952-62. Epub 1999/11/05.

25. Griswold MA, Jakob PM, Heidemann RM, Nittka M, Jellus V, Wang J, et al. Generalized autocalibrating partially parallel acquisitions (GRAPPA). Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2002;47(6):1202-10. Epub 2002/07/12.

26. Madore B, Glover GH, Pelc NJ. Unaliasing by fourier-encoding the overlaps using the temporal dimension (UNFOLD), applied to cardiac imaging and fMRI. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 1999;42(5):813-28. Epub 1999/11/05.

27. Kellman P, Epstein FH, McVeigh ER. Adaptive sensitivity encoding incorporating temporal filtering (TSENSE). Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2001;45(5):846-52. Epub 2001/04/27.

28. Tsao J, Boesiger P, Pruessmann KP. k-t BLAST and k-t SENSE: dynamic MRI with high frame rate exploiting spatiotemporal correlations.

Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2003;50(5):1031-42. Epub 2003/10/31.

29. Madore B. UNFOLD-SENSE: a parallel MRI method with self-calibration and artifact suppression. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2004;52(2):310-20. Epub 2004/07/30.

30. Pruessmann KP, Weiger M, Bornert P, Boesiger P. Advances in sensitivity encoding with arbitrary k-space trajectories. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2001;46(4):638-51. Epub 2001/10/09.

31. Schaeffter T, Hansen MS, Sørensen TS, editors. Fast Implementation of Iterative Image Reconstruction. International Society for Magnetic Resonance in Medicine.

32. Tsao J. On the UNFOLD method. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2002;47(1):202-7. Epub 2002/01/05.

33. Madore B. Using UNFOLD to remove artifacts in parallel imaging and in partial-Fourier imaging. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2002;48(3):493-501. Epub 2002/09/05.

34. Roemer P, Edelstein W, Hayes C, Souza S, Mueller O. The NMR phased array. Magnetic Resonance in Medicine. 1990;16(2):192-225.

35. Hansen PC. Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion: SIAM; 1998.

36. Ying L, Xu D, Liang ZP. On Tikhonov regularization for image reconstruction in parallel MRI. Conference proceedings : Annual International Conference of the IEEE Engineering in Medicine and Biology Society IEEE

Engineering in Medicine and Biology Society Conference. 2004;2:1056-9. Epub 2007/02/03.

37. Lin F-H, Kwong KK, Belliveau JW, Wald LL. Parallel imaging reconstruction using automatic regularization. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2004;51(3):559-67.

38. Kellman P, Sorger JM, Epstein FH, McVeigh ER. Low-latency temporal filter design for real-time MRI using UNFOLD. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2000;44(6):933-9. Epub 2000/12/07.

39. Di Bella EV, Wu YJ, Alexander AL, Parker DL, Green D, McGann CJ. Comparison of temporal filtering methods for dynamic contrast MRI myocardial perfusion studies. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2003;49(5):895-902. Epub 2003/04/22.

40. Guttman MA, Kellman P, Dick AJ, Lederman RJ, McVeigh ER. Real-time accelerated interactive MRI with adaptive TSENSE and UNFOLD. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2003;50(2):315-21. Epub 2003/07/24.

41. Harris M. General-Purpose computation on Graphics Processing Units. 2002; Available from: gpgpu.org.

42. CUDA C Programming Guide. NVIDIA Corporation; 2014 [updated February 13, 2014; cited 2014]; Available from: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

43. CUDA C Best Practices Guide. NVIDIA Corporation; 2014 [updated February 13, 2014; cited 2014]; Available from: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html.

44. Corporation I. Intel® Xeon Phi™ Coprocessor 7120D (16GB, 1.238 GHz, 61 core). 2014 [cited 2014]; Available from:

http://ark.intel.com/products/80310/Intel-Xeon-Phi-Coprocessor-7120D-16GB-1_238-GHz-61-core.

45. Corporation I. Intel® Xeon Phi™ Product Family Performance. [cited 2014]; Available from: http://www.intel.co.uk/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf.

46. Moran PR. A flow velocity zeugmatographic interlace for NMR imaging in humans. Magn Reson Imaging. 1982;1(4):197-203. Epub 1982/01/01.

47. Steeden JA, Atkinson D, Taylor AM, Muthurangu V. Split-acquisition real-time CINE phase-contrast MR flow measurements. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2010;64(6):1664-70. Epub 2010/10/13.

48. Steeden JA, Atkinson D, Hansen MS, Taylor AM, Muthurangu V. Rapid flow assessment of congenital heart disease with high-spatiotemporal-resolution gated spiral phase-contrast MR imaging. Radiology. 2011;260(1):79-87. Epub 2011/03/19.

49. Hansen MS, Atkinson D, Sorensen TS. Cartesian SENSE and k-t SENSE reconstruction using commodity graphics hardware. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2008;59(3):463-8. Epub 2008/02/29.

50. Sorensen TS, Schaeffter T, Noe KO, Hansen MS. Accelerating the nonequispaced fast Fourier transform on commodity graphics hardware. IEEE transactions on medical imaging. 2008;27(4):538-47. Epub 2008/04/09.

51. Stone SS, Haldar JP, Tsao SC, Hwu WM, Sutton BP, Liang ZP. Accelerating Advanced MRI Reconstructions on GPUs. J Parallel Distrib Comput. 2008;68(10):1307-18. Epub 2008/10/01.

52. Group OM. OMG IDL Syntax and Semantics. [cited 2014]; Available from: http://www.omg.org/cgi-bin/doc?formal/02-06-39.

53. Ltd A. omniORB : Free CORBA ORB. [cited 2014]; Available from: http://omniorb.sourceforge.net/.

54. Mortensen KH, Steeden JA, Panzer J, Taylor AM, Muthurangu V. Isometric exercise in cardiac magnetic resonance imaging: an initial experience using fast imaging. Journal of Cardiovascular Magnetic Resonance. 2011;13(Suppl 1):P386.

55. Jones A, Steeden JA, Pruessner JC, Deanfield JE, Taylor AM, Muthurangu V. Detailed assessment of the hemodynamic response to psychosocial stress using real-time MRI. Journal of magnetic resonance imaging : JMRI. 2011;33(2):448-54. Epub 2011/01/29.

56. Steeden JA, Knight DS, Bali S, Atkinson D, Taylor AM, Muthurangu V. Self-navigated tissue phase mapping using a golden-angle spiral acquisition-proof of concept in patients with pulmonary hypertension. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2014;71(1):145-55. Epub 2013/02/16.

57. Steeden JA, Jones A, Pandya B, Atkinson D, Taylor AM, Muthurangu V. High-resolution slice-selective Fourier velocity encoding in congenital heart disease using spiral SENSE with velocity unwrap. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2012;67(6):1538-46. Epub 2012/04/19.

58. Alted F. Why Modern Cpus Are Starving and What Can Be Done About It. Comput Sci Eng. 2010;12(2):68-71.

59. Gregerson A. Implementing fast MRI gridding on GPUs via CUDA. NVidia Whitepaper, Online: http://cnnvidiacom/docs/IO/47905/ECE757_Project_Report_Gregersonpdf. 2008.

60. Podlozhnyuk V. Image convolution with CUDA. NVIDIA Corporation white paper, June. 2007;2097(3).

61. Luo Y, Duraiswami R, editors. Canny edge detection on NVIDIA CUDA. Computer Vision and Pattern Recognition Workshops, 2008 CVPRW'08 IEEE Computer Society Conference on; 2008: IEEE.

62. Zhang N, Chen Y-s, Wang JL, editors. Image parallel processing based on GPU. Advanced Computer Control (ICACC), 2010 2nd International Conference on; 2010: IEEE.

63. Bernstein MA, Zhou XJ, Polzin JA, King KF, Ganin A, Pelc NJ, et al. Concomitant gradient terms in phase contrast MR: Analysis and correction. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 1998;39(2):300-8.

64. White SW, Quail AW, de Leeuw PW, Traugott FM, Brown WJ, Porges WL, et al. Impedance cardiography for cardiac output measurement: an evaluation of accuracy and limitations. European heart journal. 1990;11 Suppl I:79-92. Epub 1990/12/01.

65. Hecht HS, DeBord L, Sotomayor N, Shaw R, Dunlap R, Ryan C. Supine bicycle stress echocardiography: peak exercise imaging is superior to postexercise imaging. Journal of the American Society of Echocardiography : official publication of the American Society of Echocardiography. 1993;6(3 Pt 1):265-71. Epub 1993/05/01.

66. Klein C, Schalla S, Schnackenburg B, Bornstedt A, Fleck E, Nagel E. Magnetic resonance flow measurements in real time: Comparison with a standard gradient-echo technique. Journal of magnetic resonance imaging : JMRI. 2001;14(3):306-10.

67. Rosset A, Spadola L, Ratib O. OsiriX: an open-source software for navigating in multidimensional DICOM images. Journal of digital imaging : the official journal of the Society for Computer Applications in Radiology. 2004;17(3):205-16. Epub 2004/11/10.

68. Odille F, Steeden JA, Muthurangu V, Atkinson D. Automatic segmentation propagation of the aorta in real-time phase contrast MRI using

nonrigid registration. Journal of magnetic resonance imaging : JMRI. 2011;33(1):232-8.

69. Frick MH, Somer T. Base-Line Effects on Response of Stroke Volume to Leg Exercise in the Supine Position. Journal of applied physiology. 1964;19:639-43. Epub 1964/07/01.

70. Pandya B, Kowalik GT, Knight DS, Tann O, Derrick G, Muthurangu V. Towards a more comprehensive assessment of cardiovascular fitness-magnetic resonance augmented cardiopulmonary exercise testing (MR-CPEX). Journal of Cardiovascular Magnetic Resonance. 2013;15(Suppl 1):P58.

71. Guier WH, Friesinger GC, Ross RS. Beat-by-beat stroke volume from aortic-pulse-pressure analysis. IEEE transactions on bio-medical engineering. 1974;21(4):285-92. Epub 1974/07/01.

72. Oh JK, Tajik J. The return of cardiac time intervals: the phoenix is rising. Journal of the American College of Cardiology. 2003;42(8):1471-4. Epub 2003/10/18.

73. Hansen MS, Baltes C, Tsao J, Kozerke S, Pruessmann KP, Eggers H. k-t BLAST reconstruction from non-Cartesian k-t space sampling. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2006;55(1):85-91. Epub 2005/12/03.

74. Afacan O, Hoge WS, Janoos F, Brooks DH, Morocz IA. Rapid full-brain fMRI with an accelerated multi shot 3D EPI sequence using both UNFOLD and GRAPPA. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2012;67(5):1266-74. Epub 2011/11/19.

75. Vivekananthan K, Kalapura T, Mehra M, Lavie C, Milani R, Scott R, et al. Usefulness of the combined index of systolic and diastolic myocardial performance to identify cardiac allograft rejection. The American journal of cardiology. 2002;90(5):517-20. Epub 2002/09/05.

76. Winkler P. Effects of experimental iatrogenic hypercortisolism on systemic and pulmonary artery pressure, left ventricular mass as well as left and right ventricular dimension and function in dogs - an echocardiographic study. University of Zurich; 2009.

77. Nielsen JF, Nayak KS. Referenceless phase velocity mapping using balanced SSFP. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2009;61(5):1096-102. Epub 2009/02/21.

78. Joseph AA, Merboldt KD, Voit D, Zhang S, Uecker M, Lotz J, et al. Real-time phase-contrast MRI of cardiovascular blood flow using undersampled radial fast low-angle shot and nonlinear inverse reconstruction. NMR in biomedicine. 2012;25(7):917-24. Epub 2011/12/20.

79. Kowalik GT, Steeden JA, Pandya B, Odille F, Atkinson D, Taylor A, et al. Real-time flow with fast GPU reconstruction for continuous assessment of cardiac output. Journal of magnetic resonance imaging : JMRI. 2012;36(6):1477-82. Epub 2012/06/30.

80. Shin T, Nielsen JF, Nayak KS. Accelerating dynamic spiral MRI by algebraic reconstruction from undersampled k-t space. IEEE transactions on medical imaging. 2007;26(7):917-24. Epub 2007/07/26.

81. Rojo EC, Rodrigo JL, Perez de Isla L, Almeria C, Gonzalo N, Aubele A, et al. Disagreement between tissue Doppler imaging and conventional pulsed wave Doppler in the measurement of myocardial performance index. European journal of echocardiography : the journal of the Working Group on Echocardiography of the European Society of Cardiology. 2006;7(5):356-64. Epub 2005/10/04.

82. Frank MN, Haberern N. The effect of hand grip and exercise on systolic time intervals in human subjects. The American journal of the medical sciences. 1971;261(4):219-23. Epub 1971/04/01.

83. Roujol S, Senneville BDd, Vahala E, Sorensen TS, Moonen C, Ries M. Online real-time reconstruction of adaptive TSENSE with commodity CPU/GPU

hardware. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2009;62(6):1658-64.

84. Hansen MS, Sorensen TS. Gadgetron: an open source framework for medical image reconstruction. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2013;69(6):1768-76. Epub 2012/07/14.

11. Appendices

11.1 Network communication module

```
#ifndef TRANSFER_MODULE_IDL
#define TRANSFER_MODULE_IDL

module TransferModule {

    typedef long TDLong;
    typedef unsigned long TDULong;
    typedef char TDBase;

    typedef sequence<TDBase> TransferData;

    interface DataTransfer {

        TDLong Init(in TDBase id, in string oper, in TransferData inData);

        TDLong SetRawData(in TDBase id, in string oper, in TransferData inData );

        TDLong Process(in TDBase id, in string oper);

        TDLong GetResultData(in TDBase id, in string oper, in TransferData index, out TransferData
    outData);

    };
};
#endif
```

This code was used to generate C++ version of the network interfaces for the distributed reconstruction system. Each interface provides the client with a set of input parameters. A pair; *in TDBase id, in string oper* of input parameters can be used by a client to identify a remote reconstruction object (data structures, buffers) as well as to specify a variant of operation if necessary. Parameters of *TransferData* type are used to transmit an arbitrary length of data. All of the interfaces return a status of a requested operation.

11.2 Reconstruction module interface

```
#ifndef IRECONSTURCTION_H
#define IRECONSTURCTION_H

#include <stdlib.h>
#include <iostream>

namespace ICH_MRI
{
namespace SERVER
{
class IReconstruction
{
public:
    bool isRecInit;  // It decides if the PreProcess function does an initialization step or not.

public:
    IReconstruction():isRecInit(false)   {   return;     }
    virtual ~IReconstruction() {   }

          // It is used to read initialization data.
    virtual bool ReadHeader(std::istream &in) = 0;

          //  Processing functions
          //     called by servant in this order.
    virtual bool PreProcess(const char* oper) = 0;
    virtual bool Process(const char* oper) = 0;
    virtual bool PostProcess(const char* oper) = 0;
    virtual bool Store(const char* oper) = 0;
    virtual bool Clean(const char* oper) = 0;
```

```
30.
31.           // Is called by servant::* each time a new request come.
32.     virtual bool StartThread(int i) = 0;
33.
34.           // Data transmission functions.
35.     virtual int SetData(unsigned int dataSize, void* inData) = 0;
36.     virtual unsigned int GetResSize(unsigned int size, const void * index, void *&ptr) = 0;
37.     virtual unsigned int GetResultData(void *_ptr, void* outData) = 0;
38.
39.     virtual bool CopyBuffer(const IReconstruction *p, bool isCPUMem = true) = 0;
40.     virtual bool ManageBuffers(IReconstruction **mri, unsigned int buffSize, unsigned int buffId)
41.     {   return true;  }
42.
43.     virtual int Test(int argc, char** argv) = 0;
44. };
45.
46. #ifndef __DLL_EXPORT
47. #define __DLL_EXPORT __declspec(dllimport)
48. typedef IReconstruction* (*_GetIReconstruction)(void);
49. typedef bool (*_StartLibrary)(void);
50. typedef bool (*_StopLibrary)(void);
51. #else
52. #undef __DLL_EXPORT
53. #define __DLL_EXPORT __declspec(dllexport)
54. #endif
55.
56. #if defined(_WINDLL)
57. extern "C" __DLL_EXPORT IReconstruction* __cdecl GetIReconstruction(void);
58. extern "C" __DLL_EXPORT bool __cdecl StartLibrary(void);
59. extern "C" __DLL_EXPORT bool __cdecl StopLibrary(void);
60. #endif
61.
62. }
63. }
64. #endif
```

The listing presents the reconstruction module interface used by the implemented server application. The presented class serves as a *root* for all modules that are accessible through the server. The presented abstract methods must be provided within the deriving reconstruction module/class. Similarly, a new module must implement the global module initialisation (*StartLibrary*, *StartLibrary*) and reconstruction object creation (*GetIReconstruction*) functions.

11.3 The pseudo code of the conjugate gradient linear solver algorithm for the SENSE reconstruction

1. $\vec{r} = IE^H D^2 \vec{S}$
2. $\vec{p} = \vec{r}$
3. $\vec{b} = 0$
4. $\vec{q} = 0$
5. $stop = rhr = \vec{r}^H \vec{r}$
6. $while(true)$
7. $\quad \vec{q} = (IE^H D^2 EI + \lambda^2 I\theta^{-1} I)\vec{p}$
8. $\quad \alpha = \frac{rhr}{\vec{p}^H \vec{q}}$
9. $\quad \vec{b} = \vec{b} + \alpha\vec{p}$
10. $\quad \vec{r} = \vec{r} - \alpha\vec{q}$
11. $\quad rhr1 = \vec{r}^H \vec{r}$
12. $\quad if\ i \geq MAXITER\ or\ \frac{rhr1}{stop} < EPS$
13. $\quad\quad \vec{b} = I\vec{b}$
14. $\quad\quad break$
15. $\quad endif$
16. $\quad \vec{p} = \vec{r} + \frac{rhr1}{rhr}\vec{p}$
17. $\quad rhr = rhr1$
18. $endwhile$

The algorithm presents the conjugate-gradient solver (Section 1.4.2), which is an iterative method for solving sparse systems of linear equations. The algorithm was adapted to the reconstruction needs, based on the Equations 1-41, 1-42 and 1-43.

11.4 The template of element-wise matrix-vector operations on GPU

```
1.  // The function multiplies each row of matrix, 'mat' with elements of array 'vec'.
2.  // It is an element-wise multiplicatiion.
3.  // Threads of each thread block iterate through y-dim., thus there is no need to span them
4.  // on the whole y-dim size.
5.  // (e.g. threadBlockSize.y = 1 & gridSize.y = 1 will iterate through all data rows;
6.  //    it is advised to play with these parameter to achieve better performance).
7.  // x-dim : number/index of an element in a row;
8.  // y-dim : index of a row.
9.  // Shared memory is used to store 'vect' values. These are shared among
10. //    threads with the same y-index.
11. // vSize - number of elements in a vector 'vec'.
12. // rowSize - size of each row in bytes.
13.
14. extern volatile __shared__ char _scal[];
15.
16. template <class T1, class T2>
17. __global__ void mulElemMatVec(T1 *mat, T1 *out,
18.                   FFT_UNSIGN nRows, FFT_UNSIGN rowSize,
19.                   T2 *vec, FFT_UNSIGN vSize)
20. {
21.       // Thread's index.
22.    FFT_UNSIGN id = threadIdx.x;
23.       // The possition within a vector.
24.    FFT_UNSIGN vId = id + blockDim.x * blockIdx.x;
25.       // The index of matrix row to be processed.
26.    FFT_UNSIGN rowId = threadIdx.y + blockDim.y * blockIdx.y;
27.
28.    T2* scal = (T2*)_scal;
29.
30.    if(vId >= vSize) {   return;   }
31.       // Read in the vector values.
32.    if(threadIdx.y == 0)
33.    {
34.       scal[id] = *(vec + vId);
35.    }
36.       // Synchronise the treads within the thread block.
37.    __syncthreads();
38.
39.    T1 tmp;
40.    T1 *_ptr;
41.       // iterate through all of the matrix rows.
42.    for(; rowId < nRows; rowId += blockDim.y * gridDim.y)
43.    {
44.          // Read-in matrix data.
45.       _ptr = FLAT_MEM_PTR(mat, 0, rowId, rowSize, T1);
46.       tmp = _ptr[vId];
47.          // Scale the data.
48.       tmp *= scal[id];
49.          // Store the result.
50.       _ptr = FLAT_MEM_PTR(out, 0, rowId, rowSize, T1);
51.       _ptr[vId] = tmp;
52.    }
53. }
```

The appendix presents a simple GPU *kernel* dedicated to speed-up the *element-wise multiplication* of matrices. The *kernel* was written in C for CUDA and can be complied for NVIDIA GPU cards.

11.5 The gridding optimisation tests and results

This appendix presents the extended evaluation and solution to the problem of gridding of data acquired on the non-repeating trajectories; which was introduced in Chapter 7. The tests concentrated on optimisation of the gridding process, as the new reconstruction required reformulation of this step. The optimisation steps I described aimed to expand the developed reconstruction onto the non-repeating trajectories (generalised reconstruction) without compromise in the provided reconstruction performance. The gridding tests were done with the repeating and non-repeating trajectories for comparison.

This appendix complements the previous description and results by extending the tests on to different GPU enabled hardware. The tests were run on the previously described system (Section 3.4 and 4.6) using the three computers, which specifications are presented in Tab. 7-1, as the external reconstructors.

11.5.1 Gridding tests

The same spiral PCMR sequence, as in the transmission tests (Section 3.5), was used to acquire multiple data sets (as presented in Tab. 11-1 and the following) for the optimisation tests.

The boundary limits; *Acquisition time*, *Upper-* and *Lower-limits* are presented in Tab. 11-1. As previously stated, the *acquisition time* and *Upper-limit* were the same in all tests, as they are related to the sequence and scanner reconstructor performance. The first was defined as the time needed to acquire the whole set of frames and the second was the time needed by the native scanner reconstructor to run the gridding on the set of frames acquired with the non-repeating trajectories.

As discussed in the Section 7.2.1, the *Lower-limit* was calculated, as the time needed by an external computer to run the batched version of gridding for data on the repeating trajectory. The results are presented in Tab. 11-1. The tests showed speed-up compared to the *Acquisition time*, of ~43x, ~58x and ~15x for Work-station, Desktop and Laptop respectively (all further results are presented in this order). Assuming that the gridding accounted to about half of

each iteration time and that at least seven iterations are required, the Laptop would not provide sufficient speed-up for the reconstruction process. Nevertheless, the consecutive optimisation stages were run on the Laptop for the completeness of the comparative study. In the test, the Desktop performed notably better, which can be assigned to higher clock rate and number of CUDA cores (Tab. 7-1).

The test allowed estimation of the maximum achievable speed-up with the GPU gridding. The batched gridding results were linearly dependent on the number of gridded frames. Consequently, the maximum achievable speed-up (Tab. 11-1; Ratio) was fixed for each machine; ~19x, ~25x and ~6x.

Number of frames	28		44		60		76		92		Average
	[ms]	Ratio	[ms]	Ratio	[ms]	Ratio	[ms]	Ratio	[ms]	Ratio	Ratio
Acquisition time	1162.6		1826.9		2491.2		3155.5		3819.8		
Upper-limit	555.1	2.1	744.4	2.5	1040.9	2.4	1300.9	2.4	1616.4	2.4	2.3±0.1
Lower-limit											
Work-station	26.8	20.7	42.0	17.7	57.2	18.2	72.4	18.0	87.7	18.4	**18.6±1.4**
Desktop	20.2	27.5	31.7	23.5	43.2	24.1	54.7	23.8	66.2	24.4	**24.7±1.9**
Laptop	78.2	7.1	122.7	6.1	167.4	6.2	211.5	6.2	256.7	6.3	**6.4±0.5**

Tab. 11-1 Estimation of the gridding optimisation limits.

The *Upper-limit* was determined as the time needed by the native image reconstructor to grid data on the non-repeating trajectories. The *Lower-limit* was calculated as the time needed by an external computer to grid data on repeating trajectory. *Ratios* for the *Upper-limit* were calculated with reference to the *acquisition time* and for the *Lower-limit* with reference to the *Upper-limit*.

The initial assessment (Section 7.2.1) looked into gridding times providing all data structures were available on a GPU. The aim was to determine an impact of sequential scheduling of the gridding tasks as compared with the single *batch* call. The results for this rather impractical approach were collected in Tab. 11-2. The tests showed that providing the whole data structures were already available in the GPU memory, the sequential scheduling of gridding operations yields the same results as the batched gridding (the *Lower-limit*). Nevertheless, the memory consumption (as discussed in Section 7.2.1) was too high and better solutions were needed.

Number of frames		28	44	60	76	92	Average
Total / per frame	Work-station	26.82/0.96	42.08/0.96	57.35/0.96	72.61/0.96	87.83/0.95	- / 0.96±0.00
`	Desktop	20.27/0.72	31.85/0.72	43.45/0.72	54.95/0.72	66.55/0.72	- / 0.72±0.00
	Laptop	78.19/2.79	123.06/2.80	167.67/2.79	212.33/2.79	256.36/2.79	- / 2.79±0.00
Difference	Work-station	0.00/0	0.05/0	0.10/0	0.17/0	0.13/0	- / -
[ms]/[%]	Desktop	0.09/0	0.15/0	0.25/1	0.25/0	0.35/1	- / -
	Laptop	-0.03/0	0.36/0	0.30/0	0.86/0	-0.32/0	- / -
Ratio	Work-station	1.00	1.00	1.00	1.00	1.00	**1.00±0.00**
	Desktop	1.00	1.00	1.01	1.00	1.01	**1.01±0.00**
	Laptop	1.00	1.00	1.00	1.00	1.00	**1.00±0.00**

Tab. 11-2 Results - *Sequential (pre-calculated & pre-stored).*

The timing results for *sequential* scheduling of gridding operations on GPU with data structures pre-calculated and stored in GPU's memory. The results are presented as total time needed to grid a set of frames (including how much each frame accounted to) and their comparison to the equivalent *Lower-limit* result. The timing ratio comparison and timings each frame accounted to were averaged and presented in the last column.

11.5.2 Sequential approach

To reduce the memory consumption, the *sequential* tests allowed only a single gridding matrix to be stored on a GPU. The algorithm sequentially created a matrix in CPU memory related to each frame, which then was used to replace the one in GPU memory (Section 7.2.2). The *naive* strategy (without buffering) showed very poor results which were ~33x, ~39x and ~7x slower (Tab. 11-3) than expected (as compared with the *Lower-limit*). The overlapping version was only marginally better ~33x, ~37x and ~6x (Tab. 11-4). The improvement was connected to a slight shift in the creation of matrix times with which the final timing results were driven. In all cases the serial matrix creation took >98 % of the total time.

-Number of frames		28	44	60	76	92	**Average**
Gridding	Work-station [ms]	889.20	1400.10	1903.20	2410.20	2917.20	
Total	Desktop [ms]	780.00	1224.60	1669.20	2117.70	2570.10	
	Laptop [ms]	569.40	893.10	1216.80	1540.50	1856.40	
Gridding	Work-station [ms]	31.76	31.82	31.72	31.71	31.71	**31.74±0.05**
per Frame	Desktop [ms]	27.86	27.83	27.82	27.86	27.94	**27.86±0.05**
	Laptop [ms]	20.34	20.30	20.28	20.27	20.18	**20.27±0.06**
Create	Work-station [ms]	877.50	1380.60	1899.30	2398.50	2909.41	
Total	Desktop [ms]	760.50	1212.90	1653.60	2090.40	2535.00	
	Laptop [ms]	553.80	877.50	1209.00	1509.30	1836.90	
Create	Work-station [ms]	31.34	31.38	31.66	31.56	31.62	**31.51±0.14**
per Frame	Desktop [ms]	27.16	27.57	27.56	27.51	27.55	**27.47±0.17**
	Laptop [ms]	19.78	19.94	20.15	19.86	19.97	**19.94±0.14**
Create	Work-station [%]	98.68	98.61	99.80	99.51	99.73	**99.27±0.58**
Ratio	Desktop [%]	97.50	99.04	99.07	98.71	98.63	**98.59±0.64**
	Laptop [%]	97.26	98.25	99.36	97.97	98.95	**98.36±0.82**
Create	Work-station [ms]	0.42	0.44	0.07	0.15	0.08	**0.23±0.18**
Difference per	Desktop [ms]	0.70	0.27	0.26	0.36	0.38	**0.39±0.18**
Frame	Laptop [ms]	0.56	0.35	0.13	0.41	0.21	**0.33±0.17**
Comparison	Work-station [ms]	862.37	1358.08	1845.96	2337.76	2829.50	
Difference	Desktop [ms]	759.82	1192.91	1626.01	2063.00	2503.90	
(Lower-limit)	Laptop [ms]	491.18	770.40	1049.43	1329.04	1599.72	
Comparison	Work-station [ms]	30.80	30.87	30.77	30.76	30.76	**30.79±0.05**
Difference per	Desktop [ms]	27.14	27.11	27.10	27.14	27.22	**27.14±0.05**
Frame (Lower-limit)	Laptop [ms]	17.54	17.51	17.49	17.49	17.39	**17.48±0.06**
Ratio	**Work-station**	**33.15**	**33.32**	**33.25**	**33.27**	**33.26**	**33.25±0.06**
(Lower-limit)	**Desktop**	**38.65**	**38.64**	**38.65**	**38.71**	**38.82**	**38.69±0.08**
	Laptop	**7.28**	**7.28**	**7.27**	**7.28**	**7.23**	**7.27±0.02**

Tab. 11-3 Results - *Sequential (naive)*.

Timing results for scheduling of gridding operations on GPU with a single matrix calculated on CPU and no additional buffering. The results comprise total time needed for the whole task (including estimated times each frame accounted to), times creation of all matrices took (including averaged times of each operation) and comparison to the equivalent Lower-limit. The timing ratio comparison and timings each frame accounted to were averaged and presented in the last column.

Number of frames		28	44	60	76	92	**Average**
Gridding	Work-station [ms]	873.60	1396.20	1883.70	2379.00	2960.10	
Total	Desktop [ms]	756.60	1177.80	1610.70	2047.50	2433.60	
	Laptop [ms]	479.70	741.00	1006.20	1275.30	1528.80	
Gridding	Work-station [ms]	31.20	31.73	31.40	31.30	32.18	**31.56±0.40**
per Frame	Desktop [ms]	27.02	26.77	26.85	26.94	26.45	**26.81±0.22**
	Laptop [ms]	17.13	16.84	16.77	16.78	16.62	**16.83±0.19**
Create	Work-station [ms]	873.60	1388.40	1868.10	2363.40	2948.41	
Total	Desktop [ms]	741.00	1170.00	1591.20	2020.20	2402.40	
	Laptop [ms]	471.90	733.20	986.70	1263.60	1501.50	
Create	Work-station [ms]	31.20	31.55	31.14	31.10	32.05	**31.41±0.40**
per Frame	Desktop [ms]	26.46	26.59	26.52	26.58	26.11	**26.45±0.20**
	Laptop [ms]	16.85	16.66	16.45	16.63	16.32	**16.58±0.21**
Create	Work-station [%]	100.00	99.44	99.17	99.34	99.60	**99.51±0.31**
Ratio	Desktop [%]	97.94	99.34	98.79	98.67	98.72	**98.69±0.50**
	Laptop [%]	98.37	98.95	98.06	99.08	98.21	**98.54±0.45**
Create	Work-station [ms]	0.00	0.18	0.26	0.21	0.13	**0.15±0.10**
Difference per	Desktop [ms]	0.56	0.18	0.32	0.36	0.34	**0.35±0.14**
Frame	Laptop [ms]	0.28	0.18	0.33	0.15	0.30	**0.25±0.08**
Comparison	Work-station [ms]	846.77	1354.18	1826.46	2306.56	2872.40	
Difference	Desktop [ms]	736.42	1146.11	1567.51	1992.80	2367.40	
(Lower-limit)	Laptop [ms]	401.47	618.30	838.83	1063.84	1272.12	
Comparison	Work-station [ms]	30.24	30.78	30.44	30.35	31.22	**30.61±0.40**
Difference per	Desktop [ms]	26.30	26.05	26.13	26.22	25.73	**26.09±0.22**
Frame (Lower-limit)	Laptop [ms]	14.34	14.05	13.98	14.00	13.83	**14.04±0.19**
Ratio	**Work-station**	**32.56**	**33.23**	**32.91**	**32.84**	**33.75**	**33.06±0.45**
(Lower-limit)	**Desktop**	**37.49**	**37.16**	**37.29**	**37.43**	**36.76**	**37.23±0.29**
	Laptop	**6.13**	**6.04**	**6.01**	**6.03**	**5.96**	**6.03±0.06**

Tab. 11-4 Results - *Sequential (overlapping).*

Timing results for scheduling of gridding operations on GPU with a single matrix calculated on CPU; with additional buffering. The results comprise total time needed for the whole task (including estimated times each frame accounted to), times creation of all matrices took (including averaged times) and comparison to the equivalent Lower-limit. The timing ratio comparison and timings each frame accounted to were averaged and presented in the last column.

For the overlapping version to make a difference the proportion of creation to transmission time would need to be very close to one. Unfortunately, the creation was up to ~45x, ~41x and ~31x slower in tests.

The assessment of the basic stages of the gridding process was carried out in the course of *sequential* tests. These included *Creation* of a matrix, single *Gridding* operation as a matrix multiplication on GPU and *Copy* as a transfer data structures on a GPU. Tab. 11-5 comprises average times for these operations.

		Work-station	Desktop	Laptop
Creation	[ms]	32.05 ± 0.81	26.63 ± 0.62	17.45 ± 1.52
Gridding	[ms]	1.03 ± 0.00	0.76 ± 0.00	2.96 ± 0.16
Copy	[ms]	0.71 ± 0.05	0.65 ± 0.06	0.55 ± 0.00

Tab. 11-5 Results - Average timings from the *sequential* approach tests.

The average timing results for single calculation of a gridding matrix (*Creation*), transmission of matrix structures onto GPU (*Copy*) and execution of single gridding operation by GPU

Creation was implemented as a serial CPU code. As the Laptop had the highest processor frequency it needed the least time to create the gridding matrices. It was ~2x quicker executing this CPU code than the Work-station machine. However, the Laptop had the slowest GPU used as a coprocessor. Each *Gridding* operation was over 4x slower than execution on the Laptop, and almost 3x slower on the Work-station machine, than the Desktop.

The tests showed discrepancies between the averaged individual gridding times (Tab. 11-5) and estimated per frame times (Tab. 11-2). This was especially visible in the Laptop and Work-station results. It suggests that the adjacent gridding operations (not separated by the data transmissions) were able to achieve some degree of overlap which levelled the cumulative time with the *Lower-limit*. The data transmission could not be avoided, consequently the maximum achievable speed-up reduced to ~17x, ~23x and ~6x (after correcting for the acquired average gridding times; Tab. 11-5).

An important step of the tests was to find out if the data transfers could be overlapped with the longer matrix multiplications on GPU. For this reason, the *Sequential (pre-calculated)* tests (Section 7.2.2) assessed the case in which the matrices were already available in the CPU memory. Each such gridding process involved four tasks; three data transmissions and one matrix multiplication.

The timing results for this test are presented in Tab. 11-6. The times were ~0.4, ~0.8 and ~0.7 ms/frame longer than the batched gridding (the *Lower-limit*) resulting in ~1.4x, ~2.1x and ~1.2x slow-down. Comparison of the results with the transmission times (~0.7 ms, ~0.7 ms and ~0.6 ms; Tab. 11-5) suggested that only Work-station was able to achieve some degree of overlap between tasks. This can be assigned to Work-station's improved architecture and scheduling capabilities. However, partitioning of the data transmission

meant only a portion of the transmission time was overlapped with the matrix multiplication operation.

The *sequential* tests run on the tested hardware revealed that creation of matrices was the major limitation of the new gridding approach for data on non-repeating trajectories. The execution times were dictated by the speed of CPUs.

Only the Laptop achieved time comparable (~0.95x) to the *Upper-limit* (execution on the scanner) and a good speed up (~2.5x) as compared to the *Acquisition time*. Work-station and Desktop were slower than the *Upper-limit*; ~1.8x and ~1.5x and insignificantly quicker than the *Acquisition time*; ~1.3x and ~1.6x respectively. These results reflected the CPUs hardware performance.

Number of frames		28	44	60	76	92	Average
Total / per frame	Work-station	36.43/1.30	59.89/1.36	78.95/1.32	100.36/1.32	120.44/1.31	- / 1.32±0.02
[ms]	Desktop	42.42/1.52	67.55/1.54	89.39/1.49	117.55/1.55	140.80/1.53	- / 1.52±0.02
	Laptop	96.61/3.45	152.04/3.46	206.76/3.45	261.41/3.44	315.05/3.42	- / 3.44±0.01
Difference	Work-station	9.60/0.34	17.87/0.41	21.71/0.36	27.92/0.37	32.74/0.36	- / 0.37±0.02
Total / per frame	Desktop	22.24/0.79	35.86/0.82	46.20/0.77	62.85/0.83	74.60/0.81	- / 0.80±0.02
[ms]	Laptop	18.39/0.66	29.34/0.67	39.39/0.66	49.95/0.66	58.37/0.63	- / 0.65±0.01
Ratio	**Work-station**	1.36	1.43	1.38	1.39	1.37	**1.38±0.03**
	Desktop	2.10	2.13	2.07	2.15	2.13	**2.12±0.03**
	Laptop	1.24	1.24	1.24	1.24	1.23	**1.23±0.00**

Tab. 11-6 Results - *Sequential (pre-calculated)*.

The timing results for sequential scheduling of gridding operations on GPU with data structures pre-calculated and stored in CPU's memory. Results are presented as total time needed to grid a set of frames (including how much each frame accounted to) and their comparison to the equivalent Lower-limit. The timing ratio comparison and timings each frame accounted to were averaged and presented in the last column.

11.5.3 Threaded approach

The *Threaded* approach aimed to reduce the total matrix creation time by dividing the task among multiple of CPU threads (Section 7.2.3). Providing enough computing resources were available the number of concurrently created matrices could be brought to the level sufficing to fully occupy the GPU with work. This way a set of matrices would be prepared while the previous one was used.

However this condition was impossible to meet for all the tested hardware. Work-station was limited to 12 physical processors, and the Desktop and the Laptop to four.

As discussed the number of frames and CPU threads were varied to find the optimal partitioning of the task. The tests findings for all the hardware are presented in Tab. 11-7. Additionally, the table presents the expected processing time and speed up based on the partitioning.

All the CPU processors were capable of hyper-threading (mapping of two virtual processors into one physical); however the tests showed the achieved speed up (~9x, ~4x and ~3x) was significantly lower than expected and would not exceed the number of physical processors. Also, this was reflected in an increase of the average matrix creation time (~45.1 ms, ~44.5 ms and ~34.5 ms). Collected log files showed that some runs were able to achieve the creation times similar to one found in the sequential approach, but

Number of frames		28	44	60	76	92	**Average**
Threads	Work-station	24/2/14.0	21/3/14.7	16/4/15.0	20/4/19.0	21/5/18.4	
/frames per thread	Desktop	12/ 3/ 9.3	12/4/11.0	11/6/10.0	9/9/8.4	8/ 12/ 7.7	
/expected speed-up	Laptop	9/ 4/ 7.0	8/ 6/ 7.3	8/ 8/ 7.5	10/ 8/ 9.5	8/ 12/ 7.7	
Gridding	Work-station [ms]	62.40	95.20	125.58	125.21	160.88	
Expected	Desktop [ms]	81.06	107.07	161.07	242.47	317.43	
	Laptop [ms]	68.53	101.05	134.16	134.24	199.41	
Gridding	Work-station [ms]	106.97	162.69	211.71	262.97	319.80	
Total	Desktop [ms]	205.03	312.00	403.37	503.66	601.72	
	Laptop [ms]	156.00	220.63	291.94	369.94	469.49	
Gridding	Work-station [ms]	3.82	3.70	3.53	3.46	3.48	**3.60±0.16**
per Frame	Desktop [ms]	7.32	7.09	6.72	6.63	6.54	**6.86±0.33**
	Laptop [ms]	5.57	5.01	4.87	4.87	5.10	**5.08±0.29**
Create time	Work-station [ms]	98.06	160.46	201.69	258.51	316.46	
Acquired	Desktop [ms]	200.57	305.31	403.37	499.20	597.26	
	Laptop [ms]	149.32	209.49	276.34	354.34	455.87	
Achieved	Work-station [ms]	8.17	8.58	8.90	9.05	9.26	**8.79±0.43**
Speed-up	Desktop [ms]	3.69	3.78	3.99	4.07	4.04	**3.91±0.17**
	Laptop [ms]	3.07	3.36	3.45	3.45	3.26	**3.32±0.16**
Comparison	Work-station [ms]	80.26	120.83	154.63	190.76	232.46	
Difference	Desktop [ms]	184.91	280.36	360.30	449.11	535.66	
(Lower-limit)	Laptop [ms]	77.88	97.80	124.58	157.63	212.85	
Comparison	*Work-station* [ms]	2.87	2.75	2.58	2.51	2.53	**2.65±0.15**
Difference per	Desktop [ms]	6.60	6.37	6.00	5.91	5.82	**6.14±0.33**
Frame (Lower-limit)	Laptop [ms]	2.78	2.22	2.08	2.07	2.31	**2.29±0.29**
Ratio	**Work-station**	**4.01**	**3.89**	**3.71**	**3.64**	**3.66**	**3.78±0.16**
(Lower-limit)	**Desktop**	**10.19**	**9.86**	**9.36**	**9.23**	**9.11**	**9.55±0.46**
	Laptop	**2.00**	**1.80**	**1.74**	**1.74**	**1.83**	**1.82±0.10**

Tab. 11-7 Results - *Threaded*

Timing results for scheduling of gridding operations on GPU with parallel matrix calculations on CPU. Apart of the acquired results the table presents estimation of potential speed up and total time based on the number of worker threads, the size of a task and the best results from the sequential approach. The rest are total time needed for the whole task (including estimated times each frame accounted to), total matrices creation, achieved speed-up (as compared to the sequential approach) and comparison to the equivalent Lower-limit. The timing ratio comparison and timings each frame accounted to were averaged and presented in the last column.

the rest required significantly longer time (about double the time). This behaviour was observed regardless of the number of concurrently scheduled threads and increased with total number of matrix creations. Presumably, execution of some threads assigned to the same physical processor caused an interrupted execution due to insufficient computing resources. Consequently, the total time needed to create all of the matrices was greater than expected; as based on the total number of threads involved in the task (Tab. 11-7).

The next step of the tests was to run the *Threaded* approach using pre-calculated matrices stored in CPU memory (as discussed in Section 7.2.3). The impact of the multi-threading was notable only for the Work-station and Desktop computers, as compared with the *Sequential (pre-calculated)* approach (Tab. 11-6); however the Desktop's improvement was not very significant. As previously stated, providing all data structures were pre-calculated in CPU's memory (excluding the necessity of costly matrix coefficients calculations) the Work-station machine could achieve almost the maximum expected speed up of ~17x. The Desktop machine achieved only ~13x as compared with the expected maximum speed up of ~25x.

The *threaded* results for the Work-station suggested that all the data transmission operations were almost completely hidden with the longer matrix multiplication operations (Tab. 11-8), as in contrast to the *sequential* test (Tab. 11-6). Presumably, the multithreaded scheduling of GPU operations resulted in more beneficial ordering and consequently in better utilisation of the GPU. Comparison of the time difference per frame, with the average memory transmission time, suggested the Desktop and Laptop's GPUs were not able to repeat the Work-station's result, which can be assigned to their hardware capabilities.

Number of frames		28	44	60	76	92	Average
Total / per frame	Work-station	30.09/1.07	46.80/1.06	62.40/1.04	78.00/1.03	93.60/1.02	- / 1.04±0.02
[ms]	Desktop	35.66/1.27	62.40/1.42	80.23/1.34	102.51/1.35	124.80/1.36	- / 1.35±0.05
	Laptop	95.83/3.42	156.00/3.55	207.26/3.45	266.31/3.50	322.77/3.51	- / 3.49±0.05
Difference	Work-station	3.38/0.12	4.94/0.11	5.32/0.09	5.79/0.08	6.26/0.07	- / 0.09±0.02
Total / per frame	Desktop	15.53/0.55	30.76/0.70	37.15/0.62	47.97/0.63	58.75/0.64	- / 0.63±0.05
[ms]	Laptop	17.71/0.63	33.17/0.75	39.90/0.66	54.00/0.71	66.14/0.72	- / 0.70±0.05
Ratio	**Work-station**	1.13	1.12	1.09	1.08	1.07	**1.10±0.02**
	Desktop	1.77	1.97	1.86	1.88	1.89	**1.88±0.07**
	Laptop	1.23	1.27	1.24	1.25	1.26	**1.25±0.02**

Tab. 11-8 Results - *Threaded (pre-calculated)*.

The timing results for scheduling of gridding operations on GPU with data structures pre-calculated and stored in CPU's memory. Results are presented as total time needed to grid a set of frames (including how much each frame accounted to) and their comparison to the equivalent Lower-limit. The timing ratio comparison and timings each frame accounted to were averaged and presented in the last column.