Andreas Schwarzl, BSc

Fast Regularized Reconstruction for dynamic MR Imaging using Infimal Convolution of TV Type Functionals

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl-Ing. Dr.techn. Rudolf Stollberger

Institute for Medical Engineering
Graz University of Technology

Second Supervisor: Dipl-Phys. Matthias Schlögl

Graz, April 2016

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Abstract: Fast Regularized Reconstruction for dynamic MR Imaging using Infimal Convolution of TV Type Functionals

Dynamic Magnetic Resonance Imaging (dMRI) aims to visualize time-dependent physiological processes within the human body. Important examples are functional cardiac imaging, dynamic contrast-enhanced (DCE) MRI or time-resolved angiography among others. Since fast acquisition schemes already reached physical admissible limits in terms of nerve stimulation and SAR, in-vivo applications are always bounded by a trade-off between spatial and temporal resolution or spatial coverage depending on the time-scale of the motion under investigation. In order to improve the described limitations, undersampling strategies in combination with recent advances in iterative MRI reconstruction based on compressed sensing and parallel imaging are applied.

This project focuses on the fast implementation of a novel reconstruction technique termed "Infimal Convolution of Total Generalized Variation functionals (ICTGV)" as sophisticated regularization functional in a variational setting to recover diagnostically valuable image quality from undersampled dMRI data. The software presented in this work utilizes GPU functionality by CUDA to accelerate an existing MATLAB implementation in a fast parallel manner.

Two clinical relevant imaging scenarios are examined to compare the performance of the software to the MATLAB based CPU reference implementation. Furthermore, the utilization of a new vendor independent MRI raw data format is demonstrated, to facilitate integration into the clinical work-flow. The results in terms of GPU execution time exhibits clinical acceptable reconstruction times and possible speedups up to a factor of 40. In conclusion usage limitations are discussed and further improvements are proposed.

Keywords: Dynamic MRI, Regularized Reconstruction, ICTGV, CUDA, Cardiac CINE, DCE Perfusion

Kurzfassung: Schnelle regularisierte Rekonstruktion von dynamischen MR Daten unter Verwendung der Infimalen Faltung von TV-Typ Funktionalen.

Die Darstellung dynamischer Vorgänge, wie u.a. der Herzfunktion, kontrastverstärkter Perfusionsbilder oder zeitaufgelöster angiografischer Daten, stellt einen wichtigen klinischen Aspekt der dynamischen Magnetresonanztomografie dar. Die Anwendung dieser Modalitäten erfordert jedoch, in Abhängigkeit der benötigten Aufnahmegeschwindigkeit, eine Abstimmung der räumlichen und zeitlichen Auflösung bzw. eine Einschränkung der abgebildeten Bildregion. Da neueste Bildgebungssequenzen bereits an den Grenzen der erlaubten peripheren Nervenstimulation und SAR-Aufnahme implementiert sind, werden moderne Unterabtaststrategien in Kombination mit auf Compressed Sensing und Parallel Imaging basierenden iterativen Rekonstruktionen zur Verbesserung der genannten Einschränkungen verwendet.

Diese Arbeit setzt eine neue Methode der dynamischen Regularisierung unter Verwendung von Variationsmodellen ein, der sogenannten "Infimal Convolution of Total Generalized Variation functionals (ICTGV)", um diagnostisch wertvolle Bildinformation aus den unterabgetasteten dynamischen Daten zu gewinnen. Die daraus erstellte Software ist eine schnelle Implementierung eines bestehenden MATLAB Projekts unter Einsatz von GPU Funktionalitäten mittels CUDA.

Die Software wird anhand zwei klinisch relevanter Anwendungsfälle evaluiert und anhand der Rekonstruktionszeiten mit der CPU Referenzlösung verglichen. Zusätzlich wird die Verwendung eines herstellerunabhängigen Datenformats präsentiert, das die Integration in den klinischen Workflow ermöglicht. Die Ergebnisse zeigen klinisch implementierbare Beschleunigungen im Vergleich zur Referenz von bis zu 40. Abschließend werden grundlegende Anwendungseinschränkungen diskutiert und mögliche Verbesserungenvorschläge gegeben.

Keywords: Dynamisches MRT, Regularisierte Rekonstruktion, ICTGV, CUDA, MR-Herzbildgebung, DCE Perfusionsbildgebung

Contents

List of Figures

1 Introduction

This chapter gives an introduction to the basic principles of dynamic Magnetic Resonance Imaging (dMRI) and to two basic acquisition strategies, which are typically implemented in clinical practice. In this context, variational reconstruction techniques are introduced and the aim of this work, a fast implementation of a novel reconstruction technique for dMRI, is presented.

1.1 Dynamic MRI

Dynamic MRI aims the visualization of dynamically changing processes within the clinical context, ranging from the morphological motion of the beating heart or its valves to the visualization of organ perfusion by the use of contrast enhancing agents and the general application of Magnetic Resonance Angiography (MRA).

All these imaging types require the acquisition of a time series with a high temporal resolution, in order to suppress motion artifacts due to respiration or patient movement. Since MRI scan times are restricted by physiological, in terms of peripheral nerve stimulation an specific absorption rate (SAR), and hardware limits, modern subsampling strategies have to be applied in order to leverage the speed of current acceleration techniques like fast acquisition sequences and parallel imaging. Recent implementations in the field of dMRI are found in the application of the well-known SENSE framework [1] to the time domain as shown in [2]–[4], the application of compressed sensing techniques considering the time domain [5], [6] and further the application of low-rank and sparse (L+S) reconstructions [7].

Generally, the overall acquisition time in dMRI is aimed to be as short as possible to allow clinical feasible acquisitions which can be performed during a single breath-hold (10-15 seconds) and with best motion artifact suppression. In case of uncooperative patients, breathing and motion artifacts have to be corrected with retrospective registration strategies. In both cases, accelerated acquisition strategies help to improve

the spatio-temporal resolution and image quality. This work refers to two common imaging techniques in this setting, namely cardiac CINE imaging and dynamic contrast enhanced perfusion, whose fundamentals are presented in the following two sections. The application of data subsampling strategies, which largely violate the Nyquist theorem, requires sophisticated regularization techniques in order to compensate artifacts introduced by undersampling. Section 1.3, Page 7 gives an overview of the regularization functions employed in this work.

1.1.1 Cardiac CINE

The capturing of a time-series (movie) of the functional heart motion is referred to as cardiac CINE imaging [8]. The movement during a heart beat is divided into multiple subsequent phases (frames), which are composed to be viewed as a movie. Since the motion of a single heart beat is considerably faster than the acquisition of sufficient k-space lines for each phase, the k-space has to be filled by collecting data of several subsequent cardiac cycles. In order to synchronize the acquisition to the heart motion electrocardiogram (ECG) triggering is applied prospectively or retrospectively in the sense of ECG gating.

Figure 1.1 depicts the process of a cardiac CINE acquisition. The first row shows an ECG signal with several time-slots denoting the different phases of the heartbeat. Typically, the acquisition is triggered by the detection of the distinctive R wave. Depending on the implemented pulse sequence interleaved Cartesian or non-Cartesian lines are acquired, while non-Cartesian trajectories, e.g. radial spokes, are considered to be less prone to motion artifacts. In this figure the subsequent collection of k-space lines for only the first cardiac phase is presented. The acquisition of the distinct phases is performed analogously.

After reconstruction of all phases the data can be viewed as a looped movie, to simulate the periodicity of the beating heart. The CINE can give different functional information like cardiac wall movement during contraction, valve behavior and chamber perfusion.

1.1.2 Dynamic Contrast-Enhanced Perfusion

Dynamic contrast-enhanced (DCE) perfusion imaging is the visualization of an intensity variation over time by using contrast enhancing agents [9]. In general, acquisitions

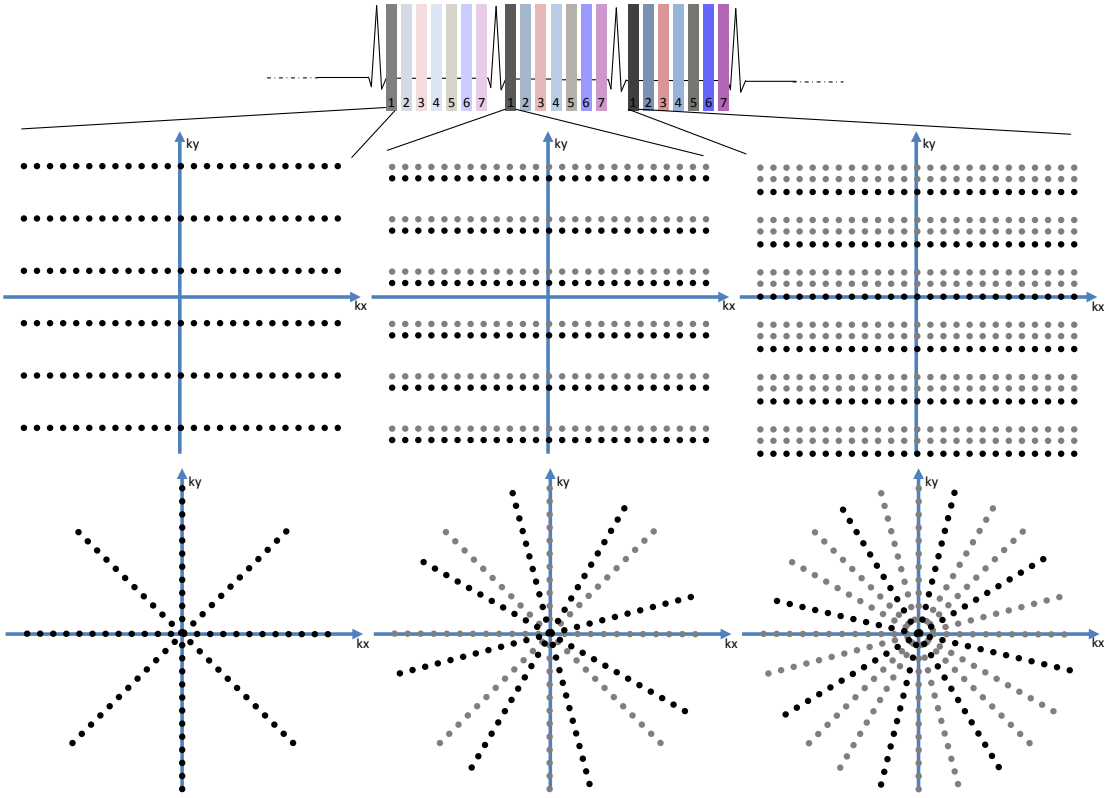


Figure 1.1: Cardiac CINE: Explanation of the interleaved acquisition of multiple phases (here 7 frames) during the cardiac cycle. The ECG-triggered acquisition shows that multiple cardiac cycles are required to fill the k-space for each phase. The middle row shows the collection of Cartesian and the bottom row of radial acquisitions solely to fill the k-space for the first cardiac phase. Every cardiac phase is acquired analogously.

are performed before, during and after the application of the contrast agent. Since many applications need to evaluate the first-pass of the contrast agent through the investigated tissue or organ, a high temporal resolution is mandatory for best image quality.

Typically, the DCE perfusion evaluates the distribution of the contrast agent over time in order to support diagnosis. The difference compared to cardiac CINE imaging is that no repetitive acquisition of the same signal phase can be performed, since the contrast agent is usually applied only once. The temporal resolution largely depends on the amount of necessary k-space readouts per time-frame, in contrast to the need of an sufficiently acquired spatial k-space for good image quality, hence there exists a definite trade-off between spatial and temporal resolution.

In case of cardiac perfusion imaging, multiple slices are acquired simultaneously, where in general three short-axis views and one four-chamber view time-frames are fully sampled during one heart beat, but with low spatial resolution. The main benefit of undersampling strategies lies in the possibility to increase spatial resolution and the spatial coverage during on heart beat (e.g. seven short-axis views) in order to improve the analysis for perfusion defects. In contrast to that, DCE applications within the abdomen, e.g. to detect HCC liver tumors, benefit from increased temporal resolution, which enables a better characterization of tumor lesions.

1.2 Sampling Strategies

Apart from the implementation of the various fast imaging sequences, vendors typically offer similar enhanced k-space filling options, in order to reduce the total number of acquisitions and hence the acquisition time. This section gives a short overview of three methods, which are considered in this work, namely the rectangular field of view (FOV) method, the Partial-Fourier sampling and the Asymmetric Echo (AE) acquisition.

1.2.1 Rectangular FOV

The rectangular FOV method uses different sampling rates in phase and frequency encoding direction, where typically the phase encoding step size (gradient increment) is increased and hence the total amount of acquired lines is decreased. This leads

to a rectangular shaped acquisition matrix ($N_x \neq N_y$) and a reduced FOV in the corresponding direction. Since the FOV relates to the sampling rate in k-space as

$$\begin{aligned} FOV_x &= \frac{1}{\Delta k_x} \\ FOV_y &= \frac{1}{\Delta k_y} \end{aligned} \tag{1.1}$$

the FOV is reduced by the reciprocal of the change of Δk in phase encoding direction. For example, a doubled step size halves the FOV in the corresponding direction. The image resolution ($\Delta y = FOV_y / N_y$) is not affected by the rectangular acquisition. Figure 1.2 shows an example k-space employing the rectangular FOV method with a reduced amount of lines in the phase encoding direction (k_y) resulting in a rectangular shaped acquisition matrix.

Typically, the rectangular FOV phase encoding is applied in directions where the anatomy fits into smaller matrix sizes and does not lead to backfolding artifacts. Frequency encoding is chosen in directions where the anatomy extends over the matrix size (FOV) and backfolding is additionally suppressed by oversampling the read-out in this direction.

1.2.2 Partial Fourier

The Partial Fourier method uses the assumption of redundancy in k-space, in order to partially fill the k-space and synthesize missing data by conjugate symmetry and advanced phase correcting algorithms, like Homodyne detection, the Cuppen implementation or projection onto convex sets (POCS) [10]. Similarly to the rectangular FOV method, the main focus is to reduce the total number of acquisitions and therefore to reduce overall scan time.

Figure 1.3 shows an example of a partially filled k-space. The bright dots indicate not acquired but synthesized k-space points. The fact that field inhomogeneities and phase errors always occur in practice, requires symmetric sampling of the contrast regions in the k-space center. According to that and depending on the synthesizing method, Partial Fourier images can be acquired by measuring as far as only 60% of the full k-space.

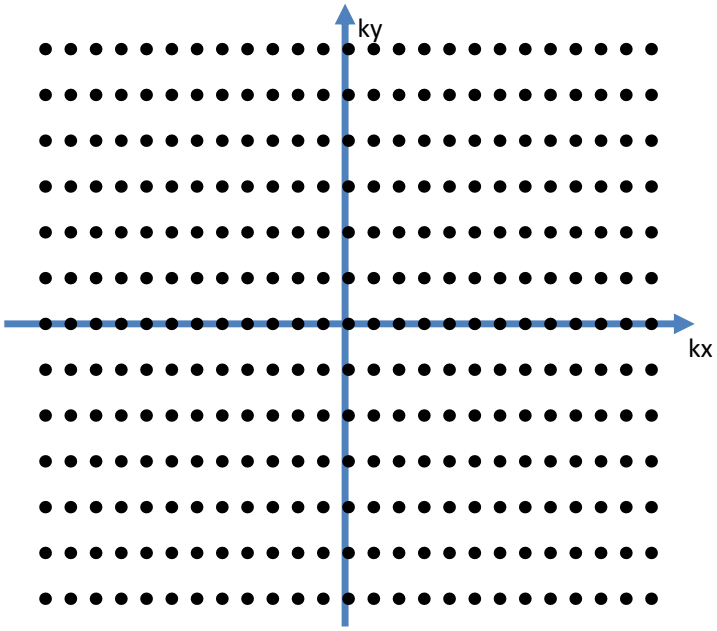


Figure 1.2: Rectangular FOV acquisition. A decreased k-space sampling rate in the phase encoding direction k_y reduces the imaged FOV in the corresponding direction, but the rectangular-shaped matrix size leads to an accelerated total acquisition time.

1.2.3 Asymmetric Echoes

While the Partial Fourier method reduces the total number of acquired phase encoding lines, the Asymmetric Echo method reduces the amount of samples in frequency encoding direction by not sampling the full signal (echo) symmetrically around the k-space center. Figure 1.4 depicts this strategy, where an asymmetrically acquired k-space is shown (dark dots). In order to be less error prone, again only a small fraction around the k-space center is acquired symmetrically. In general, this approach reduces the echo time (TE) of each readout, since the dephasing readout gradient is not applied in the full extent.

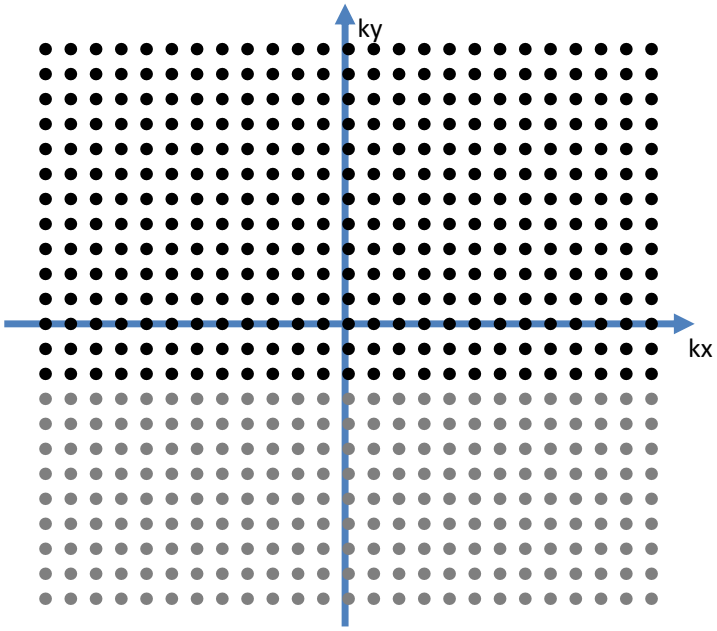


Figure 1.3: Partial Fourier acquisition. The k-space is filled partially in phase encoding direction (k_y). Slightly more than one half of the lines are acquired (dark dots) while the missing parts (bright dots) are synthesized by using conjugate symmetry assumptions.

1.3 Variational Reconstruction Techniques

1.3.1 Overview

Variational reconstruction techniques gained more and more importance in the field of MRI. Starting with the application of the theory for nonlinear noise removal [11], particularly the application of variational regularization functions is found widely spread in undersampled reconstructions, like shown in [12]–[14]. This section gives a short introduction to the application of variational methods in the field of regularized dynamic MRI reconstruction of undersampled data.

Considering the dynamic setting, this work employs three different regularization methods, namely the spatio-temporal Total Variation (TV, [11]) functional, the Second Order spatio-temporal Total Generalized Variation (TGV^2, [15]) and the Infimal Convolution of TGV type functionals ($ICTGV$, [16]).

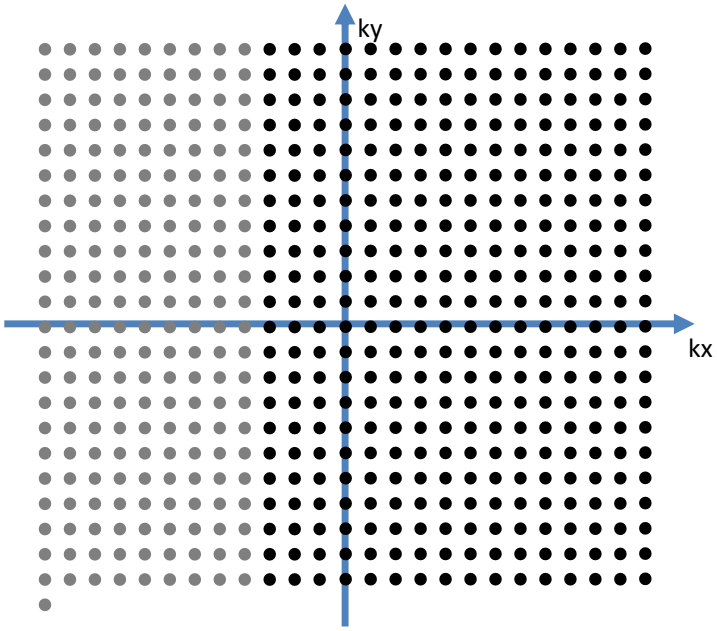


Figure 1.4: Asymmetric Echo acquisition. The echo time is reduced by acquiring asymmetrically around the k-space center in frequency encoding direction (k_x). The dark dots indicate the acquired samples, while the skipped and synthesized fractions are shown as bright dots.

1.3.2 Infimal Convolution Of TV type Functionals

Regarding to the regularization method proposed in [16], its concrete application to the dynamic MRI setting [17] is considered. Generally, it is aimed to solve the following basic optimization problem:

$$\min_u \frac{\lambda}{2}\|\mathcal{K}u - d\|_2^2 + \mathcal{R}(u) \tag{1.2}$$

where u is the unknown image data, $\mathcal{K}$ the MR specific forward Fourier operator, d the measured multi-coil data, λ the basic regularization parameter and $\mathcal{R}$ the applied regularization function in the image domain.

Following [18] the Total Variation (TV) regularization term in the discrete setting is defined as:

$$\mathcal{R}(u) = \mathrm{TV}_\beta(u) = \left\| |\nabla u|_\beta \right\|_1 \tag{1.3}$$

Additionally, an extra regularization parameter $\beta = \frac{ds}{dt}$ is introduced to the derivative operator, which defines the scaling between spatial and temporal derivatives:

$$\mathrm{TV}_{\beta}(u) = \left\| |\nabla u|_{\beta} \right\|_1 = \sqrt{(ds\delta_x u)^2 + (ds\delta_y u)^2 + (dt\delta_t u)^2} \tag{1.4}$$

The Second Order Total generalized variation (TGV^2) in the discrete setting is expressed as:

$$\mathcal{R}(u) = \mathrm{TGV}^2_{\beta_1,\beta_2}(u) = \min_v \alpha_1 \left\| |\nabla u - v|_{\beta_1} \right\|_1 + \alpha_0 \left\| |\mathcal{E} v|_{\beta_2} \right\|_1 \tag{1.5}$$

where $\mathcal{E} v$ is defined as a symmetric derivative operator:

$$\mathcal{E} v = \frac{1}{2}\left(\nabla v + \nabla v^T\right) \tag{1.6}$$

While TV and TGV^2 basically compromise different weighting of spatial and temporal domain with extra regularization parameter β, the newly introduced $ICTGV^2$ functional finds a good trade-off between spatial consistence and high temporal resolution in dynamic parts of the image by utilizing the Infimal Convolution of two differently scaled TGV^2 functions, defined as:

$$\mathcal{R} = \mathrm{ICTGV}^2_{\beta,\gamma}(u) = \min_v \gamma_1 \mathrm{TGV}^2_{\beta_1}(u - v) + \gamma_2 \mathrm{TGV}^2_{\beta_2}(v) \tag{1.7}$$

Here, γ_1 weights the influence of the temporal inconsistent, dynamic image parts (moving foreground) and γ_2 the image regions with time constant content (background).

1.3.3 Dynamic MRI Reconstruction

This section shows the application of the above mentioned regularization functions to the setting of dynamic MRI reconstruction. Equation (1.2) can be extended to the following form, where the unknown image sequence u is derived by solving the following optimization problem

$$u^* = \arg\min_u \frac{\lambda}{2} \|\mathcal{K}u - d\|_2^2 + \mathcal{R}(u) \tag{1.8}$$

and using the following definitions:

- $u \in \mathbb{C}^{NxMxT}$ Image sequence
- $c \in \mathbb{C}^{NxMxC}$ Estimated coil sensitivities
- $d \in \mathbb{C}^{\tilde{N}x\tilde{M}xCxT}$ Multi-channel k-space data
- $\mathcal{K}$ MR operator
- $\mathcal{F} : \mathbb{C}^{NxM} \rightarrow \mathbb{C}^{NxM}$ Cartesian Fourier operator
- $\tilde{\mathcal{F}} : \mathbb{C}^{NxM} \rightarrow \mathbb{C}^{\tilde{N}x\tilde{M}}$ Non-Cartesian Fourier operator
- $\mathcal{M}_t \in \{0,1\}^{NxMxT}$ Sampling pattern of time frame
- N, M Image dimensions
- T Number of time frames
- C Number coil channels
- $\tilde{N}, \tilde{M}$ K-space encoding dimensions, Cartesian: $(N, M) = (\tilde{N}, \tilde{M})$

Here, $\mathcal{K}$ is defined as the MR forward operator, consisting of the masked forward Fourier operation of multi-coil images per time frame for the Cartesian case or of the non-uniform Fourier transform (NUFFT, [19]) in case of non-Cartesian sampling:

$$\begin{aligned} K : (u, c_i)_t &\mapsto \mathcal{M}_t \mathcal{F} \Big\{ c_i \cdot u_t \Big\}_{i=1\dots C, t=1\dots T} \\ K : (u, c_i)_t &\mapsto \tilde{\mathcal{F}}_t \Big\{ c_i \cdot u_t \Big\}_{i=1\dots C, t=1\dots T} \end{aligned} \tag{1.9}$$

Thus, the Fourier operator for a certain time point $t = t_i$ transforms the sensitivity corrected time frame to the multi-coil data domain including the application of the k-space sampling pattern M_{t_i}:

$$\mathcal{F}_{t=t_i}(u_{t_i}, c) := \mathcal{M}_{t_i} \left(\mathcal{F} \Big\{ u_{t_i} \cdot c_1 \Big\}, \cdots, \mathcal{F} \Big\{ u_{t_i} \cdot c_C \Big\} \right) = d_{t_i} \tag{1.10}$$

1.3.4 Primal-Dual Algorithm

Regarding to [18] and following the successful application of primal-dual algorithms to MRI reconstructions [12], the basic nonlinear optimization problem is assumed to be in the form of

$$\min_{x \in X} F(\mathcal{K}x) + G(x) \tag{1.11}$$

where F and G are a convex lower-semicontinuous functions and $\mathcal{K}$ is a linear operator. Equation(1.11) can be transformed to the saddle-point formulation

$$\min_{x \in X} \max_{y \in Y} \langle \mathcal{K}x, y \rangle + G(x) - F^*(y) \tag{1.12}$$

in order to be solved with an efficient algorithm presented in [18]. Its outline is given in Algorithm 1. Here, $F^*(y)$ denotes the convex conjugate of $F(x)$, i.e. $F^*(y) = \sup_x \langle x, y \rangle - F(x)$.

Algorithm 1 General formulation of the Primal dual algorithm by Chambolle-Pock [18]

1: **Initialize**:
2: $(x^0, \bar{x}^0, y^0) \qquad \sigma, \tau \geq 0 \qquad \theta \in [0, 1]$
3: **while** Stopping criteria are not met **do**:
4: **Dual Update:**
5: $y^{n+1} \leftarrow (I + \sigma\delta\, F^*)^{-1}(y^n + \sigma\mathcal{K}\bar{x}^n)$
6: **Primal Update**:
7: $x^{n+1} \leftarrow (I + \tau\delta\, G)^{-1}(x^n - \tau\, K^*\, y^{n+1})$
8: **Extragradient step**:
9: $\bar{x}^{n+1} \leftarrow x^{n+1} + \theta(x^{n+1} - x^n)$

A common case is to choose $\theta = 1$. In order to apply the primal-dual algorithm successfully, the resolvent operators (proximal maps) for F^* and G have to be defined by

$$\begin{aligned} x = prox_{\tau F}(y) &= (I + \tau\delta\, F)^{-1}(y) \\ &= \arg\min_x \frac{\|x - y\|^2}{2\tau} + F(x) \end{aligned} \tag{1.13}$$

and typically are considered to be easy computable in a closed form solution. The convergence of the iterative reconstruction can be monitored with the computation of the so-called primal dual gap, defined as:

$$G(x, y) = F(\mathcal{K}x) + G(x) + (G^*(-\mathcal{K}^* y) + F^*(y)) \tag{1.14}$$

The concrete algorithms for the different considered regularization problems are derived in Section 2, Page 13.

1.4 GPU computation for dMRI

In recent years many iterative algorithms in the field of undersampled MRI reconstructions have been implemented. One common problem is that the reconstruction times are long by trend without further improvements and not fitting clinically acceptable time limits. Therefore, the utilization of GPUs for parallelized image reconstruction gained more and more importance. The fact that comparable cheap commodity graphics hardware can be utilized to perform hundreds of computations in parallel is exploited to speed up time-consuming reconstruction steps or to perform whole reconstructions on graphics devices.

In order to implement a reconstruction in parallel (either as a CPU multi-core or GPU solution), some basic requirements have to be met. Therefore, it is only sensible to consider parallel design if computations can be performed independently in parallel or if the effort for merging single results is small compared to the computational cost of each single computation.

As shown in [12], the application of iterative primal-dual solvers for GPU-based MR reconstructions suit well for these requirements, since many operations can be performed on a per-element (pixel) basis. Finite differences or fundamental vector algebra functions are some basic examples to mention in this context. Only the evaluation of vector measures like norm computations or the computation of the MR operator require synchronized data access. One major drawback of primal-dual solvers is that depending on the problem size a huge amount of memory, thus memory that has to be available on the GPU device, is required.

This work employs the Compute Unified Device Architecture (CUDA) developed and supported by NVIDIA (NVIDIA Corporation, Santa Clara, USA), in order to port a MATLAB (The MathWorks Inc., Natick, USA) based dMRI reconstruction to C++ utilizing GPUs. Section 2.4, Page 28, gives more details on memory limitations and on the parallelization strategy.

2 Methods

This section comprises the derivation of the primal-dual formulations and the corresponding algorithms for the different regularization methods explained in Section 1, Page 1. Moreover, the necessary model parameters are introduced and an explanation of the coil calibration method used in this work is presented. Subsequently, details to the CUDA implementation and the testing environment are given.

2.1 dMRI Regularized Reconstruction

In order to derive the algorithms for the three different regularization methods (TV, TGV^2 and $ICTGV$) the optimization problems are transformed into the primal-dual form of (1.12). For the sake of simplicity, the mentioned spatio-temporal scaling β of the derivative operator described in (1.3) is disregarded.

2.1.1 TV regularization

$$\begin{aligned} u^* &= \arg\min_u \frac{\lambda}{2}\|\mathcal{K}u - d\|_2^2 + \mathrm{TV}_\beta(u) \\ &= \arg\min_u \frac{\lambda}{2}\|\mathcal{K}u - d\|_2^2 + \|\nabla u\|_1 \end{aligned} \tag{2.1}$$

Putting (2.1) in the form of (1.12) with $F(Kx) = \|\nabla u\|_1$ and $G(x) = \frac{\lambda}{2}\|\cdot\|_2^2$ the problem is given as

$$u^* = \arg\min_{u\in X}\max_{p\in Y} -\langle u, \mathrm{div}\, p\rangle - \delta_P(p) + \frac{\lambda}{2}\|\mathcal{K}u - d\|_2^2 \tag{2.2}$$

where $X : \{x \in \mathrm{C}^{NxMxT}\}$ and representing the domain of the derivative operator $\nabla : X \to Y$ with $Y : \{y \in X^3\}$. The convex conjugate $F^*(y)$ of $\|\cdot\|_1$ is the so-called indicator function δ_P, defined as:

$$\delta_P(p) = \begin{cases} 0, & \text{if } p \in P \\ \infty, & \text{if } p \notin P \end{cases} \tag{2.3}$$

where the convex set P is defined as: $P = \{p \in Y : \|p\|_\infty \leq 1\}$

In the discrete setting, the infinity norm is defined as $\|p\|_\infty = \max_{i,j} |p_{i,j}|$ Furthermore, the following identity is used:

$$\langle \nabla u, p \rangle = -\langle \mathrm{div} p, u \rangle \tag{2.4}$$

The functional part $G(x)$ needs also to be dualized, since due to the undersampling the proximal operator $(I + \tau G)^{-1}(x - \ldots)$ can not be computed in a closed form solution. This yields the final form of the primal-dual problem with TV regularization as:

$$u^* = \arg\min_{u \in X} \max_{p \in Y, r \in Z} -\langle u, \mathrm{div} p \rangle - \delta_P(p) + \langle \mathcal{K}u - d, r \rangle - \frac{1}{2\lambda}\|r\|_2^2 \tag{2.5}$$

with $Z : \{z \in \mathrm{C}^{\tilde{N}x\tilde{M}xCxT}\}$ and the convex conjugate $G^*(r)$ derived as:

$$\begin{aligned} G^*(r) &= \sup_{\tilde{r}} \langle r, \tilde{r} \rangle - G(\tilde{r}) = \sup_{\tilde{r}} \langle r, \tilde{r} \rangle - \frac{\lambda}{2}\|\tilde{r}\|_2^2 \\ &\to r - \lambda\|\tilde{r}\|_2 = 0 \to \tilde{r} = \frac{r}{\lambda} \\ &\to G^*(r) = \frac{\langle r, r \rangle}{\lambda} - \frac{1}{2\lambda}\|r\|_2^2 \leq \frac{1}{2\lambda}\|r\|_2^2 \end{aligned} \tag{2.6}$$

Proximal operators

In order to apply the primal-dual algorithm, the resolvent operators for F^* and G^* have to be defined following Equation 1.13 as:

$$\begin{aligned} prox_{\sigma F^*}(\tilde{p}) &= (I + \sigma\delta F^*)^{-1}(\tilde{p}) \\ &= proj_{\sigma F^*}(\tilde{p}) = \frac{\tilde{p}}{\max\left(1, \frac{|\tilde{p}|}{\sigma}\right)} \end{aligned} \tag{2.7}$$

$$\begin{aligned} prox_{\sigma G^*}(\tilde{r}) &= (I + \sigma\delta G^*)^{-1}(\tilde{r}) \\ &= \min_r \frac{\|r - \tilde{r}\|_2^2}{2} + \frac{\sigma}{2\lambda}\|r\|_2^2 \Rightarrow r = \frac{\tilde{r}}{1 + \frac{\sigma}{\lambda}} \end{aligned} \tag{2.8}$$

Algorithm

Collecting all information from above, Algorithm 2 shows the complete outline of the dMRI-TV regularization problem.

Algorithm 2 Primal dual algorithm to solve the dMRI-TV regularization problem

Initialize:
$(u^0, \bar{u}^0, p^0, r^0) \qquad \sigma, \tau \geq 0$

while Stopping criteria are not met **do**:
Dual Update:
$p^{n+1} \leftarrow proj_{\sigma F^*}(p^n + \sigma(\nabla \bar{u}^n))$
$r^{n+1} \leftarrow prox_{\sigma G^*}(r^n + \sigma(K\bar{u}^n - d))$

Primal Update:
$u^{n+1} \leftarrow u^n - \tau(-div(p^{n+1}) + K^* r^{n+1})$

Extragradient step:
$\bar{u}^{n+1} \leftarrow 2u^{n+1} - u^n$

2.1.2 TGV2 regularization

$$\begin{aligned} u^* &= \arg\min_u \frac{\lambda}{2}\|\mathcal{K}u - d\|_2^2 + \mathrm{TGV}_\beta^2(u) \\ &= \arg\min_{u,v} \frac{\lambda}{2}\|\mathcal{K}u - d\|_2^2 + \alpha_1\|\nabla u - v\|_1 - \alpha_0\|\mathcal{E}v\|_1 \end{aligned} \tag{2.9}$$

Putting (2.9) in the form of (1.12) results in

$$
\begin{aligned}
u^* = \arg \min_{u \in X, v \in Y} \max_{p \in Y, q \in W, r \in Z} & \langle \nabla u - v, p \rangle - \frac{1}{\alpha_1} \delta_P(p) + \langle \mathcal{E} v, q \rangle - \frac{1}{\alpha_0} \delta_Q(q) \\
& + \langle \mathcal{K} u - d, r \rangle + \frac{1}{2\lambda} \|r\|_2^2 \\
= \arg \min_{u \in X, v \in Y} \max_{p \in Y, q \in W, r \in Z} & - \langle u, \operatorname{div} p \rangle - \langle v, p \rangle - \frac{1}{\alpha_1} \delta_P(p) \\
& - \langle v, \operatorname{div}_{\mathcal{E}} q \rangle - \frac{1}{\alpha_0} \delta_Q(q) \\
& + \langle \mathcal{K} u - d, r \rangle + \frac{1}{2\lambda} \|r\|_2^2
\end{aligned}
\tag{2.10}
$$

with the domain of the symmetric derivative $\mathcal{E} : Y \to W$ with $W : \{w \in X^6\}$. The following identity in the discrete setting is used:

$$
\langle \mathcal{E} v, q \rangle = -\langle v, \operatorname{div}_{\mathcal{E}}(q) \rangle \tag{2.11}
$$

Algorithm

An algorithmic outline of the dMRI-TGV^2 regularization problem is given in Algorithm 3.

2.1.3 ICTGV regularization

$$
u^* = \arg \min_u \frac{\lambda}{2} \|Ku - d\|_2^2 + \mathrm{ICTGV}^2_{\beta,\gamma}(u) \tag{2.12}
$$

In order to derive the primal-dual algorithm, (2.12) is transformed to

$$
\begin{aligned}
u^* = & \min_{u,v} \frac{\lambda}{2} \|Ku - d\|_2^2 + \mathrm{ICTGV}^2_{\beta,\gamma}(u) \\
= & \min_{u,v,w_1,w_2} \frac{\lambda}{2} \|Ku - d\|_2^2 \\
& + \gamma_1 \alpha_1 \|\nabla(u - v) - w_1\|_1 + \gamma_1 \alpha_0 \|\mathcal{E} w_1\|_1 \\
& + \gamma_2 \alpha_1 \|\nabla v - w_2\|_1 + \gamma_2 \alpha_0 \|\mathcal{E} w_2\|_1
\end{aligned}
\tag{2.13}
$$

Algorithm 3 Primal dual algorithm to solve the dMRI-TGV^2 regularization problem

Initialize:

$(u^0, \bar{u}^0, v^0, \bar{v}^0, p^0, q^0, r^0) \qquad \sigma, \tau \geq 0$

while Stopping criteria are not met **do**:

Dual Update:

$p^{n+1} \leftarrow proj_{\sigma F^*}(p^n + \sigma(\nabla \bar{u}^n - \bar{v}^n))$

$q^{n+1} \leftarrow proj_{\sigma F^*}(q^n + \sigma(\mathcal{E}\bar{v}^n))$

$r^{n+1} \leftarrow prox_{\sigma G^*}(r^n + \sigma(K\bar{u}^n - d))$

Primal Update:

$u^{n+1} \leftarrow u^n - \tau(-div(p^{n+1}) + K^* r^{n+1})$

$v^{n+1} \leftarrow v^n - \tau(-div_{\mathcal{E}}(q^{n+1}))$

Extragradient step:

$\bar{u}^{n+1} \leftarrow 2u^{n+1} - u^n$

$\bar{v}^{n+1} \leftarrow 2v^{n+1} - v^n$

and by additionally dualizing the data term the full form results in:

$$\begin{aligned}\min_{u,v,w_1,w_2} \max_{p_1,q_1,p_2,q_2,r} &\langle \nabla (u - v) - w_1, p_1 \rangle + \langle \mathcal{E}\, w_1, q_1 \rangle - \frac{1}{\gamma_1\, \alpha_1}\delta_P(p_1) - \frac{1}{\gamma_1\, \alpha_0}\delta_Q(q_1) \\ &+ \langle \nabla\, v - w_2, p_2 \rangle + \langle \mathcal{E}\, w_2, q_2 \rangle - \frac{1}{\gamma_2\, \alpha_1}\delta_P(p_2) - \frac{1}{\gamma_2\, \alpha_0}\delta_Q(q_2) \\ &+ \langle Ku - d, r \rangle - \frac{1}{2\lambda}\|r\|_2^2\end{aligned} \tag{2.14}$$

Algorithm

The algorithm to solve the dMRI-$ICTGV$ regularization problem is outlined in Algorithm 4.

Algorithm 4 Primal dual algorithm to solve the dMRI-$ICTGV$ regularization problem

Initialize:

while Stopping criteria are not met **do**:

Dual Update:

$p_1^{n+1} \leftarrow proj_{\sigma\gamma_1\alpha_1 F^*}(p_1^n + \sigma(\nabla(\bar{u}^n - \bar{v}^n) - \bar{w_1}^n))$

$q_1^{n+1} \leftarrow proj_{\sigma\gamma_1\alpha_0 F^*}(q_1^n + \sigma(\mathcal{E}\,\bar{w_1}^n))$

$p_2^{n+1} \leftarrow proj_{\sigma\gamma_2\alpha_1 F^*}(p_2^n + \sigma(\nabla\,\bar{v}^n - \bar{w_2}^n))$

$q_2^{n+1} \leftarrow proj_{\sigma\gamma_2\alpha_0 F^*}(q_2^n + \sigma(\mathcal{E}\,\bar{w_2}^n))$

$r^{n+1} \leftarrow prox_{\sigma G^*}(r^n + \sigma(K\bar{u}^n - d))$

Primal Update:

$u^{n+1} \leftarrow u^n - \tau(-div(p_1^{n+1}) + K^*\,r^{n+1})$

$v^{n+1} \leftarrow v^n - \tau(-div(p_1^{n+1}) - div(p_2^{n+1}))$

$w_1^{n+1} \leftarrow w_1^n - \tau(-p_1^{n+1} - div_{\mathcal{E}}\,(q_1^{n+1}))$

$w_2^{n+1} \leftarrow w_2^n - \tau(-p_2^{n+1} - div_{\mathcal{E}}\,(q_2^{n+1}))$

Extragradient step:

$\bar{u}^{n+1} \leftarrow 2u^{n+1} - u^n$

$\bar{v}^{n+1} \leftarrow 2v^{n+1} - v^n$

$\bar{w_1}^{n+1} \leftarrow 2w_1^{n+1} - w_1^n$

$\bar{w_2}^{n+1} \leftarrow 2w_2^{n+1} - w_2^n$

2.2 Reconstruction parameters

2.2.1 Regularization parameters

The basic regularization parameter λ in the above presented optimization problems (2.1), (2.9) and (2.12) controls the balancing between the fit to the measured data and noise regularization. Following [20], a strict separation between regularization and model parameters is applied in this work.

While the model parameters can be tuned to the expected image structure, the regularization parameter is automatically adapted to the expected noise level. The work in [20] shows that a robust method can be defined to linearly depend the regularization value on the amount of subsampling or reduction factor R in the acquired data set as:

$$\lambda(R) = kR + d \tag{2.15}$$

using the parameters k and d, which basically depend on the image structure and have been evaluated by numerical experiments in [17]. See Table 2.1 for explicit values.

2.2.2 Model parameters

The model parameters introduced by the TV regularization functionals are aimed to be tuned for expected image structures, e.g. for cardiac CINE or perfusion imaging respectively.

Time-Space Weighting in TV/TGV^2 functionals

As already mentioned in (1.4) and (1.5) the derivative operators introduce an extra parameter β for time-space weighting. In the case of $ICTGV$ regularization, each TGV term uses its specific time-space weights β_1 and β_2. These weights are automatically normalized in a manner that the elliptic integration over all possible gradient directions equals 1. Given a time-space ratio $\beta = \frac{ds}{dt}$ the values for ds and dt are computed by defining the integral:

$$\frac{1}{2\pi}\int_0^{\pi}\int_0^{2\pi}\sqrt{(ds\,sin\theta\,cos\phi)^2+(ds\,sin\theta\,sin\phi)^2+(dt\,cos\theta)^2}\delta_\phi\delta_\theta = 1 \tag{2.16}$$

The solution is implemented numerically following the methods in [21], Sec 17.6.

Inherent TGV parameters

Independently from the parameters above and following [12] the parameters α_0 and α_1, used for the application of the TGV^2 norm, are defining the weighting between first and second order derivative and are set to the constant ratio of $\frac{1}{\sqrt{2}}$, thus $\alpha_0 = \sqrt{2}$ and $\alpha_1 = 1$.

The definition of the $ICTGV$ norm in (1.7) introduces a further trade-off between the two TGV^2 functionals with the parameters γ_1 and γ_2, which are defined by normalized convex combination:

$$\begin{aligned}\gamma_1(\alpha) &= \frac{\alpha}{\min(\alpha, 1-\alpha)}\\ \gamma_2(\alpha) &= \frac{1-\alpha}{\min(\alpha, 1-\alpha)}\end{aligned} \tag{2.17}$$

using $\alpha \in (0,1)$ as trade-off parameter between the two functionals. See Table 2.1 for explicit parameter values.

2.2.3 Reconstruction parameters

The application of the primal-dual algorithms, as shown in Section 2.1, Page 13, requires the definition of two step-size parameters σ and τ, used in the dual ascend and primal descent steps respectively. In order to speed-up the convergence of the algorithm the step sizes are adapted during the iterative reconstruction. Following [17] the step sizes are updated like:

$$S(\sigma\tau, \eta) = \begin{cases}\eta & \theta\sigma\tau \geq \eta\\ \sqrt{\theta\sigma\tau} & \sigma\tau \geq \eta \geq \theta\sigma\tau\\ \sqrt{\sigma\tau} & \text{else}\end{cases} \tag{2.18}$$

The parameter η is computed per iteration by utilizing the differences in the primal variables after computing the primal descent step. The computed differences are denoted as:

$$\xi = \begin{pmatrix} du \\ dv \\ dw_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} u^{n+1} - u^n \\ v^{n+1} - v^n \\ w_1^{n+1} - w_1^n \\ w_2^{n+1} - w_2^n \end{pmatrix} \tag{2.19}$$

Using ξ the update parameter is then computed as:

$$\eta = \frac{\|\xi\|_2}{\|H(\xi)\|_2} \tag{2.20}$$

with $H(\xi)$ defined as:

$$\begin{aligned} H(\xi) =& (\nabla (du - dv) - dw_1) + (\mathcal{E}\, dw_1) \\ &+ (\nabla\, dv - dw_2) + (\mathcal{E}\, dw_2) + \mathcal{K} du \end{aligned} \tag{2.21}$$

Non-Uniform Sampling

In case of non-uniform acquisition the parameters for the utilized NUFFT toolbox [22] in case of CPU and gpuNUFFT [23] in case of GPU reconstruction respectively, are shown in Table 2.2, with OS denoting the oversampling ratio, KW the interpolation kernel width in grid units and SW an GPU implementation specific parameter, which defines the size of parallel computed sectors. See [23] for more details.

2.2.4 Stopping criteria

Throughout the experiments, 500 iterations turned out to yield sufficient convergence according to the primal-dual gap Section 1.3.4, Page 10. Thus, the primary stopping criterion is selected to be the total number of iterations.

Table 2.1: Regularization and model parameters overview and selected values

Symbol	Denomination	Chosen value	
Model parameters ICTGV		Func.	Perf.
α_0/α_1	Ratio of TGV weights	$1/\sqrt{2}$	$1/\sqrt{2}$
$ds/dt \rightarrow \beta_1$	Spatio-temporal weight 1st comp.	4	4
$ds/dt \rightarrow \beta_2$	Spatio-temporal weight 2nd comp.	0.5	0.5
$\alpha \rightarrow \gamma_{1,2}(\alpha)$	Weighting between 1st and 2nd comp.	0.5	0.6423
Model parameters TGV		Func.	Perf.
α_0/α_1	Ratio of TGV weights	$1/\sqrt{2}$	$1/\sqrt{2}$
$ds/dt \rightarrow \beta$	Spatio-temporal weight	6.5	6.5
Model parameters TV		Func.	Perf.
$ds/dt \rightarrow \beta$	Spatio-temporal weight	6.5	6.5
Regularization parameters			
k	Slope	0.34	0.08
d	Intercept	4.57	1.56

Table 2.2: Parameter values used at application of the (gpu)NUFFT to compute the forward and backward non-Cartesian MR operation.

Parameter	Value
OS	2.0
KW	3.0
SW	8

2.3 Coil Sensitivity Estimation

2.3.1 Strategy

In case no coil sensitivities are present, an iterative coil construction algorithm as described in [20] is implemented in this work. The basic assumptions are that the coil sensitivities are constant over the time series and that a nearly full k-space coverage is acquired considering the accumulation of the whole time series into one dense k-space.

2.3.2 Iterative Reconstruction

The basic steps used to construct the coil sensitivities are:

- Time-averaged, coil-wise reconstruction (v, u_0)
- Generation of absolute values of sensitivities by $H1$-regularization
- Iterative reconstruction of coil sensitivities and phase image generation
- Post processing

In order to compute the time-averaged reconstruction, the acquisitions over all time frames are accumulated and weighted by the sum of the k-space trajectories of all frames:

$$\begin{aligned} \mathrm{v}_k = \arg\min_{\bar{v}} \sum_{t=1}^{T} \frac{1}{2} \| d_{k,t} - M_t \mathcal{F}(\bar{v}) \|_2^2 \\ \Rightarrow \bar{v}_k = \mathcal{F}\left(\frac{1}{\bar{M}} \sum_{t=1}^{T} d_{k,t} \right) \end{aligned} \tag{2.22}$$

with $\bar{M} = \sum_{t=1}^{T} M_t$. The phase corrected sum-of-squares combination of all coil images of v is defined as u_0:

$$u_0 = \sqrt{\sum_{k=1}^{C} |v_k|} \cdot \exp\left(i \sum_{k=1}^{C} \angle v_k \right) \tag{2.23}$$

The per coil absolute value of the sensitivity is computed by $H1$-regularization of the coil-reconstruction image:

$$\begin{aligned} |b_k| &= \arg\min_b \frac{\mu}{2}\|bu_0 - |v_k|\|_2^2 + \|\nabla b\|_2^2 \\ \Rightarrow (\mu u_0 + \Delta)|b_k| &= (\mu u_0 - |v_k|) \\ |b_k| &= (\mu u_0 + \Delta)^{-1}(\mu u_0 - |v_k|) \end{aligned} \tag{2.24}$$

Additionally, the absolute values $|b_k|$ of each coil are normalized by the sum-of-squares combination of all coils:

$$b_k = \frac{|b_k|}{\sum_{k=1}^{C} |b_k|^2} \tag{2.25}$$

The phase of the sensitivities is computed in an iterative manner. First, the coil reconstruction v_k with the largest $L2$-norm value is chosen and its phase b_{k0} is initialized with a zero phase. Additionally, the image phase u_{k0} is computed by TGV-regularization:

$$\begin{aligned} b_{k0} &= |b_{k0}| \\ u_{k0} &= \arg\min_u \nu\,\mathrm{TGV}(u) + \|b_{k0}u - v_{k0}\|_2^2 \end{aligned} \tag{2.26}$$

Repeatedly, the coil with the maximum overlap with the already computed coil reconstructions is chosen, in order to compute its corresponding sensitivity and to update the phase image u:

$$\begin{aligned} &\text{Repeat for } k = 2 \cdots C \\ &\qquad w = \sum_{l=1}^{k} |u_{jl}| \\ &\qquad j_k = \max_{j_k} |b_{j_k} \cdot v_{j_{k-1}}| \qquad \text{(find coil with maximum overlap)} \\ &\qquad \angle b_{j_k} = \arg\min_{\bar{b}} \frac{\mu}{2}\left\|\nabla \bar{b}\right\|_2^2 + \frac{1}{2}\left\|w(\bar{b}|b_{j_{k-1}}|u_{j_{k-1}} - v_{j_k})\right\|_2^2 \\ &\qquad b_{j_k} = |b_{j_k}| \cdot \exp i\angle b_{j_k} \\ &\qquad u_{j_k} = \arg\min_u \nu\,\mathrm{TGV}(u) + \sum_{i=0}^{k-1}\left\|b_{j_i}u - v_{0_k}\right\|_2^2 \end{aligned} \tag{2.27}$$

Post process sensitivities by:

$$(b_j)_j = \arg\min_{(b_j)_j} \frac{\mu^*}{2}\sum_j \left(\left\|\nabla b_j\right\|_2^2 + \frac{1}{2}\left\|ub_j - v_j\right\|_2^2\right) \tag{2.28}$$

2.3.3 Algorithms

This section explains the used algorithms in order to solve the coil construction problems described above. Subsequently, these are a CG solver to compute the inverse of the problem stated in (2.24), a primal-dual solver for TGV^2 denoising and a $L2 - H1$ optimizer as required in (2.26)-(2.28).

CG Solver

The problem described in (2.24) is considered as linear system of equations in the form of $A \cdot x = b$ and is solved by application of the common iterative Conjugate Gradient (CG) method [24]. This method is already implemented in the AGILE library [25], while the linear operator A has to be defined as:

$$A(x) = \mu\, u_0\, x + div(\nabla\, x) \tag{2.29}$$

See the Appendix for an outline of the CG algorithm.

PD L2-H1 subproblem

The quadratic regularization of the gradient function, also known as $L2 - H1$ problem, again is solved by application of the primal-dual solver by [18]. The outline of the algorithm is shown in Algorithm 5, while the required proximal operator for F^* is defined equivalently to (2.8) and G^* is derived as:

$$\begin{aligned} prox_{\tau\, G^*}\,(\tilde{p}) &= (I + \tau\delta\, G^*)^{-1}(\tilde{p}) \\ &= \min_p \frac{\|p - \tilde{p}\|_2^2}{2} + \frac{\tau}{2}\|w(p|b_{j_k}|u_{j_{k-1}} - v_{j_k})\|_2^2 \\ \Rightarrow p &= \frac{\tilde{p} + \tau\, v_{j_k} b_{j_k} u_{j_{k-1}} w^2}{1 + \tau\,(w b_{j_k} u_{j_{k-1}})^2} \end{aligned} \tag{2.30}$$

Algorithm 5 Primal dual algorithm to solve the $L1 - H2$ regularization problem

Initialize:
$u^0, \bar{u}^0, \sigma, \tau$

while Stopping criteria are not met **do**:
Dual Update:
$p^{n+1} \leftarrow prox_{\sigma\mu F^*}(p^n + \sigma(\nabla \bar{u}^n))$

Primal Update:
$u^{n+1} \leftarrow prox_{\tau G^*}(u^n - \tau(-div(p^{n+1})))$

Extragradient step:
$\bar{u}^{n+1} \leftarrow 2u^{n+1} - u^n$

PD TGV2 denoising subproblem

Equations (2.26) and (2.27) show another application of the TGV norm as a regularization function. In this case the TGV norm is applied in a denoising sense as has been shown in [12]. The algorithm to solve the problem is presented in Algorithm 6.

The proximal mapping G^* is defined as:

$$\begin{aligned} prox_{\tau G^*}(\tilde{u}) &= (I + \tau\delta G^*)^{-1}(\tilde{u}) \\ &= \min_u \frac{\|u - \tilde{u}\|_2^2}{2} + \frac{\tau}{2} \sum_{i=0}^{k-1} \|b_{j_i} u - v_{0_k}\|_2^2 \\ \Rightarrow u &= \frac{\tilde{u} + \tau \sum_{i=0}^{k-1} v_{j_i} b_{j_i}}{1 + \tau \sum_{i=0}^{k-1} b_{j_i} b_{j_i}} \end{aligned} \tag{2.31}$$

2.3.4 Parameters

Explicit parameter values used in the algorithms explained above are depicted in Table 2.3.

Algorithm 6 Primal dual algorithm to solve the TGV^2 denoising problem

Initialize:
$u^0, \bar{u}^0, \sigma, \tau$

while Stopping criteria are not met **do**:
Dual Update:
$p^{n+1} \leftarrow proj_{\sigma\nu\alpha_1 F^*}(p^n + \sigma(\nabla \bar{u}^n - \bar{v}^n))$
$q^{n+1} \leftarrow proj_{\sigma\nu\alpha_0 F^*}(q^n + \sigma(\mathcal{E}\bar{v}^n))$

Primal Update:
$u^{n+1} \leftarrow prox_{\tau G^*}(u^n - \tau(-div(p^{n+1})))$
$v^{n+1} \leftarrow v^n - \tau(-div_{\mathcal{E}}(q^{n+1}))$

Extragradient step:
$\bar{u}^{n+1} \leftarrow 2u^{n+1} - u^n$
$\bar{v}^{n+1} \leftarrow 2v^{n+1} - v^n$

Table 2.3: Parameters used in algorithms of the coil construction process. Primal dual L1-regularization parameters in the first row and primal dual TGV^2 denoising parameters in the second row. μ^* and maxIt* denote the values used in the coil post-processing step.

Method	Parameter value						
	μ (μ^*)	ν	τ	σ	α_0	α_1	maxIt (maxIt*)
PD-L2-H1	2.0(0.1)	-	$\frac{1}{\sqrt{8}}$	$\frac{1}{\sqrt{8}}$	$\sqrt{2}$	1	500(1000)
PD-TGV	-	0.2	$\frac{1}{\sqrt{12}}$	$\frac{1}{\sqrt{12}}$	$\sqrt{2}$	1	100

2.4 CUDA Implementation

This section presents the basic parallelization approach of the CUDA accelerated C++ implementation built in this work. Good introductions to the principles of CUDA programming are given in [26]–[28].

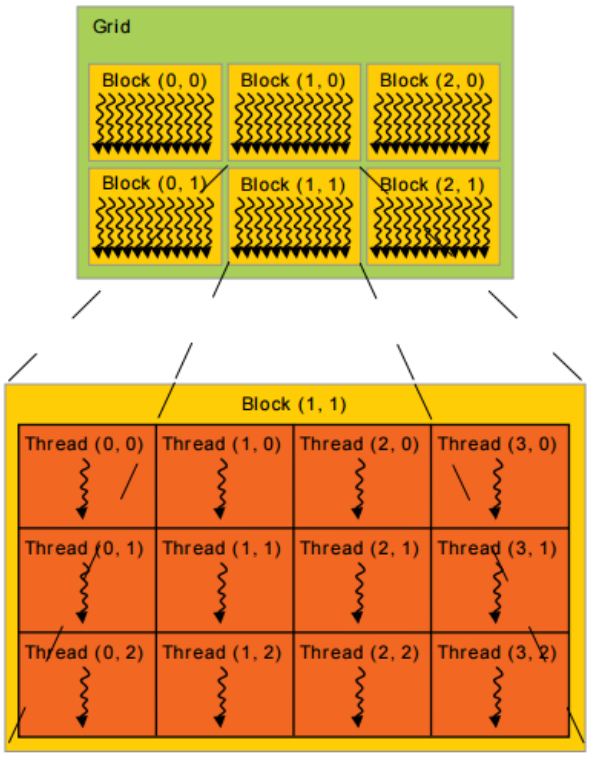
2.4.1 Parallel approach

The mentioned algorithms and operations perfectly fit the requirements for single instruction, multiple data (SIMD) architectures, since a large amount of the necessary vector operations is on a per-element basis. Therefore, the CUDA-based AGILE C++ library [25] is most suitable as the primary used framework to utilize GPUs for processing.

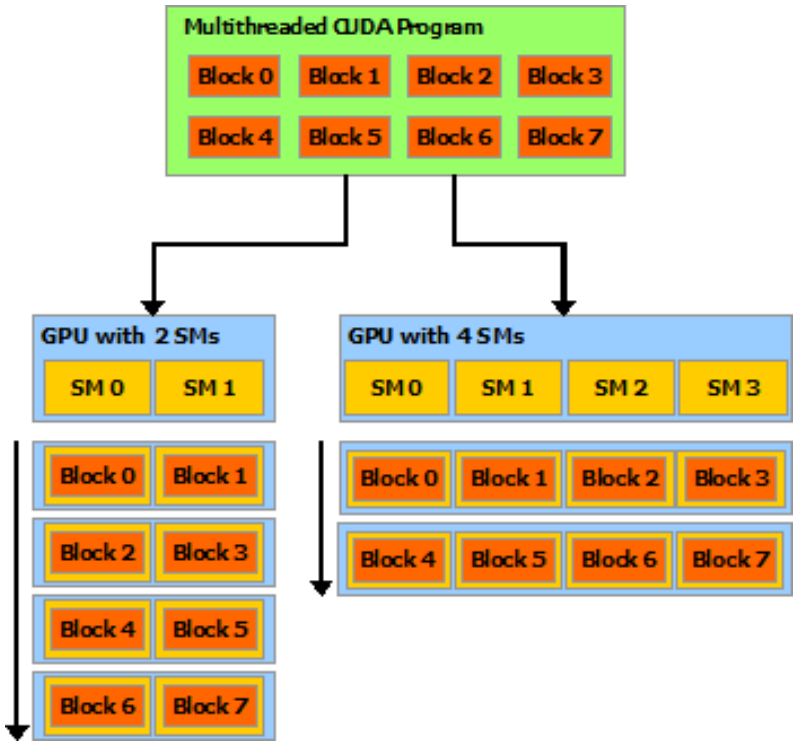
Figure 2.1(a) shows the basic CUDA organization of a kernel call with parallel executed threads and blocks of threads, further arranged in a grid of blocks. It can be seen that every block and every thread is associated with an index. These indices can be exploited to organize contiguous data access and therefore, each thread can be seen to be processing computations only for its corresponding data point (e.g. per pixel). Depending on the hardware architecture and the amount of available on-device stream multiprocessors (SM), thread-blocks are further processed in parallel. The actual arrangement and the number of blocks and threads-per-block are usually defined at compile time and depend on the problem dimensions. Figure 2.1(b) shows the automatic scaling of parallel executed thread blocks depending on the actual number of available SMs. Currently, typical high-end graphics devices contain 16-24 SMs (each consisting of 128 CUDA cores) and 4-6GB of available device memory.

2.4.2 Memory layout

As mentioned above a vast amount of the operations can be computed on a per-element basis and therefore, the straight forward to use memory access pattern is linear access. Figure 2.2 depicts an example of consecutive memory access of 8 threads of 2 thread-blocks on linear memory. Since the amount of data is typically larger than the total amount of threads, all memory locations are processed by shifting the thread-blocks in an interleaved manner.



(a)



(b)

Figure 2.1: (a) Organization of parallel executed thread-blocks into an execution grid. Each thread-block runs its threads in parallel and each thread can be seen to be independently processing computations for its corresponding data point. (b) Hardware-specific scale of parallel executed thread-blocks depending on the actual amount of stream-multiprocessors (SM) on the device. Images taken from [26]

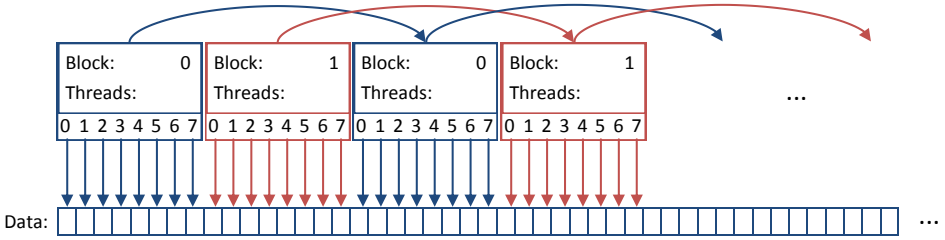


Figure 2.2: Thread-wise parallel access on linear memory array locations. Every thread of the two thread-blocks processes its corresponding data element. All memory locations are processed by linearly shifting the thread-blocks in an interleaved manner.

Basically, depending on the computational need read-access is possible to any arbitrary global memory location, but whenever multiple threads require simultaneous write access to a memory location, more sophisticated thread and memory strategies, such as atomic write operations and shared memory, have to be applied. The basic components of the AGILE library are low-level element-wise linear vector operations, like vector addition, scaling, etc., hence see the above mentioned literature for more details on complex patterns.

For example, the memory layout for the image sequence is defined as:

$$\begin{pmatrix} u_{0,0,t=0} \\ u_{0,1,t=0} \\ \cdots \\ u_{N-1,M-1,t=0} \\ u_{0,0,t=0} \\ \cdots \\ u_{N-1,M-1,t=T} \end{pmatrix}^T \tag{2.32}$$

and for the $b1$ data as:

$$\begin{pmatrix} b1_{0,0,chn=0,t=0} \\ b1_{0,1,chn=0,t=0} \\ \cdots \\ b1_{N-1,M-1,chn=0,t=0} \\ b1_{0,0,chn=1,t=0} \\ \cdots \\ b1_{N-1,M-1,chn=C,t=T} \end{pmatrix}^T \tag{2.33}$$

In order to access an element (x, y) of coil c at a certain time point t the memory location index is computed as:

$$\text{ind}(x, y, c, t) = x + y \cdot (M) + c \cdot (N \cdot M) + t \cdot (N \cdot M \cdot C) \tag{2.34}$$

2.4.3 Numerical computations

Apart from the implementation of the MR forward and backward operators, which is described in the next section, a central part, in all different kinds of reconstructions, is the computation of the gradient and divergence operators. Numerically, the gradient operations ($3D$ and symmetrical 2^{nd} derivative) are computed as forward differences and the divergence operations as backward differences respectively. These functions have been added to the 'low-level' package of the AGILE library. The latest version of the AGILE library has been designed basically for $2D$ applications, so extensions to support large $3D$ data-sets, in the sense of time series data, have been developed and integrated. These extensions basically comprise:

- Refactoring to support latest CUDA architecture
- Finite forward and backward differences for $3D$ data vectors
- Forward and backward FFT operations for vector based data sets
- Extended vector utilities ($l1/l2$-norm, min-max element computation, element-wise compare)
- Extended thread assignment for large data sets

2.4.4 MR Operator

In case of Cartesian acquisition a masked FFT operator supporting time-series data has been added to the AGILE library. In its back end, the well-known CUFFT [29] library is utilized.

In case of non-Cartesian reconstruction the capabilities of the gpuNUFFT [23] library are used. Since the gpuNUFFT library initially has been created for large $3D$ data sets, the performance of the $2D$ Gridding and inverse Gridding processes has been optimized by computing multiple coils at once per Gridding operation. The amount of simultaneously computed coils is derived during runtime by querying the amount of available global device memory. The gpuNUFFT's C++ API has been extended

to support input data that is already residing in GPU memory, in order to allow an optimal integration with data managed by the AGILE library.

2.4.5 Memory Estimation / Hardware Limits

One major drawback of the implementation of primal dual algorithms is the huge memory consumption in order to keep the state of all vectors. The overall GPU memory consumption can be a limiting factor for large problems. Table 2.4 contains rough memory estimates for problems of the dimension (M, N, C, T).

Table 2.4: Amount of data variables in problem dimensions and required memory estimation for the different regularization functions. The first column describes the type and dimensions of necessary distinct data vector variables. For the purpose of demonstration, the required memory for a Cartesian data set with the dimensions (N=nEnc,M=nRO,C,T): $256, 256, 32, 30$ and for a non-Cartesian (N,M,C,T,nRO,nEnc): $256, 256, 32, 30, 384, 192$ is shown.

	Cartesian			Non-Cartesian		
# Vectors (Dimension)	*TV*	*TGV*	*ICTGV*	*TV*	*TGV*	*ICTGV*
# image $(N \cdot M \cdot T)$	12	36	58	12	36	58
# kspace $(nRO \cdot nEnc \cdot C \cdot T)$	3	3	3	3	3	3
# b1 $(N \cdot M \cdot C)$	1	1	1	1	1	1
# complex values	214E+6	262E+6	305E+6	238E+6	285E+6	328E+6
Required memory (MB)	1.636	1.996	2.326	1.816	2.176	2.506

2.5 Software Integration

2.5.1 Data Formats

The current software version created in this work supports two different input and three output formats. Input data can either exist in the native binary format of the

AGILE library (extension used in this context is ".bin") or the multivendor-capable common raw data format called *ISMRMRD* (".h5", [30]). Reconstruction results can be stored either by using the two formats mentioned above or by using the de-facto medical imaging standard *DICOM* [31]. The use of *DICOM* embedded measurement data as input is currently not supported by the software.

AGILE Binary

The AGILE library supports an own implementation of a binary data type, in order to allow communication with other software, e.g. *MATLAB*. The library supports both import and export routines for the use within MATLAB. See the Appendix for further details and demos how to perform the data exchange.

ISMRMRD (HDF)

The preferred format for raw input data is the new *ISMRMRD* format developed by a committee of the *ISMRM* community. Its main aspect is the introduction of a non-proprietary file standard containing all necessary meta-information and data, in order to be able to reconstruct images from any MRI experiment [30]. Various tools exist [1], which automatically convert vendor-specific raw measurement data to the independent *ISMRMRD* format.

The file format is based on the well-known Hierarchical Data Format (*HDF*5 [32]), which is intrinsically also used by *MATLAB*. Thus, ".h5" files can directly be opened and managed within the *MATLAB* environment. See the Appendix for an example of how to load data from ".h5" files.

DICOM

Apart from the two mentioned formats for input and output data, reconstruction results can also be stored in the industry standard *DICOM* format. In order to generate these images the *DICOM* Toolkit [33] is utilized.

[1] https://github.com/ismrmrd

Data Flow

Figure 2.3 sketches the basic data flow implemented in this work. It shows that the reconstruction can either be started without the use of the coil construction part, if $b1$ and $u0$ data is already available, or with a preliminary data normalization and coil construction step based on raw scanner data, e.g. raw measurement data converted to ISMRMRD.

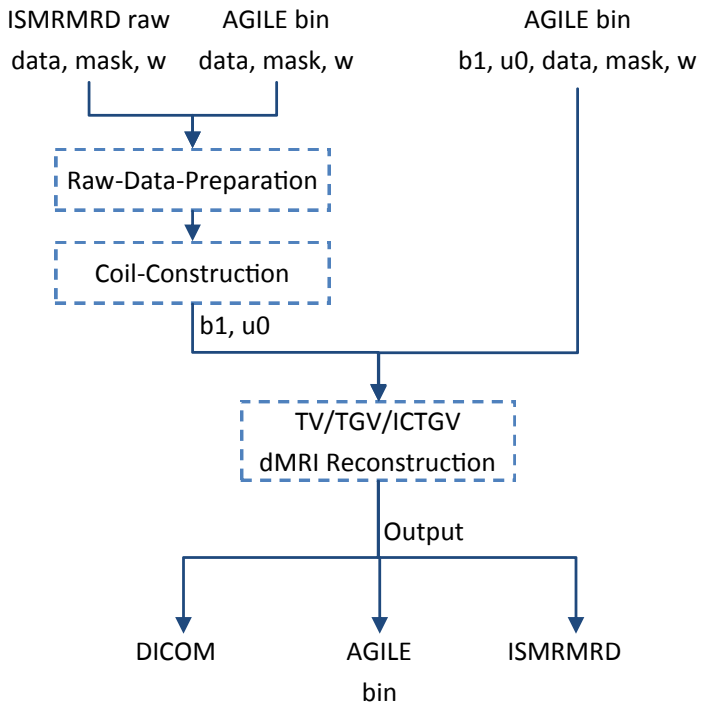


Figure 2.3: Data flow showing different data processing paths depending on available input data.

2.5.2 Raw Data Processing

The data preparation step considers the different acceleration methods implemented and manifested in the raw measurement data, and prepares and reshapes the data arrays, in order to be used in the image reconstruction part. This section gives an overview of the required meta information and schematically describes the core features of the preparation step.

Meta Information

In order to successfully perform the raw data preparation and coil construction steps, some fundamental information has to be provided either in form of command line arguments or encoded in the raw data meta information. Basically, the necessary content consists of:

- Encoding space dimensions (readouts, encodings, coils, frames)
- Reconstruction space dimensions (width, height)
- Type of trajectory (Cartesian, radial or arbitrary)
- Total number of acquisitions
- Acquisition details:
  - Number of samples
  - Line number or k-space location of every sample in case of non-uniform acquisition
  - Center line
  - Center column
  - Oversampling

Apart from the basic problem dimensions, details about the acquired number of samples per line and the defined center column give information about partial Fourier and asymmetric echo acquisitions. Potential oversampled or rectangular FOV acquisitions are further derived from the ratio between the reconstruction and encoding space dimensions.

Data Completion

The partial Fourier acquired data is currently completed by the most basic compensation method, namely zero filling. Although this consequently leads to artifacts in case of large skipped k-space portions, it is the method of choice for the first version, but primary offers a defined interface to employ more sophisticated methods, as described in Section 1.2.2, Page 5.

Zero-filling is also implemented to complete asymmetrically acquired echo data. Nevertheless, artifacts do not affect the results with large extent, since most vendors implicitly implement a twofold readout oversampling in frequency encoding direction. Furthermore, the readout oversampling is automatically removed for each line by

inverse FFT, data cropping (halving) in image domain and again followed by forward FFT.

Automatic Trajectory Generation

In case no trajectory information is provided in the raw measurement data files, the k-space sampling mask in case of Cartesian sampling or a basic radial trajectory can be generated automatically, if the meta information conveys at least the type of trajectory.

Normalization

For best reconstruction results and in order to maintain an expected value scale, the input data is automatically normalized to the range of approximately $[0, 255]$. In the Cartesian case, this is done performing a time-averaged reconstruction, i.e. the k-space summation of all time frames into a dense k-space and the sum-of-squares multi-coil reconstruction of it, followed by the evaluation of the normalization factor.

The normalization factor is defined as the median value of all elements larger than 90% of the maximum absolute value of the time-averaged reconstruction image.

The evaluation of the normalization factor in the Non-Cartesian case is performed slightly different. Initially estimated coil sensitivities are applied to the time-averaged reconstruction before sum-of-squares combination, which leads to a more robust normalization factor.

Fourier shift preparation

A further data preparation step, which can additionally be applied for Cartesian data sets, is the point-wise multiplication of the k-space data with a so-called "Chop" function F_c that eliminates the need for explicit FFT-shift calls during the iterative reconstruction and thus reduces the overall computation time. The function is derived from the Fourier shift theorem by $[\frac{N}{2}, \frac{M}{2}]$ and is defined as:

$$
\begin{aligned}
f\left(x-\frac{N}{2}, y-\frac{M}{2}\right) \quad &\overset{FT}{\longleftrightarrow} \quad F(k,l)e^{-j(2\pi k(\frac{N}{2})/N)-j(2\pi l(\frac{M}{2})/M)} \\
&\longleftrightarrow \quad F(k,l)\underbrace{e^{-j\pi(k+l)}}_{F_c(k,l)} \\
F_c(k,l) &= -1^{k+l}
\end{aligned}
\tag{2.35}
$$

Coil Reconstruction

The coil sensitivity reconstruction and the estimation of the initial solution are implemented as described in Section 2.3, Page 23. The results are automatically exported to the output directory, in order to be reused in upcoming image reconstructions.

Reconstruction and Parameters

After all preprocessing steps are completed, the dMRI TV, $TGV2$ or $ICTGV$ reconstructions can be started. Reconstruction results are automatically exported to the user-defined results directory.

The reconstruction can be parametrized either by command line arguments or by a parameter configuration file. See the Appendix, Page 73 for a README on setup, program options and command line arguments.

2.5.3 Test environment

The project has been developed and measurements of both CPU and GPU performance have been recorded on the following LINUX-based system:

- Intel Core i5-3350P, 3.10 GHz
- 16GB RAM
- openSUSE 12.2, 64bit
- Matlab 2013a
- NVIDIA GeForce GTX 770, 4095MB, Driver Version: 340.46
- CUDA Toolkit 6.5
- GCC 4.7.1

2.6 Performance evaluation

2.6.1 Reconstruction times

The main focus of this work is the implementation of a fast and flexible dMRI reconstruction solution. Hence, the most important evaluation criterion is the overall reconstruction time of the algorithm considering differently scaled and acquired input data sets.

The reconstruction times of the three dMRI regularization functions (TV, $TGV2$, $ICTGV$) were recorded separately for the initialization and the algorithmic reconstruction parts. The times have been evaluated by the use of C++ system timers and by using MATLAB's $tic - toc$ timing functions respectively.

2.6.2 Convergence

In order to be able to analyze the convergence behavior depending on the regularization method and the acceleration factor, the primal-dual gap was evaluated at every 10 iterations during reconstruction.

2.6.3 Reference implementation

The CPU reference solution consists of a MATLAB package, fully capable to reconstruct the mentioned regularization and precomputation steps and is the result of the work presented in [34]. Non-Cartesian Fourier transformations are implemented by using the well-known nonuniform FFT MATLAB toolbox (NUFFT, [22]) and an identical parameter configuration (Table 2.1) has been applied in both the CPU and GPU reconstructions. Both CPU and GPU reconstructions have been performed in single precision floating point format.

In order to give an estimate of the similarity of the GPU and CPU reconstruction the root mean square error (RMSE) of the normalized difference images is computed as:

$$RMSE(x,y) = \sqrt{\frac{\sum_{k=1}^{N} |x_k - y_k|^2}{N}} \tag{2.36}$$

2.7 Test Data Sets

For evaluation purposes of the reconstruction times for the CPU and GPU solutions, the following Cartesian and non-Cartesian data sets were acquired and analyzed with respect to different acceleration factors (R) on a whole-body MRI scanner (3T, Skyra, Siemens Healthcare, Erlangen, Germany). The applied sequences are standard sequences implemented by the manufacturer.

The acquired data sets have been converted with the *ISMRMRD* provided tool *siemens_to_ismrmrd*[2]. Each data set is characterized by the shape of its trajectory, the acceleration factor R, the number of coils (C), time frames (T) and slices (S), the amount of phase encoding lines and readout (RO) samples per frame and the dimensions of the reconstruction matrix size. All acquisitions have been performed automatically with twofold oversampling in the readout direction, which has been removed automatically in the raw data preparation step.

2.7.1 Cardiac CINE

CINE imaging is demonstrated with Cartesian and and non-Cartesian data sets. Both have been acquired with different undersampling factors R. In case of Cartesian acquisition this has been controlled by the TPAT (Siemens) factors $R \in \{1, 2, 4, 6, 8\}$, while $R = 8$ means that only every eighth line has been recorded. Radial data has been undersampled by $R \approx \{1, 2, 4, 8, 18\}$, which corresponds to the number of radial spokes per frame as $\{216, 108, 54, 27, 12\}$. Table 2.5 summarizes the characteristics of the cardiac CINE data sets.

[2] https://github.com/ismrmrd/siemens_to_ismrmrd

Table 2.5: Overview of CINE imaging test data sets, characterized by number of encodings (Enc), readouts (RO), coils (C) and frames (T).

Data Set		Dimensions						
Name	Trajectory	R	Enc/ Frame	RO/ Enc	C	T	Total Acq.	Recon Matrix
$CINE_{cart}$	Cartesian TPAT	1	192	448	26	23	4.416	[224,186]
		2	96	448	26	24	2.304	
		4	48	448	26	24	1.152	
		6	36	448	26	24	864	
		8	24	448	26	24	576	
$CINE_{radial}$	Radial	1	216	448	26	22	4.752	[224,224]
		2	108	448	26	22	2.376	
		4	60	448	26	22	1.320	
		8	24	448	26	22	528	
		18	12	448	26	22	264	

2.7.2 DCE Perfusion

The DCE perfusion data set has been recorded as a fully sampled Cartesian k-space for three simultaneously acquired slices. In order to allow a detailed comparison, artificial TPAT undersampling $R \in \{1, 2, 4, 6, 8\}$ has been applied in an additional data preparation step. Table 2.6 contains the detailed data set descriptions.

Table 2.6: Overview of the DCE Perfusion test data sets, characterized by number of encodings (Enc), readouts (RO), coils (C), frames (T) and slices (S).

Data Set		Dimensions							
Name	Trajectory	R	Enc/ Frame	RO/ Enc	C	T	S	Total Acq.	Recon Matrix
$Perf_{cart}$	Cartesian TPAT	1	128	256	5	70	3	26.880	[128,128]
		2	64	256	5	70	3	13.440	
		4	32	256	5	70	3	6.720	
		6	22	256	5	70	3	4.620	
		8	16	256	5	70	3	3.360	

2.7.3 Raw Data Integration

Two extra data sets have been analyzed, aimed to show the capabilities of the built software to support special vendor-specific accelerated acquisition techniques: Rectangular FOV, Partial Fourier and Asymmetric Echo acquisition. Table 2.7 comprises the acquisition details.

Table 2.7: Overview of the raw data integration test data sets, characterized by number of encodings (Enc), readouts (RO), coils (C) and frames (T).

Data Set		Dimensions						
Name	Trajectory	R	Enc/ Frame	RO/ Enc	C	T	Total Acq.	Recon Matrix
$CINE_{pf}$	Cartesian TPAT, Partial Fourier	8	24	448	26	24	577	[224,186]
$CINE_{pf,ae}$	Cartesian TPAT, Partial Fourier, Asym. Echo	8	24	352	26	25	601	[224,186]

3 Results

3.1 Cardiac CINE

The following section presents the reconstruction results for the Cardiac CINE data sets $CINE_{cart}$ and $CINE_{radial}$.

Figures 3.1 and 3.2 show reconstructed absolute coil sensitivities after the initialization step for selected coils. The absolute values of three selected coils are depicted with respect to different data subsampling factors R.

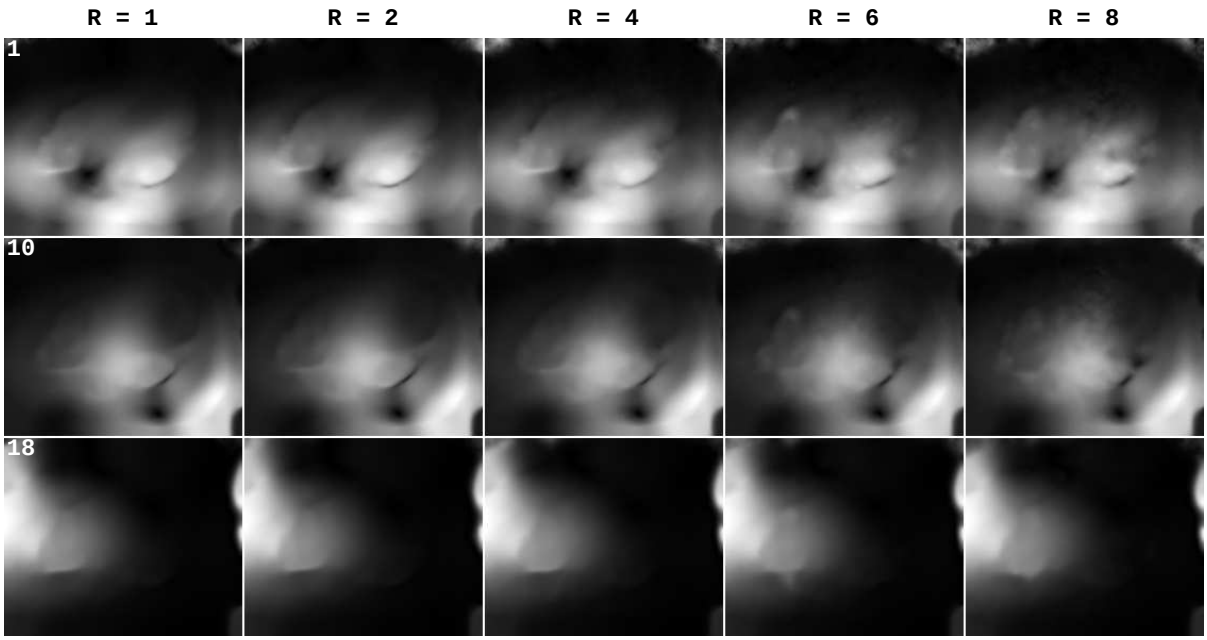


Figure 3.1: Sample absolute coil images (rows) of the $CINE_{cart}$ data set for the coils $1, 10, 18$. The reconstructed absolute coil images are presented for different data subsampling factors R (columns).

The Figures 3.3 -3.6 contain reconstructed frame images of the $CINE_{cart}$ and $CINE_{radial}$ data sets with differently applied regularization methods (left column: TV, center: TGV, right: $ICTGV$). The displayed image region of interest is zoomed as displayed in the

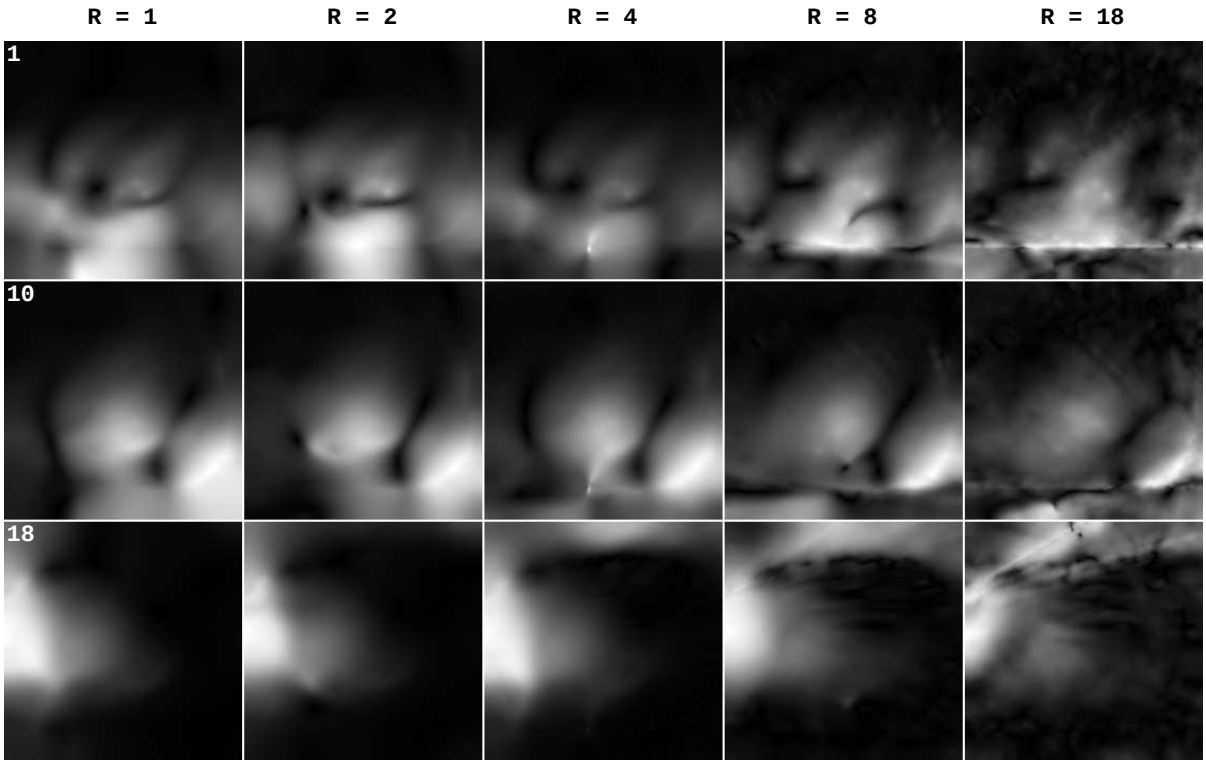


Figure 3.2: Sample absolute coil images (rows) of the $CINE_{radial}$ data set for the coils $1, 10, 18$. The reconstructed absolute coil images are presented for different data subsampling factors R (columns).

presented full reference frame. The basic undersampling artifacts are shown in a simple reference image after direct sum-of-squares (SOS) reconstruction.

The results of the applied GPU reconstruction in contrast to the reference CPU solution are presented as normalized absolute differences in Figures 3.7 and 3.8. Moreover, the root-mean-square-error (RMSE) per frame is depicted as an additional error estimate. The displayed differences are rescaled for better visibility.

3.1.1 Reconstruction Times

The comparisons of the initialization and reconstruction times are given in Figures 3.9 and 3.10. Each figure shows the CPU and GPU initialization times and speedup in the left column and the reconstruction times and reconstruction speedups in the right column. Table 3.1 summarizes the reconstruction times and speedups for the $CINE_{cart}$ data set for all applied undersampling factors R.

3.1.2 PD Gaps

In order to show the convergence of the GPU and CPU algorithms, the progress of the Primal Dual Gap (PD-Gap) during the reconstruction of the $CINE_{cart}$ (left column) and the $CINE_{radial}$ (right colum) data sets with different regularization methods is depicted in Figure 3.11. The plot is drawn logarithmically and as a reference the $O(1/N)$ function is additionally shown.

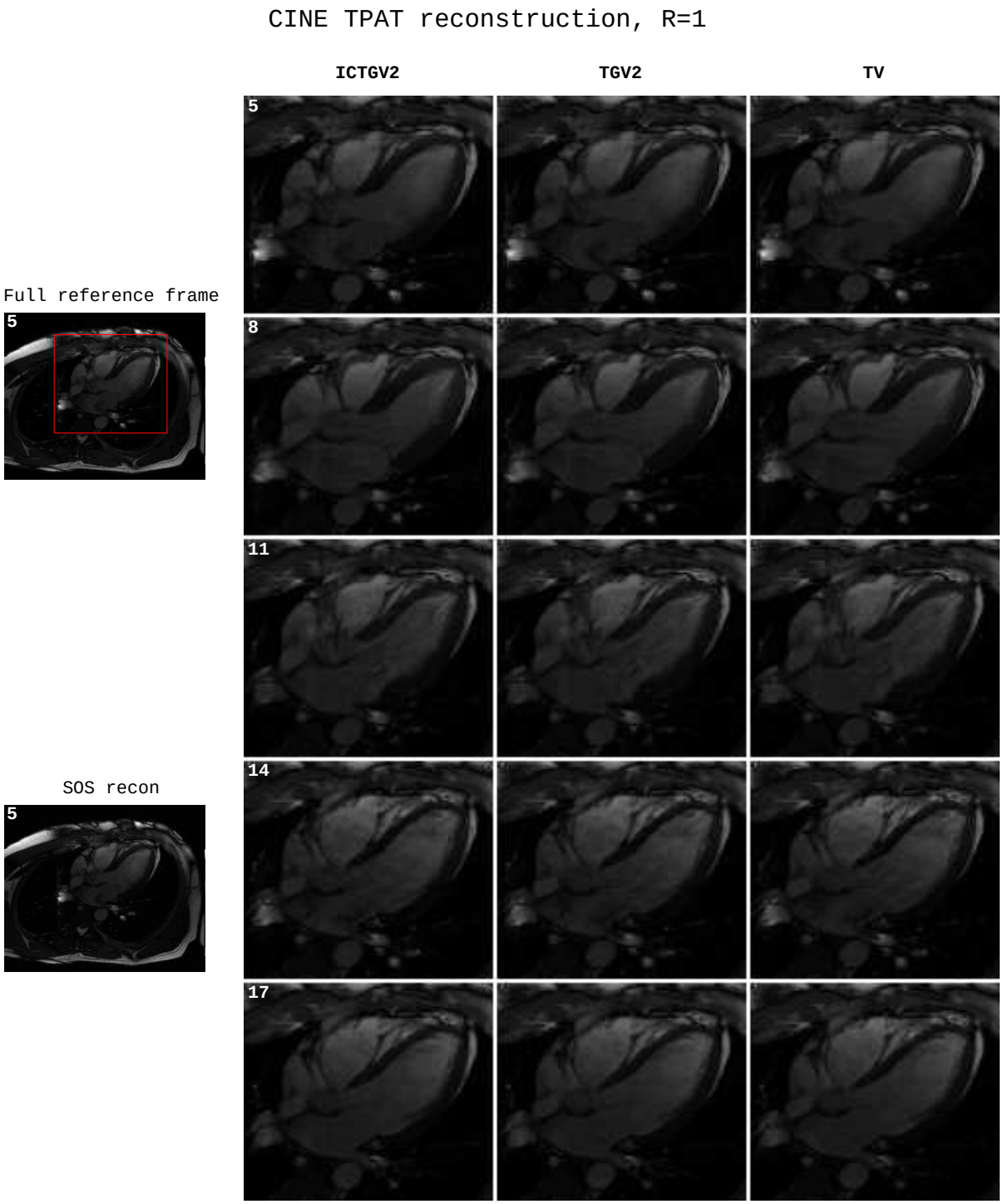


Figure 3.3: Reconstructed frame images (rows) of the $CINE_{cart}$ data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (top left). The basic undersampling artifacts are shown in the SOS reference image (bottom left).

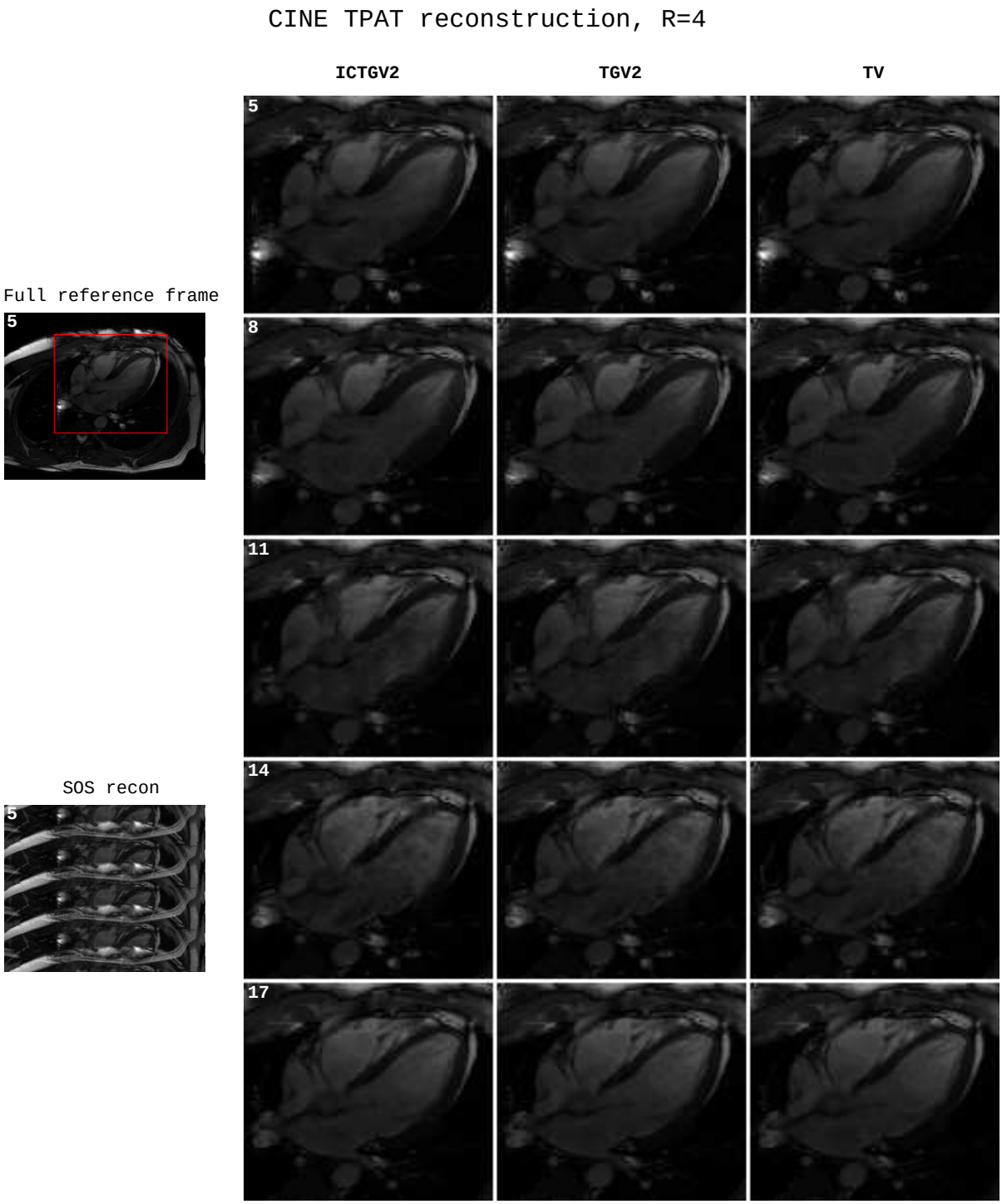


Figure 3.4: Reconstructed frame images (rows) of the $CINE_{cart}$ data ($R = 4$) set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (top left). The basic undersampling artifacts are shown in the SOS reference image (bottom left).

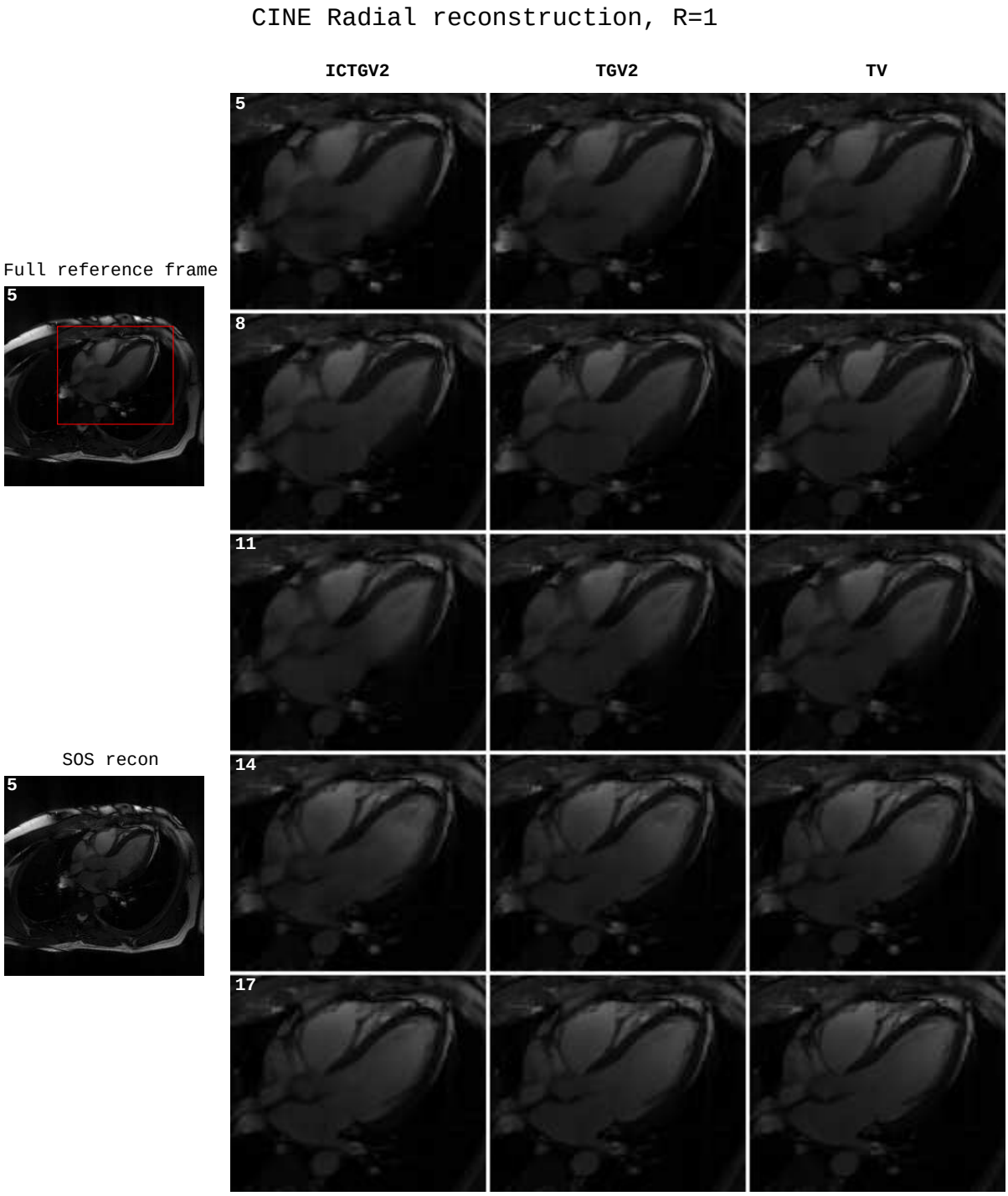


Figure 3.5: Reconstructed frame images (rows) of the $CINE_{radial}$ data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (top left). The basic undersampling artifacts are shown in the SOS reference image (bottom left).

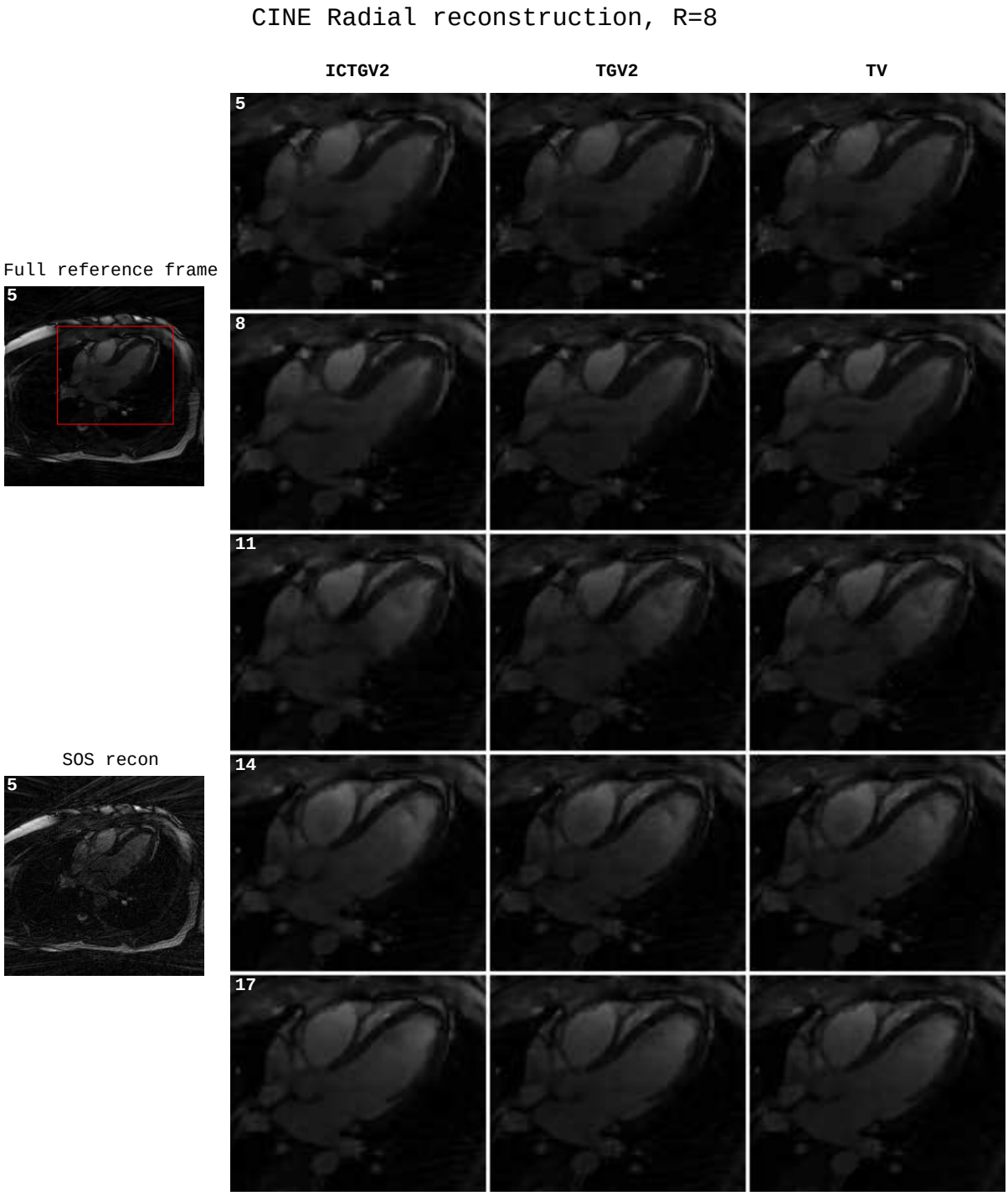


Figure 3.6: Reconstructed frame images (rows) of the $CINE_{radial}$ ($R = 8$) data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (top left). The basic undersampling artifacts are shown in the SOS reference image (bottom left).

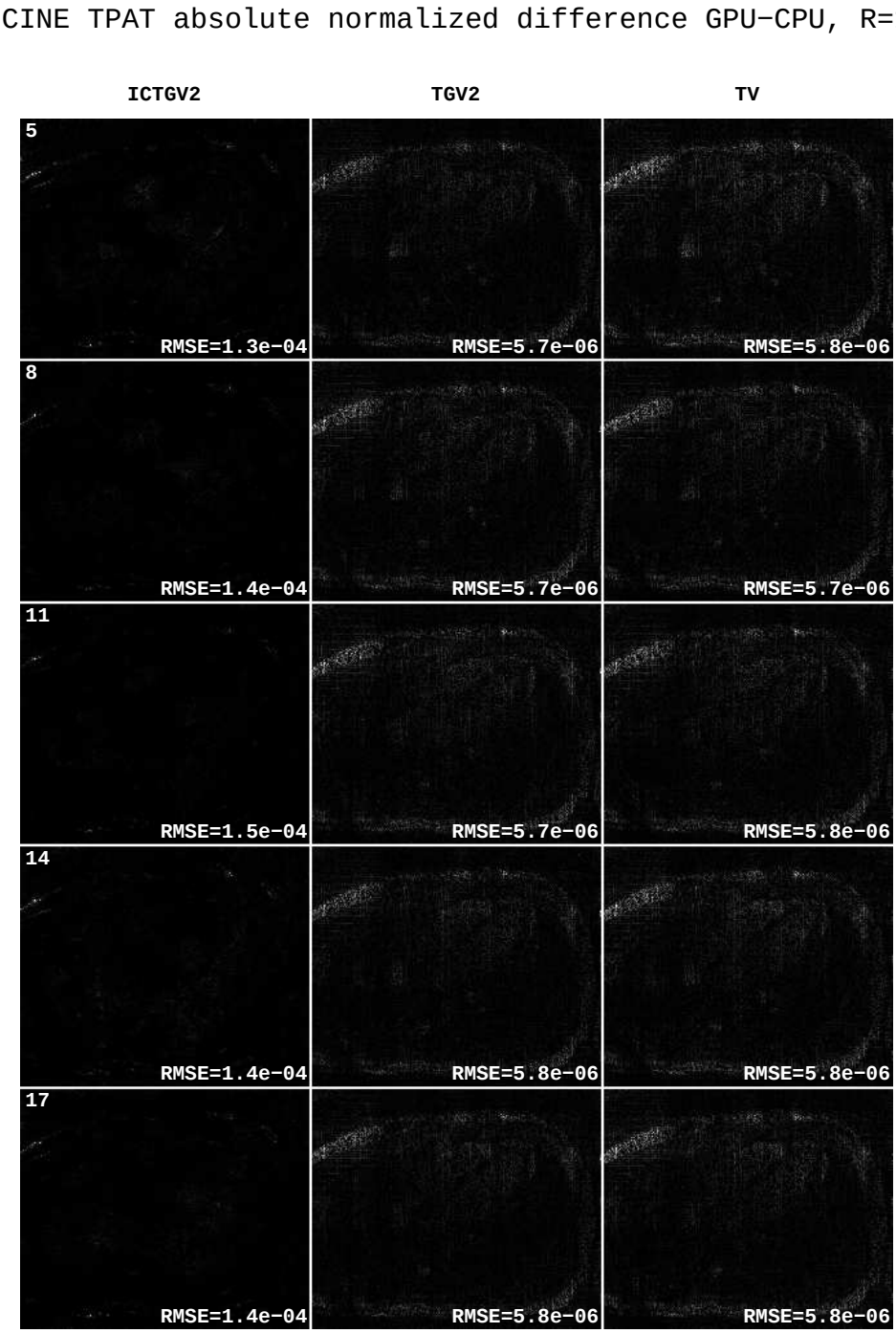


Figure 3.7: Comparison of CPU and GPU reconstructions of the $CINE_{cart}$ data set with different regularization methods (left: $ICTGV$, center: TGV, right: TV). The differences are given for selected frames (rows) and rescaled for better visibility and the RMSE is depicted.

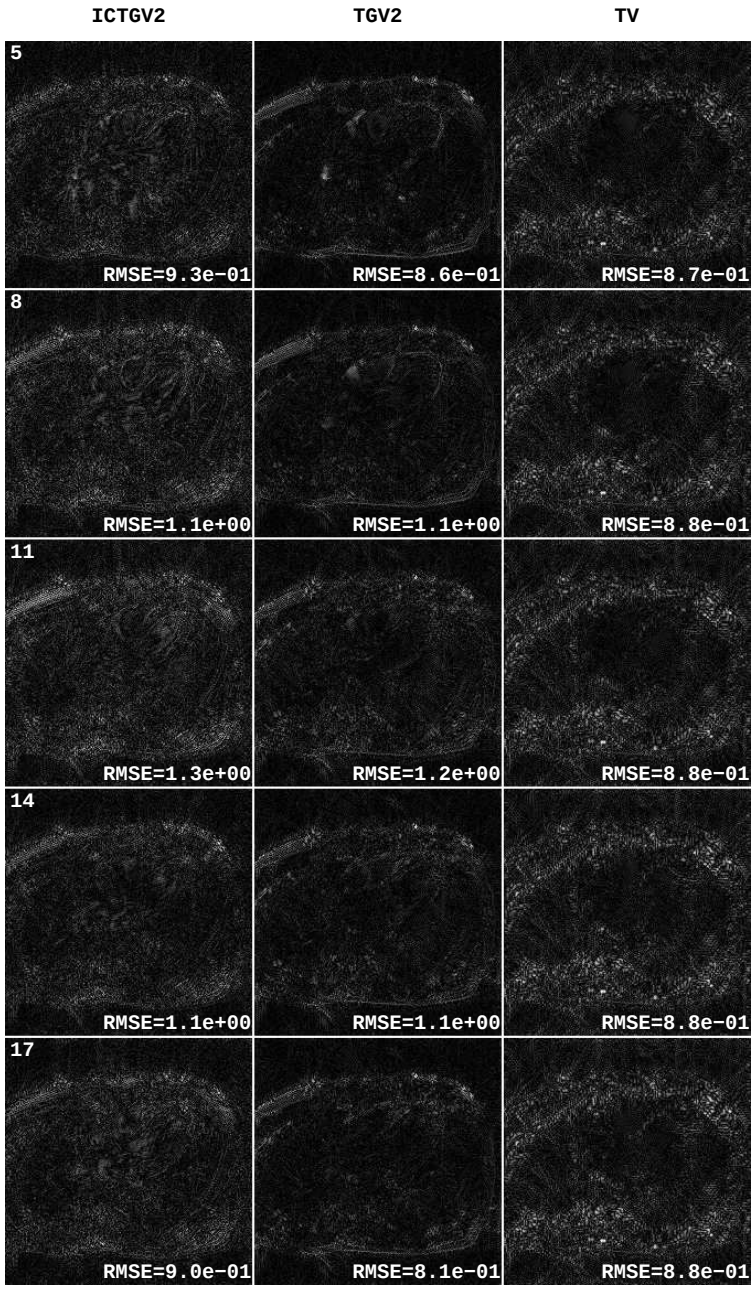


Figure 3.8: Comparison of CPU and GPU reconstructions of the $CINE_{radial}$ data set with different regularization methods (left: $ICTGV$, center: TGV, right: TV). The differences are given for selected frames (rows) and rescaled for better visibility and the RMSE is depicted.

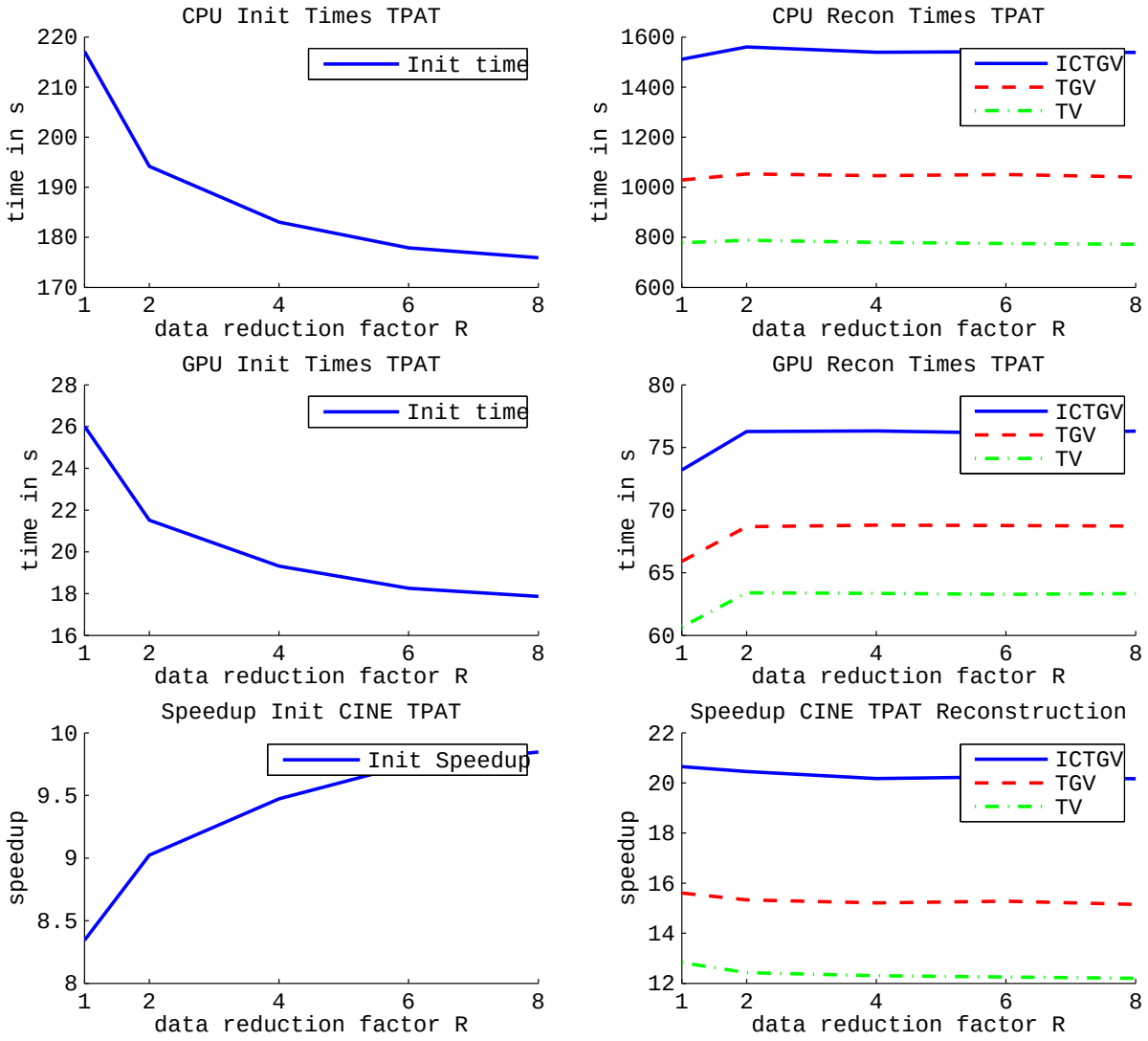


Figure 3.9: Cardiac CINE TPAT: Initialization (left) and reconstructions times (right) for the CPU (top row) and GPU (middle row) algorithms depending on the data subsampling factor R. The speedup against the subsampling factor is depicted as ratio between CPU and GPU in the bottom row.

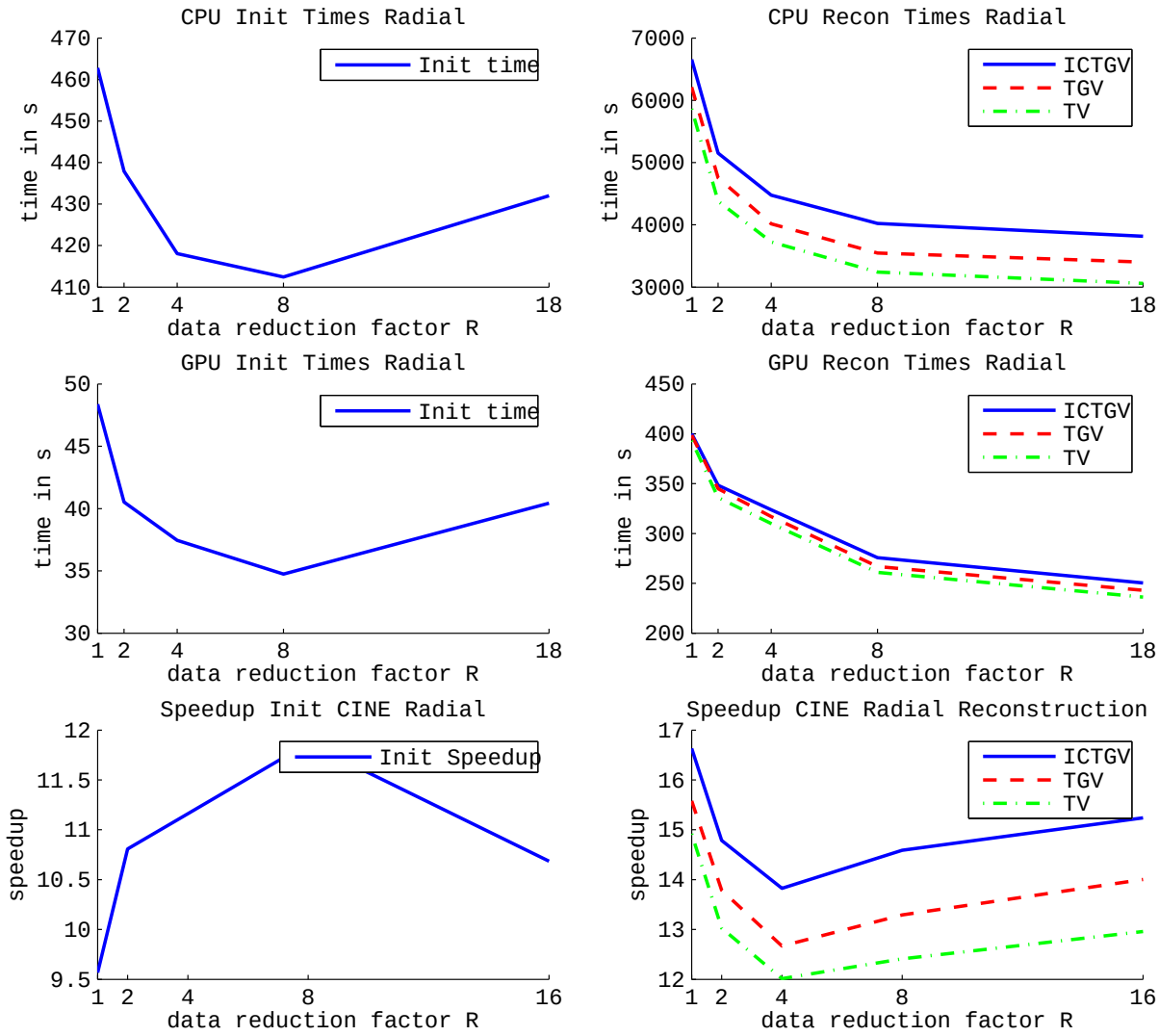


Figure 3.10: Cardiac CINE radial: Initialization (left) and reconstructions times (right) for the CPU (top row) and GPU (middle row) algorithms depending on the data subsampling factor R. The speedup against the subsampling factor is depicted as ratio between CPU and GPU in the bottom row.

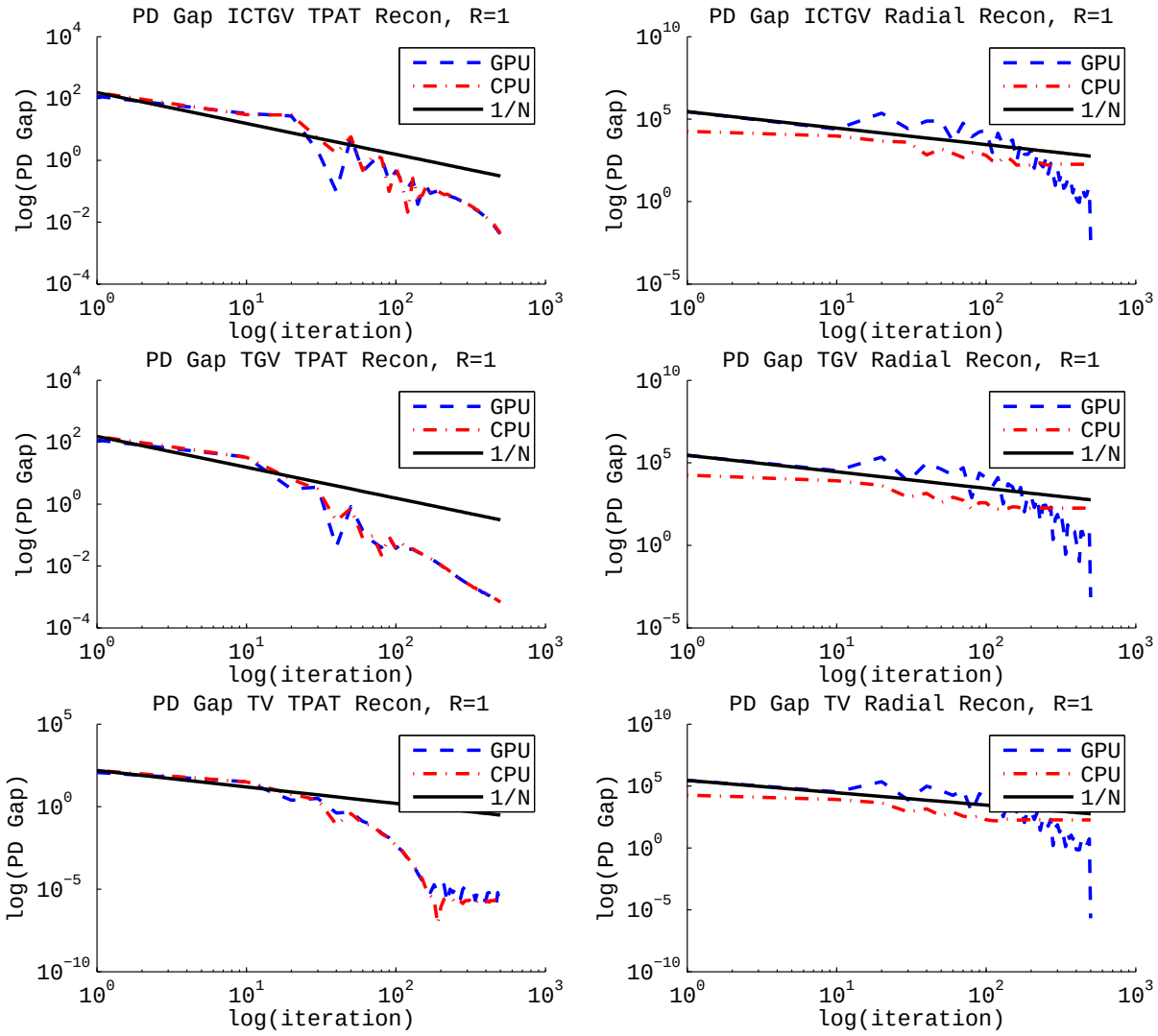


Figure 3.11: Development of the PD-Gap per iteration during the CPU and GPU reconstruction of the $CINE_{cart}$ (left) and the $CINE_{radial}$ (right) data sets with different regularization methods (rows). The Gap is plotted on a logarithmic scale.

3.1.3 Reconstruction Times Summary

Table 3.1: CPU and GPU computation times and speedup factors (SU) for coil reconstruction and *TV*, *TGV* and *ICTGV* regularization methods for $CINE_{cart}$ and $CINE_{radial}$ test cases. Speedup factor (SU) as ratio between CPU and GPU times.

		Coil			TV			TGV			ICTGV		
Data Set	R	CPU	GPU	SU	CPU	GPU	SU	CPU	GPU	SU	CPU	GPU	SU
	-	s	s	-	s	s	-	s	s	-	s	s	-
$CINE_{cart}$	1	217.0	26.0	8.3	778.4	60.7	12.8	1 028.5	65.9	15.6	1 511.3	73.2	20.7
	2	194.2	21.5	9.0	787.9	63.4	12.4	1 053.0	68.7	15.3	1 559.9	76.3	20.5
	4	183.0	19.3	9.5	779.4	63.3	12.3	1 046.5	68.8	15.2	1 539.4	76.3	20.1
	6	177.9	18.3	9.7	775.4	63.3	12.3	1 050.7	68.8	15.3	1 541.3	76.2	20.2
	8	175.9	17.9	9.9	772.6	63.3	12.2	1 041.1	68.7	15.2	1 538.5	76.3	20.2
$CINE_{radial}$	1	462.8	48.4	9.6	5 862.5	392.9	14.9	6 210.8	398.6	15.6	6 656.2	400.2	16.6
	2	437.9	40.5	10.8	4 378.6	336.1	13.0	4 754.3	344.9	13.8	5 151.3	348.4	14.8
	4	418.1	37.5	11.2	3 724.5	310.0	12.0	4 015.2	317.0	12.7	4 475.4	323.7	13.8
	6	412.4	34.8	11.9	3 240.0	261.0	12.4	3 547.4	266.9	13.3	4 022.1	275.7	14.6
	8	432.0	40.4	10.7	3 060.5	236.2	13.0	3 402.2	243.0	14.0	3 815.3	250.4	15.2

3.2 DCE Perfusion

The following section presents the reconstruction results for the DCE Perfusion data set $Perf_{cart}$. Figure 3.12 shows reconstructed absolute coil sensitivities after the initialization step for selected coils. The absolute values of three selected coils are depicted with respect to different data subsampling factors R.

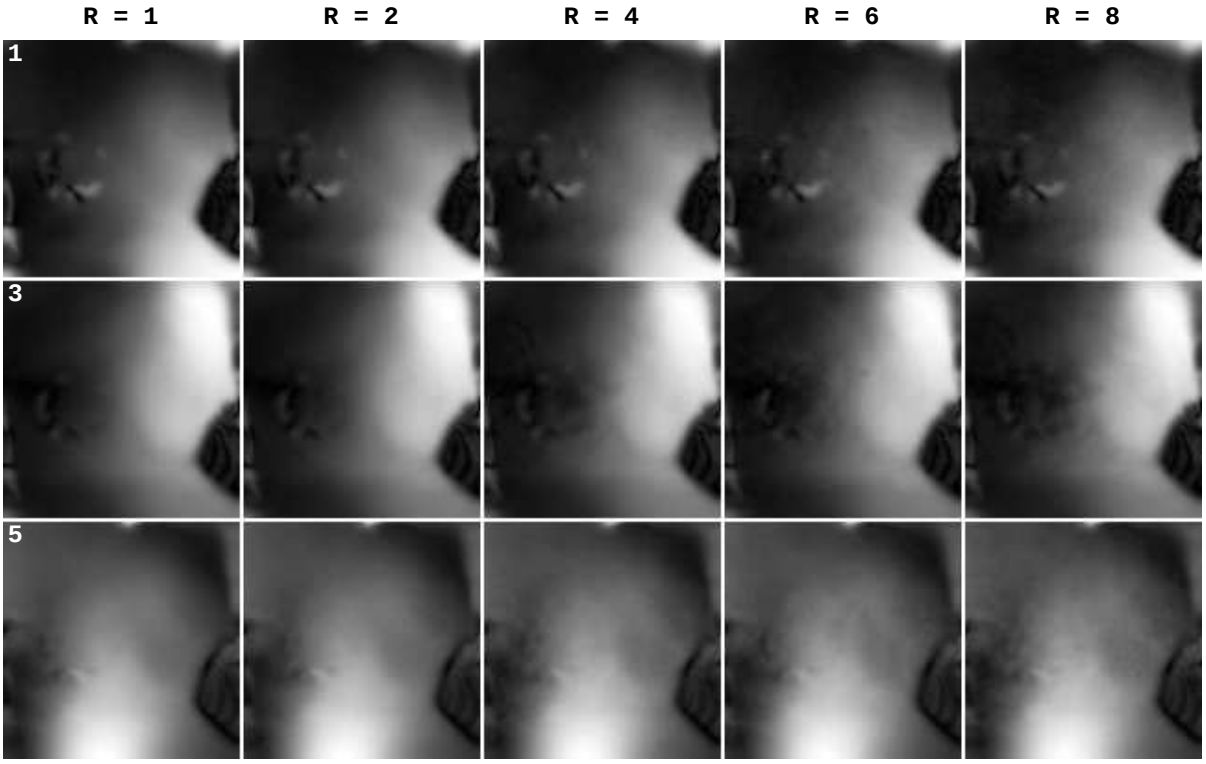


Figure 3.12: Sample absolute coil images (rows) of the $Perf_{cart}$ data set for the coils $1, 3, 5$. The reconstructed absolute coil images are presented for different data subsampling factors R (columns).

The Figures 3.13 and 3.14 contain reconstructed frame images of the $Perf_{cart}$ data sets with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the presented full reference frame and the undersampling artifacts due to the acquisition strategy are shown in a simple reference image after direct sum-of-squares (SOS) reconstruction.

The results of the applied GPU reconstruction in contrast to the reference CPU solution are presented as normalized absolute differences in Figures 3.15 and the root-mean-square-error (RMSE) per frame is depicted as an additional error estimate. The displayed differences are rescaled for better visibility.

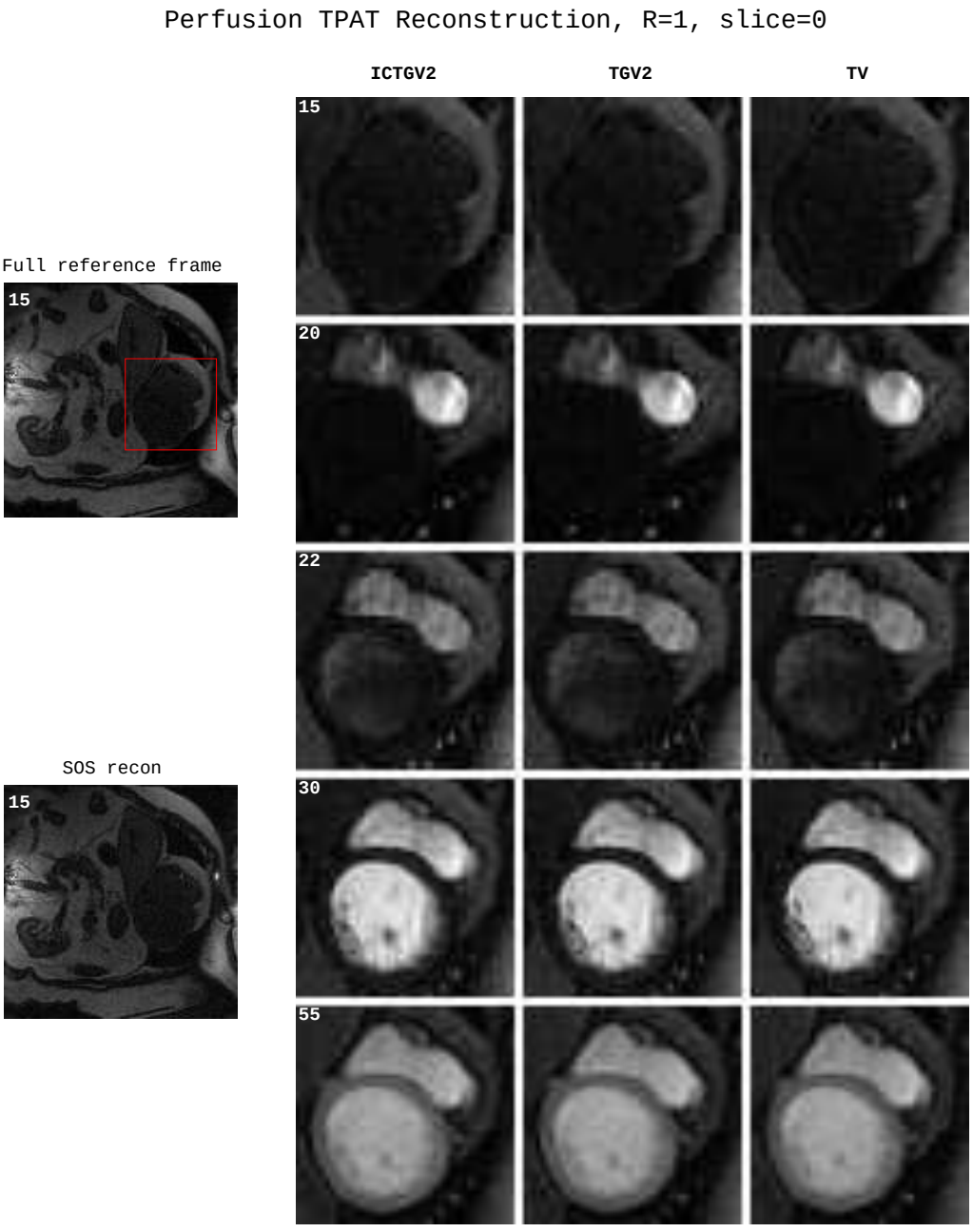


Figure 3.13: Reconstructed frame images (rows) of the $Perf_{cart}$ data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (top left). The basic undersampling artifacts are shown in the SOS reference image (bottom left).

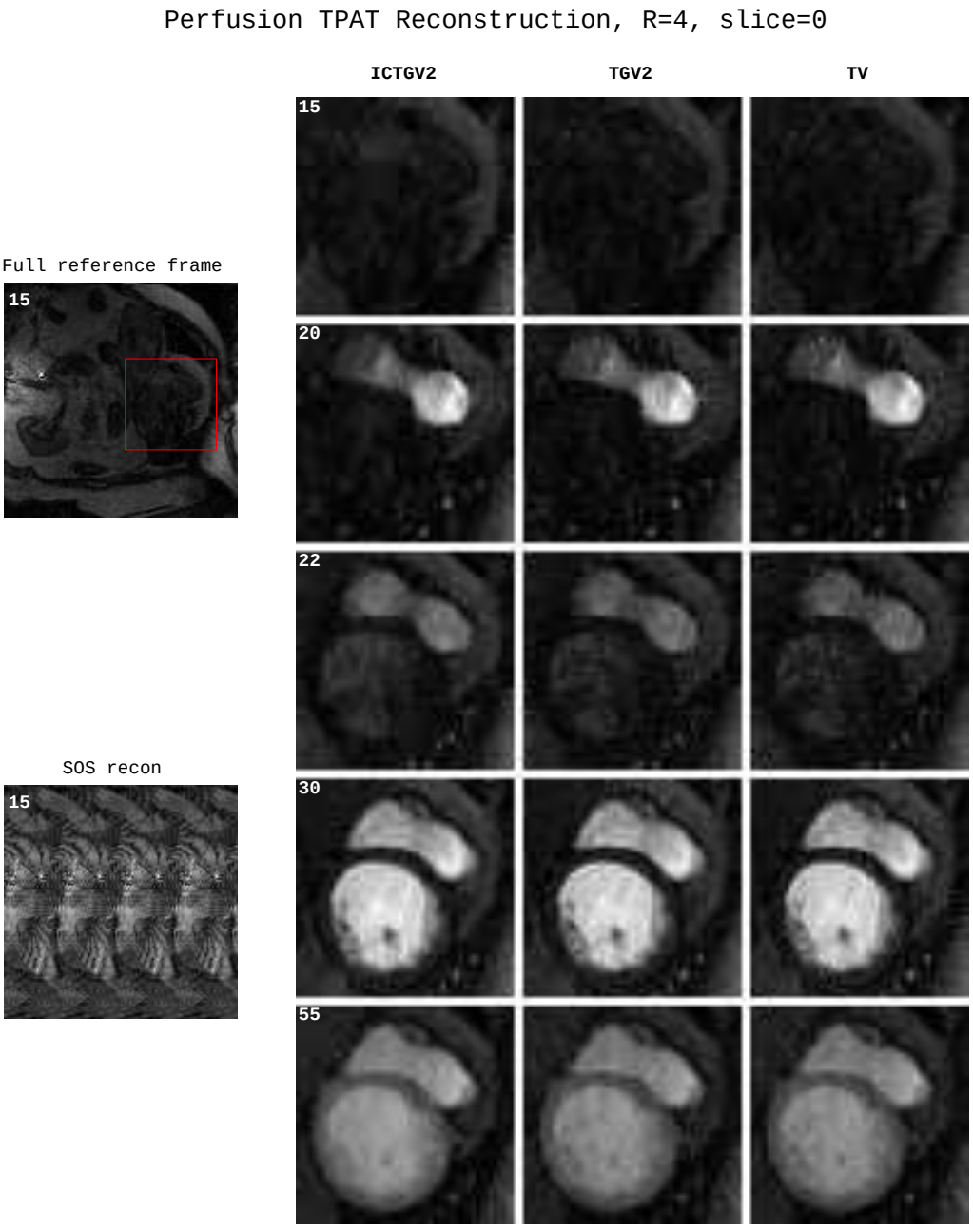


Figure 3.14: Reconstructed frame images (rows) of the $Perf_{cart}$ ($R = 4$) data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (top left). The basic undersampling artifacts are shown in the SOS reference image (bottom left).

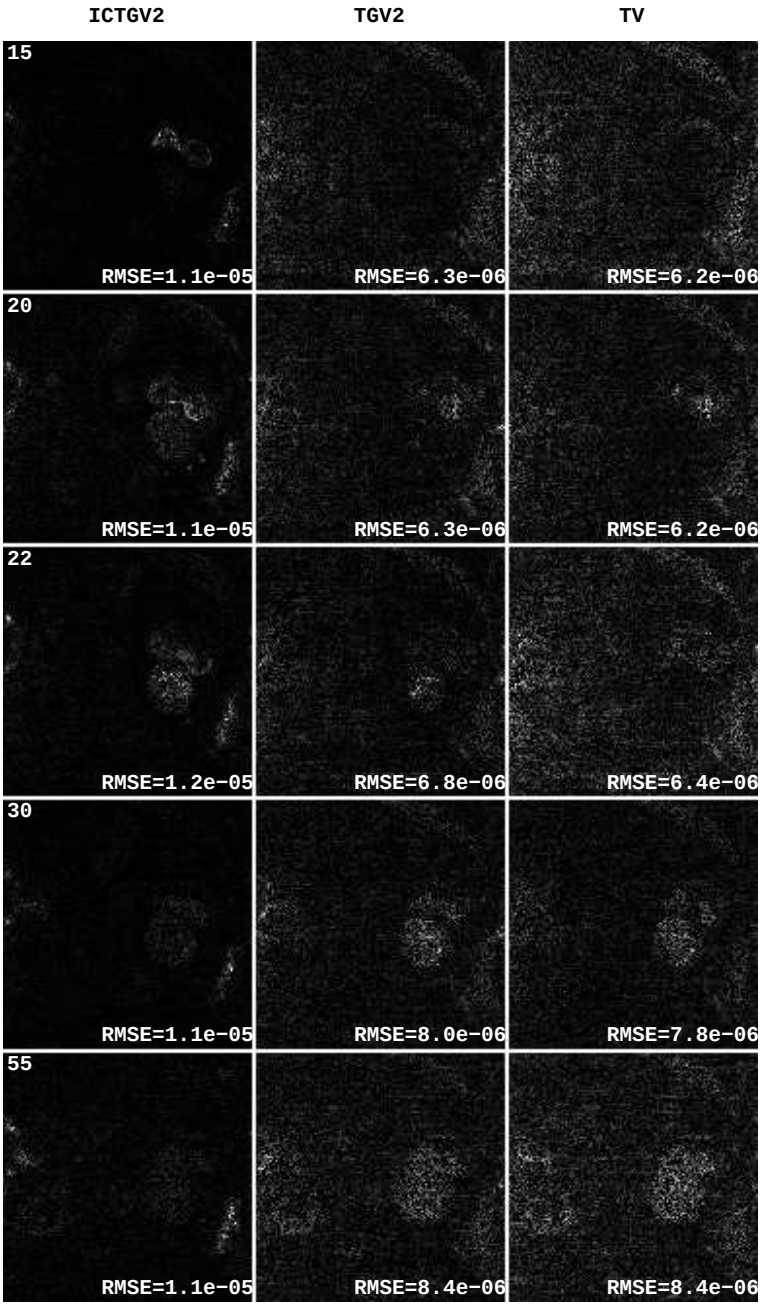


Figure 3.15: Comparison of CPU and GPU reconstructions of the $Perf_{cart}$ data set with different regularization methods (left: $ICTGV$, center: TGV, right: TV). The differences are given for selected frames (rows) and rescaled for better visibility and the RMSE is depicted.

3.2.1 Reconstruction Times

The comparisons of the initialization and reconstruction times for one slice are given in Figure 3.16. Each figure shows the CPU and GPU initialization times and speedup in the left column and the reconstruction times and reconstruction speedups in the right column. Table 3.2 summarizes the reconstruction times and speedups for the $Perf_{cart}$ data set for all applied undersampling factors R and slices.

3.2.2 PD Gaps

In order to show the convergence of the GPU and CPU algorithms, the progress of the Primal Dual Gap (PD-Gap) during the reconstruction of the $Perf_{cart}$ data sets with different regularization methods is depicted in Figure 3.17. The plot is drawn logarithmically and as a reference the $O(1/N)$ function is additionally shown.

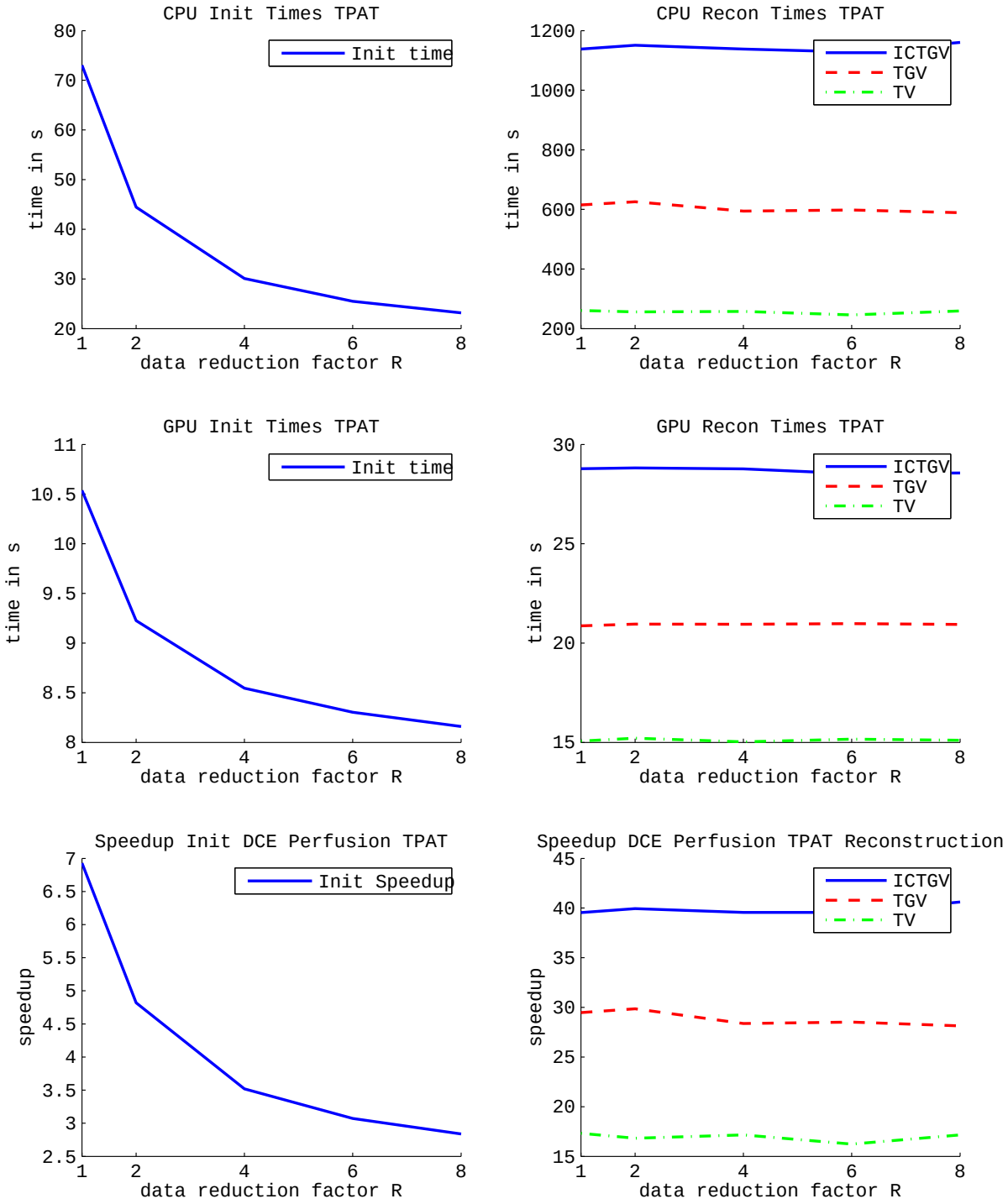


Figure 3.16: DCE perfusion test case: Initialization (left) and reconstructions times (right) for the CPU (top row) and GPU (middle row) algorithms depending on the data subsampling factor R. The speedup against the subsampling factor is depicted as ratio between CPU and GPU in the bottom row.

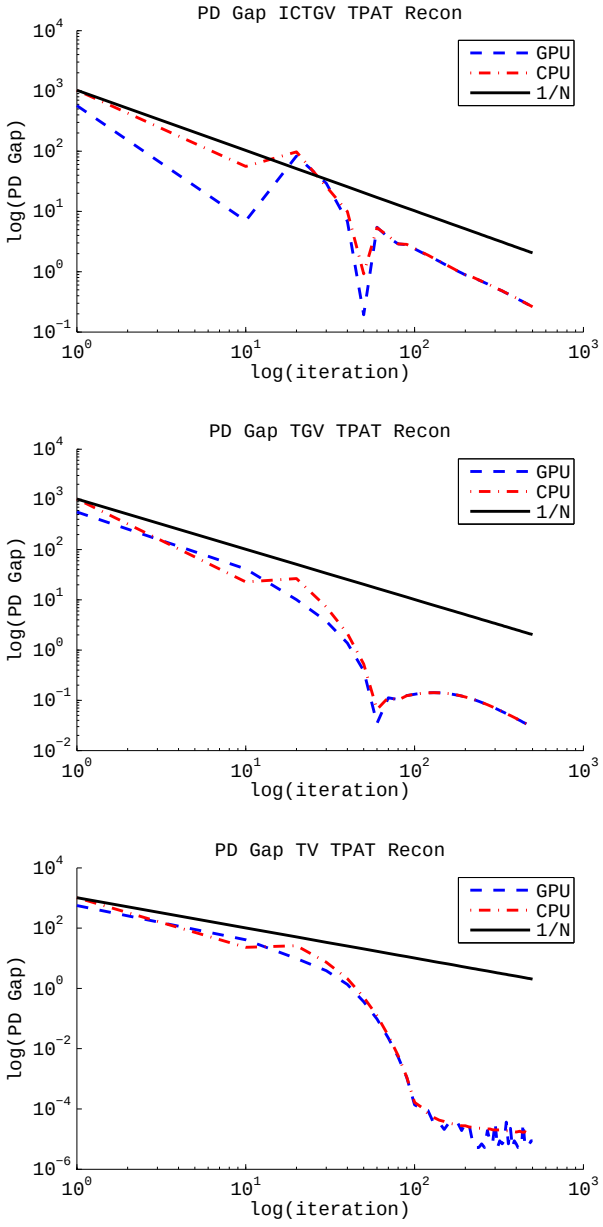


Figure 3.17: Development of the PD-Gap per iteration during the CPU and GPU reconstruction of the $Perf_{cart}$ data set with different regularization methods (rows). The Gap is plotted on a logarithmic scale.

3.2.3 Reconstruction Times Summary

Table 3.2: CPU and GPU computation times and speedup factors (SU) for coil reconstruction and *TV*, *TGV* and *ICTGV* regularization methods for all slices of the $Perf_{cart}$ test case. Speedup factor (SU) as ratio between CPU and GPU times.

		Coil Recon			TV			TGV			ICTGV		
Data Set	R	CPU	GPU	SU	CPU	GPU	SU	CPU	GPU	SU	CPU	GPU	SU
	-	s	s	-	s	s	-	s	s	-	s	s	-
$Perf_{cart}$ $slice\,0$	1	73.0	10.5	6.9	261.0	15.1	17.3	614.9	20.9	29.5	1 138.2	28.8	39.6
	2	44.5	9.2	4.8	256.0	15.2	16.8	625.5	21.0	29.9	1 150.8	28.8	39.9
	4	30.1	8.6	3.5	257.7	15.0	17.2	594.3	20.9	28.4	1 138.2	28.8	39.6
	6	25.5	8.3	3.0	246.2	15.2	16.2	598.2	21.0	28.5	1 128.6	28.5	39.6
	8	23.2	8.2	2.8	259.1	15.1	17.2	588.9	21.0	28.1	1 160.2	28.6	40.6
$Perf_{cart}$ $slice\,1$	1	71.1	10.6	6.7	257.3	15.2	16.9	609.6	20.9	29.2	1 175.4	28.8	40.8
	2	44.2	9.2	4.8	244.2	15.0	16.3	592.0	20.9	28.3	1 122.9	28.9	38.8
	4	30.1	8.5	3.5	256.4	15.0	17.1	590.3	20.8	28.4	1 124.1	28.8	39.0
	6	25.8	8.3	3.1	257.1	15.0	17.1	578.8	20.9	27.7	1 119.4	28.6	39.2
	8	23.4	8.1	2.9	246.9	15.1	16.4	590.0	20.9	28.2	1 144.7	28.7	39.9
$Perf_{cart}$ $slice\,2$	1	71.1	10.5	6.7	271.4	15.0	18.0	635.2	21.1	30.2	1 102.7	28.7	38.5
	2	43.8	9.2	4.7	261.1	15.2	17.2	615.2	20.9	29.4	1 102.9	28.8	38.3
	4	30.4	8.5	3.6	239.9	15.1	15.9	649.0	20.8	31.2	1 115.4	28.7	38.8
	6	25.7	8.3	3.0	236.5	15.1	15.6	660.3	20.8	31.7	1 161.3	28.9	40.2
	8	23.3	8.2	2.9	234.3	15.2	15.4	727.0	20.9	34.8	1 119.4	28.7	39.0

3.3 Raw Data Integration

The reconstructed frame images of the $CINE_{pf}$ and $CINE_{pf,ae}$ data sets with differently applied $ICTGV$, TGV and TV regularization methods are shown in Figures 3.18 and 3.19. The displayed image region of interest is zoomed as displayed in the presented full reference frame.

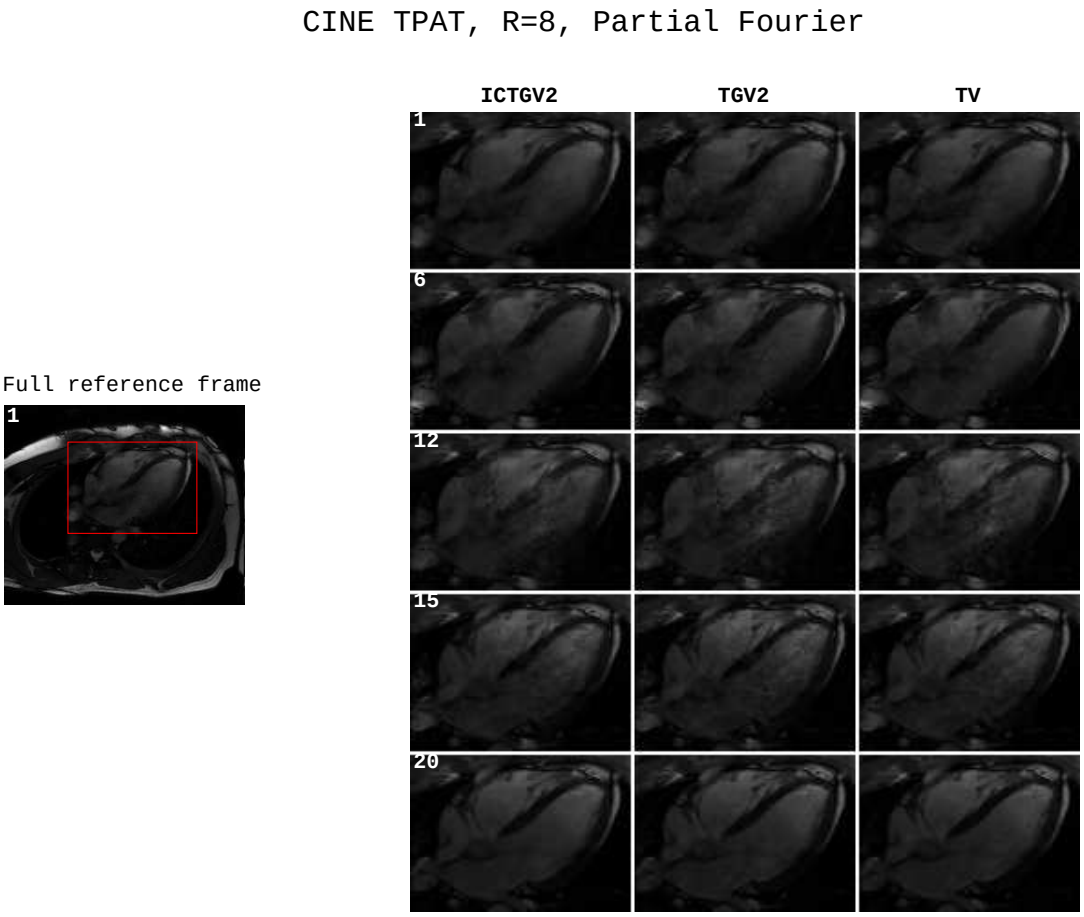


Figure 3.18: Reconstructed frame images (rows) of the $CINE_{pf}$ data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (middle left).

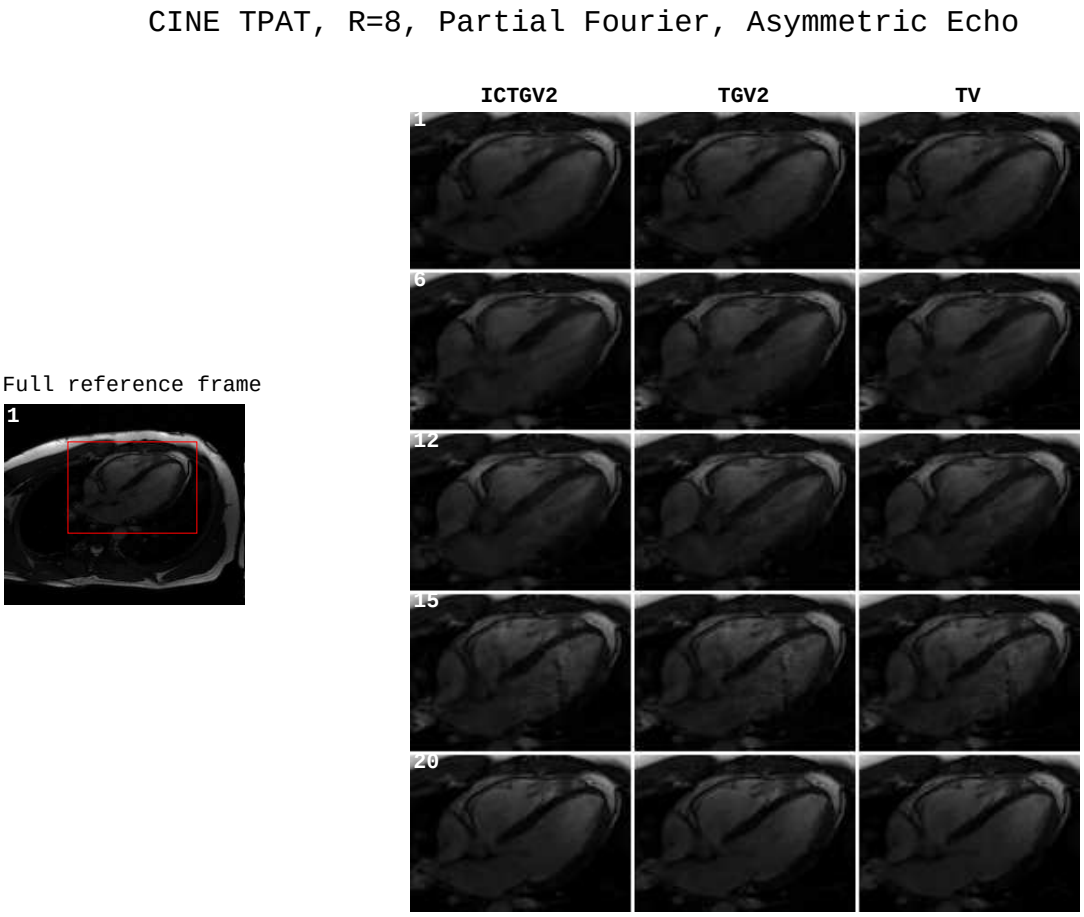


Figure 3.19: Reconstructed frame images (rows) of the $CINE_{pf,ae}$ data set with differently applied regularization methods (left: $ICTGV$, center: TGV, right: TV). The displayed image region of interest is zoomed as displayed in the full reference frame (middle left).

4 Discussion

4.1 Reconstruction performance

According to the results presented in Section 3, Page 42, it can be seen that a significant speedup between CPU and GPU is reached in comparing both the initialization and the reconstruction times. Depending on the data set, speedup factors from 10-20 in case of cardiac CINE imaging and 15-40 in case of DCE perfusion imaging can be reached.

4.1.1 Initialization

A possible maximum speedup of the initialization phase of approx. 10 can be seen, which enables fast and independent estimations of coil sensitivities. The coil images presented in Figures 3.1, 3.2 and 3.12 depict that the method results in a very robust estimation of sensitivities, also in case of large undersampling factors.

In common, the results depend on the time averaged coverage of k-space during all acquisitions. In case of shifted TPAT ($CINE_{cart}$ and $Perf_{cart}$) a nearly 100 percent coverage can be reached, whereas the acquisition of the $CINE_{radial}$ data set did not employ a shifted or rotated pattern, resulting in notable defective coil estimations at larger undersampling factors.

The initialization effort basically depends on the total amount of acquired data, because every single acquisition is prepared accordingly and hence the computation time decreases depending on the undersampling factor. Except the initialization of the radial data set slightly increases at $R = 18$ (Figure 3.10), because of an increased regularization effort (amount of iterations until convergence) during the coil estimation process.

4.1.2 Cardiac CINE

Reconstruction times

The Cardiac CINE overall *ICTGV* GPU reconstruction times (Table 3.1) of less than 2 minutes for Cartesian and less than 10 minutes for the non-Cartesian data set compared to 30 minutes and 2 hours on the CPU respectively, show that the presented algorithm allows reconstructions within the limits of clinical feasibility.

The largest speedup between CPU and GPU can be observed for the *ICTGV* reconstructions with a factor of 20 (Figure 3.3). The computational complexity leads to the best GPU utilization, in contrast to for example the less complex TV regularization, where the required effort is minor compared to synchronization and memory access costs.

Impact of undersampling

The TPAT undersampling factor has nearly no impact on the reconstruction times presented in Figure 3.9, since all operations of the MR operator, e.g. point-wise scaling or FFT, are based on the same matrix dimensions. It can be seen that the reconstruction of the fully sampled data set $R = 1$ is slightly faster, because the applied acquisition sequence only captured 23 compared to 24 frames (see Table 2.5).

In contrast, the radial CINE reconstruction times show a dependency on the subsampling factor (Figure 3.10), since basically the Gridding step (interpolation) in the non-uniform FFT computation is depending on the total amount of k-space samples.

Image reconstruction

The presented frame images of the CINE test sets show qualitatively good performances of the *TV*, *TGV* and *ICTGV* reconstructions, while *ICTGV* and *TGV* provide superior image quality and method-given less staircasing artifacts compared to the *TV* regularized images.

Less spatial resolution can be observed in the radial results, due to the above mentioned missing rotation in the acquisition pattern, no homogeneous coverage of the k-space has been recorded during time. A detailed image quality and parameter comparison has not been focus of this work and thus is topic of current research, like in [17].

The presented difference images in Figure 3.7 furthermore confirm the equivalence of the CPU and GPU reconstructions and show that the maximum RMSE is negligible in terms of the data scale of $[0, 255]$.

PD Gap

The PD Gap progression in Figure 3.11 shows that both CPU and GPU reconstructions converge at the same rate and thus can been seen as computationally equivalent within the floating point precision. The TV reconstruction shows the fastest progress and after about 200 iterations convergence within the numerical limits. The comparison to the theoretical $O(1/N)$ convergence rate [18] shows that all algorithms are basically well formulated on the presented data sets.

The radial data set shows different behavior, due to numerically different implementations between gpuNUFFT and the Fessler NUFFT. Both methods are self-adjoint, but the numerical results are not normalized in case of gpuNUFFT, leading to differently adapted step sizes during reconstruction. This may also be the cause that the PD Gap slightly exceeds the $O(1/N)$ limit during the iterations.

4.1.3 DCE Perfusion

Reconstruction times

The DCE Perfusion $ICTGV$ GPU reconstruction times for all three slices (Table 3.2) are less than 2 minutes, compared to approx. 1 hour on the CPU, and also support the clinical feasibility of the presented method.

In case of DCE Perfusion imaging, multiple slices (views) and a large amount of time frames have to be acquired, in order to guarantee the total capture of the contrast dynamics in the imaged region.

The data set description in Table 2.6 shows that this large temporal resolution can only be achieved by reduction of the spatial resolution leading to small reconstructed matrix sizes of $[128, 128]$. Therefore and the fact that less receiver coils have been used, considerably fast overall reconstruction times can be reached.

Impact of undersampling

Similarly to the results of the Cardiac CINE TPAT data sets, no strict dependency on the undersampling factor can be observed in the overall reconstruction times (Figure 3.16).

Image reconstruction

The reconstructed frame images show a good quality at TV, TGV and $ICTGV$ reconstructions, although artifacts are slightly more manifested in case of undersampled reconstruction (Figure 3.14). Again, staircasing artifacts are more perceptible in the TV reconstruction images.

PD Gap

The PD Gap progress also shows identical convergence of the CPU and GPU reconstructions and the resulting difference images prove equality within floating point precision.

4.2 Integration

The easy integration of any vendor specific raw data played a crucial role during development. In order to support arbitrary vendor data, the ISMRMRD raw data and interchange format has been selected, since it already provides predefined conversion scripts to transform arbitrary data to the standardized file format.

The file format uniformly encapsulates any required meta information, hence it is possible to describe every kind of (vendor-specific) accelerated raw data acquisition.

The two integration tests presented in Figures 3.18 and 3.19 show that Siemens acquired raw data with encoded Asymmetric Echoes, Partial Fourier and Rectangular FOV acceleration is supported and can be reconstructed with the presented software.

4.3 Implementation

As mentioned above, the primary C++ framework used in this work is the AGILE library, which is a well-defined CUDA template library with support for basic linear algebra functions. Furthermore, basic algorithms such as CG reconstructions, $2D$ forward/backward differences have already been integrated into the library. The modularity of the library allowed an easy expansion to the $3D$ case, and the benefit of reusable code in an already defined library structure.

Despite this, the AGILE data format is not sufficient for the requirements to describe the complex acquired raw data completely, since AGILE binary vectors do not support any additional meta information, for example to encode the data dimensions. Therefore, the use of the ISRMRMRD C++ library extends the framework very well and allows to scale the information arbitrarily, where no limits of meta information are given, due to the hierarchical data structure of $h5$ files. Furthermore, a built-in Matlab support for $h5$ files is given, which allows easy import/export functionality for existing Matlab projects and data sets.

The non-uniform FFT operator is computed by utilizing the gpuNUFFT library, which was seamlessly integrated to the reconstruction pipeline by the use of the provided generic C++ API.

The industry standard DICOM is supported as further output format, to provide subsequent console services with reconstruction images in the corresponding format. The DICOM files are generated by AGILE functionality.

4.4 Limitations and Outlook

As mentioned above the software presented in this work is currently only supported on Linux platforms. In order to being able to run on alternative operating systems, a port of the AGILE and other dependencies would be necessary.

Memory

The basic limitation for the use of the software is given by the required amount of global GPU device memory, which can be derived by the problem dimensions and the size of the acquired raw data. Table 2.4 shows the minimum amount of GPU

memory required for a given problem dimension. Currently, it is necessary that the whole problem solver (primal and dual vectors) and reconstruction data fits inside GPU memory.

It can be seen that the $ICTGV$ PD method requires about 1.5 times the memory of the TV solver, nevertheless, the presented clinical protocols for cardiac CINE and DCE perfusion could be reconstructed without limitations.

A possible extension to support larger data sets or even $3D$-t multi-coil datasets may exceed the available memory on current GPU models. It may be an interesting task to adapt the software to also support huge datasets, by only performing active computations on the GPU and by utilizing so-called streams to overlap memory-transfer operations and active computations.

Performance

The main bottleneck of performance in case of non-Cartesian imaging is the evaluation of the non-uniform FFT. The gpuNUFFT provides good results, although it is not perfectly optimized for $2D$ reconstructions. The method has been initially designed for large $3D$ datasets, and is implemented to perform the Gridding step in a memory-optimized online fashion. In contrast, less memory-intensive $2D$ data would allow to totally precompute the Gridding interpolation and keep the results on the GPU.

This would lead to less online computational effort and the precomputed interpolation values could be reusable especially for multi-coil data sets. This may further enhance the overall non-Cartesian reconstruction times.

Furthermore, the gpuNUFFT is not yet designed for temporal datasets, hence currently one gpuNUFFT instance has to be initialized per time frame, which leads to an extra memory overload. It may be interesting to extend the gpuNUFFT, in order to support time space data sets ($2D$-t) and to perform operations simultaneously on multiple time frames or in batched manner, like batched FFT computation, see [29] for more details. This may also hold for the FFT computation per frame for Cartesian data sets.

Online computation

The software provides an extensive C++ API, which would basically allow the integration into an online reconstruction pipeline, like the Yarra framework or the Gadgetron

pipeline [35], [36]. Moreover, the existing project may further be extended to be used as a $ICTGV$, TGV or TV undersampled reconstruction framework for time-independent pure $3D$ data sets.

The presented work allows fast reconstruction times of very challenging imaging modalities, as has been shown in the results section. The necessary reconstruction times are within clinical feasible limits and may lead to shortened cardiac protocols and hence to reductions of patient "table-times" in practice. In conclusion it can be noted that the software is ready to accelerate further research in the interesting field of dynamic MRI and to support the studies of the novel regularization concept of $ICTGV$.

Appendix

Algorithms

Algorithm 7 CG algorithm to solve linear system of equations

Initialize:

$x_0 = 0, r_0 = b, p_0 = r$

while $\|r_n\| > tol$ **do**:

$\alpha_n \leftarrow \frac{r_n^T \cdot r_n}{p_n^T \cdot A \cdot p_n}$

$x_{n+1} \leftarrow x_n + \alpha_n \cdot p_n$

$r_{n+1} \leftarrow r_n - \alpha_n \cdot A \cdot p_n$

$\beta_n \leftarrow \frac{r_{n+1}^T \cdot r_{n+1}}{r_n^T \cdot r_n}$

$p_{n+1} \leftarrow r_{n+1} + \beta_n \cdot p_n$

Code snippets

AGILE binary import/export

Listing 1: Import and export of AGILE binary format using MATLAB

```
%read AGILE binary file
addpath('/path/to/AGILE/include/agile/io');
data = readbin_vector('ictgv_recon_384_384_42.bin');
% reshape to problem dimensions
% and compensate column-major(Matlab)/row-major (CPP) order
data = permute(reshape(data, [384 384 42]),[2 1 3]);

%write AGILE binary file
%again convert column-major to row-major order
writebin_vector(permute(data,[2 1 3]),'test.bin');
```

ISMRMRD (h5) import/export

Listing 2: Import and export of H5 (ISMRMRD) format using MATLAB

```
% Load H5 file and ISMRMRD meta information
filename = '/path/to/ISMRMRDfile.h5';
header = ismrmrd_header2struct(h5read(filename,'/dataset/xml'));
% Pull Matrix size from header
Nx = str2num(header.encoding.reconSpace.matrixSize.x.Text);
Ny = str2num(header.encoding.reconSpace.matrixSize.y.Text);

% Read the raw data
```

```
raw_data = h5read(filename,'/dataset/data');

% Write h5 file
h5write('test_recon.h5','/dataset/recon',recon);
```

Source Code

The software created in this work is available as open-source on the community platform GitHub:

https://github.com/IMTtugraz/AVIONIC

Installation

Basically, the software can be compiled by using the CMake utility. The project has the following external dependencies:

- CMAKE 2.8
- CUDA Toolkit
- g++
- gpuNUFFT
- GoogleTest (Testing framework)
- Doxygen (for code docs)
- DCMTK (DICOM generation)

For more build instructions follow the README on the code repository.

Program Usage

```
Usage: avionic [options] [-d w:h:nRO:nEnc:c:f \
               <kdata> <mask/traj> | -r <rawdata>] <output>
Required:
  Either: -d             problem dimensions (see below)
          kdata          acquired k-space data file
          mask/traj      k-space sampling pattern (trajectory)
```

```
  or:     -r              enable raw data import (see below)
          rawdata         acquired rawdata file/directory
  output            reconstructed output image

Allowed options:
  -h [ --help ]                          show help message
  -g [ --debugstep ] arg (=10)           flag to export PDGap
  -v [ --verbose ]                       verbose console output
  -d [ --dims ] arg                      Data dimensions conforming to
                                         width:height:readouts:enc:coils:frames
  -n [ --nonuniform ]                    flag to indicate nonuniform data
  -e [ --extradata ]                     flag to enable export of additional
                                         result data
  -a [ --adaptlambda ]                   flag to enable dynamic adaptation of
                                         lambda depending on [adaptlambda]
                                         configuration paramters (k,d)
  -w [ --dens ] arg                      density compensation data.
  -p [ --params ] arg                    parameter configuration file
  -s [ --sens ] arg                      coil sensitivity data.
  -u [ --uzero ] arg                     initial image u0.
  -r [ --rawdata ]                       flag to indicate raw data import
  -f [ --forceOSRemoval ]                flag to force OS removal in raw data
                                         preparation
  -t [ --tpat ] arg (=1)                 artifical TPAT interleave
  -z [ --slice ] arg (=0)                slice to reconstruct
```

Example usage

Perform reconstruction (sensitivity estimation and *ICTGV* reconstruction) on pure ISMRMRD raw data file - enable automatic adaptation of lambda parameter '-a':

```
./avionic -r path/to/data/test_tpat.h5 -a result_tpat.h5
```

Perform reconstruction (sensitivity estimation and *ICTGV* reconstruction) on pure radial '-n' ISMRMRD raw data file and store result as DICOM file. Enable automatic

adaptation of lambda parameter '-a', pass parameter to use configuration file for non-Cartesian reconstruction '-p' and explicitly overwrite parameter 'coil.uH1mu':

```
./avionic -r path/to/data/test_radial.h5 -a -n -p default_noncart.cfg \
          --coil.uH1mu=5E-1 result_tpat.dcm
```

Perform *TGV*2 reconstruction based on AGILE binary vectors (e.g. exported from Matlab data) by passing the problem dimensions '-d', the required sensitivities '-s', density compensation '-w', $u0$ estimate '-u' and the required kspace data and trajectory vectors:

```
./avionic -d 384:384:384:14:12:42 -n -s b1.bin -w w.bin -u u0.bin \
          --maxIt 500 -p default_noncart.cfg -m TGV2 \
          data.bin trajectory.bin output.bin
```

Example source code

Example usage of raw data preparation

Listing 3: Raw data preparation

```
void PerformRawDataPreparation(Dimension &dims, OptionsParser &op,
                               CVector &kdata, RVector &mask,
                               RVector &w)
{
  bool completeData = true; //check for asymmetric echoes
                            //and partial fourier
  bool removeReadOutOS = true;//remove readout oversampling
  bool normalizeData = true;//normalize raw data
  bool applyChop = true;//prepare cartesian data with chop function

  RawDataPreparation rdp(op, completeData, removeReadOutOS,
                         normalizeData, applyChop);

  std::cout << "INFO: Loading RAW data from file/directory: "
            << op.kdataFilename << std::endl;
  std::string outputDir = utils::GetParentDirectory(op.outputFilename);

  rdp.PrepareRawData(kdata, mask, w, dims);

  // Extract dicom raw data and prepare it
  if (op.nonuniform)
  {
    ExportAdditionalResultsToMatlabBin(outputDir.c_str(), "w.bin", w);
  }

  ExportAdditionalResultsToMatlabBin(outputDir.c_str(), "mask.bin", mask);
  ExportAdditionalResultsToMatlabBin(outputDir.c_str(), "kdata.bin", kdata);
}
```

Example usage of Cartesian and non-Cartesian coil construction

Listing 4: Coil construction

```
void PerformCartesianCoilConstruction(Dimension &dims, OptionsParser &op,
                                      CVector &kdata, CVector &u,
                                      CVector &b1, RVector &mask,
                                      communicator_type &com)
{
  // Create MR Operator
  CartesianOperator *cartOp = new CartesianOperator(
      dims.width, dims.height, dims.coils, dims.frames, mask, false);
  CartesianCoilConstruction coilConstruction(
      dims.width, dims.height, dims.coils, dims.frames, op.coilParams, cartOp);
  coilConstruction.SetVerbose(op.verbose);

  coilConstruction.PerformCoilConstruction(kdata, u, b1, com);
  delete cartOp;
}


void PerformNonCartesianCoilConstruction(Dimension &dims, OptionsParser &op,
                                         CVector &kdata, CVector &u,
                                         CVector &b1, RVector &mask,
```

```
                                RVector &w, communicator_type &com)
{
  unsigned nFE = dims.readouts;
  unsigned spokesPerFrame = dims.encodings;
  unsigned int nTraj = dims.frames * spokesPerFrame * nFE;

  NoncartesianOperator *noncartOp = new NoncartesianOperator(
      dims.width, dims.height, dims.coils, dims.frames,
      spokesPerFrame * dims.frames, nFE, spokesPerFrame, mask, w);

  NoncartesianCoilConstruction nonCartCoilConstruction(
      dims.width, dims.height, dims.coils, dims.frames, op.coilParams,
      noncartOp);

  nonCartCoilConstruction.SetVerbose(op.verbose);
  nonCartCoilConstruction.PerformCoilConstruction(kdata, u, b1, com);
  delete noncartOp;
}
```

Example usage of $ICTGV$ PD solver

Listing 5: $ICTGV$ solver usage

```
agile::GPUEnvironment::allocateGPU(0);
agile::GPUEnvironment::printInformation(std::cout);
agile::GPUEnvironment::printUsage(std::cout);

unsigned int height = 384;
unsigned int width = 384;
unsigned int coils = 12;
unsigned int frames = 42;

unsigned int nFE = 384;
unsigned int nSpokesPerFrame = 14;
unsigned int N = width * height * frames;

// data
std::vector<CType> data;
agile::readVectorFile("data_384_14_12_42.bin", data);
CVector data_gpu(nFE * nSpokesPerFrame * frames * coils);
data_gpu.assignFromHost(data.begin(), data.end());

// b1 map
std::vector<CType> b1;
agile::readVectorFile("b1_384_384_12.bin", b1);
CVector b1_gpu(width * height * coils);
b1_gpu.assignFromHost(b1.begin(), b1.end());

// kspace mask
std::vector<RType> traj;
agile::readVectorFile("mask_384_14_42.bin", traj);
unsigned int nTraj = frames * nSpokesPerFrame * nFE;
RVector ktraj(2 * nTraj); // x + y coord
ktraj.assignFromHost(traj.begin(), traj.end());

// density data
std::vector<RType> wHost;
agile::readVectorFile("w_384_14_42.bin, wHost);
RVector w(nTraj);
w.assignFromHost(wHost.begin(), wHost.end());

// u0
std::vector<CType> u0;
agile::readVectorFile("u0_384_384.bin", u0);
CVector u0_gpu(width * height);
u0_gpu.assignFromHost(u0.begin(), u0.end());

CVector x_gpu(N);
for (unsigned frame = 0; frame < frames; frame++)
{
  agile::lowlevel::scale(1.0f, u0_gpu.data(),
                         x_gpu.data() + frame * width * height,
                         width * height);
}

BaseOperator *nonCartOp = new NoncartesianOperator(
    width, height, coils, frames, nSpokesPerFrame * frames, nFE,
    nSpokesPerFrame, ktraj, w, b1_gpu);

ICTGV2 ictgv2Solver(width, height, coils, frames, nonCartOp);

ictgv2Solver.SetVerbose(true);
ictgv2Solver.IterativeReconstruction(data_gpu, x_gpu, b1_gpu);

// get result
std::vector<CType> result(N);
x_gpu.copyToHost(result);

agile::writeVectorFile("ictgv_recon.bin", result);
delete nonCartOp;
```

Bibliography

[1] K. Pruessmann, M. Weiger, P. Boernert, and P. Boesiger, "Advances in Sensitivity Encoding With Arbitrary k-Space Trajectories," *Magnetic Resonance in Medicine*, pp. 638–651, 2001 (cit. on p. 1).

[2] P. Kellman, F. Epstein, and E. McVeigh, "Adaptive sensitivity encoding incorporating temporal filtering (TSENSE).," *Magnetic Resonance in Medicine*, vol. 45, no. 5, pp. 846–852, 2001 (cit. on p. 1).

[3] J. Tsao, P. Boesinger, and K. Pruessmann, "k-t BLAST and k-t SENSE: Dynamic MRI With High Frame Rate Exploiting Spatiotemporal Correlations," *Magnetic Resonance in Medicine*, vol. 50, pp. 1031–1042, 2003 (cit. on p. 1).

[4] J. Tsao, "On the UNFOLD method.," *Magnetic Resonance in Medicine*, vol. 47, no. 1, pp. 202–207, 2002 (cit. on p. 1).

[5] Y. Wang and L. Ying, "Compressed sensing dynamic cardiac cine MRI using learned spatiotemporal dictionary.," *IEEE Transactions on bio-medical engineering*, vol. 61, no. 4, pp. 1109–1120, 2014 (cit. on p. 1).

[6] L. Feng, M. Srichai, R. Lim, *et al.*, "Highly accelerated real-time cardiac cine MRI using k-t SPARSE-SENSE.," *Magnetic Resonance in Medicine*, vol. 70, no. 1, pp. 64–74, 2013 (cit. on p. 1).

[7] R. Otazo, E. Candes, and D. Sodickson, "Low-rank plus sparse matrix decomposition for accelerated dynamic MRI with separation of background and dynamic components," *Magnetic Resonance in Medicine*, vol. 73, no. 3, pp. 1125–1136, 2015 (cit. on p. 1).

[8] D. Atkinson and R. Edelman, "Cineangiography of the heart in a single breath hold with a segmented turboFLASH sequence.," *Radiology*, vol. 178, no. 2, pp. 357–360, 1991 (cit. on p. 2).

[9] S. Sourbron and D. Buckley, "Classic models for dynamic contrast-enhanced MRI.," *NMR in Biomedicine*, vol. 26, no. 8, pp. 1004–1027, 2013 (cit. on p. 2).

[10] G. McGibney, M. Smith, S. Nichols, and A. Crawley, "Quantitative evaluation of several partial Fourier reconstruction algorithms used in MRI.," *Magnetic Resonance in Medicine*, vol. 30, no. 1, pp. 51–59, 1993 (cit. on p. 5).

[11] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms.," *Phyisca D*, vol. 60, pp. 259–268, 1992 (cit. on p. 7).

[12] F. Knoll, K. Bredies, T. Pock, and R. Stollberger, "Second order total generalized variation (TGV) for MRI.," *Magnetic Resonance in Medicine*, vol. 65, no. 2, pp. 480–491, 2011 (cit. on pp. 7, 10, 12, 20, 26).

[13] K. Block, M. Uecker, and J. Frahm, "Undersampled Radial MRI with Multiple Coils. Iterative Image Reconstruction Using a Total Variation Constraint," *Magnetic Resonance in Medicine*, vol. 57, pp. 1086–1098, 2007 (cit. on p. 7).

[14] F. Knoll, C. Clason, K. Bredies, *et al.*, "Parallel Imaging With Nonlinear Reconstruction Using Variational Penalties.," *Magnetic Resonance in Medicine*, vol. 67, no. 1, pp. 34–41, 2012 (cit. on p. 7).

[15] K. Bredies, K. Kunisch, and T. Pock, "Total Generalized Variation," *SIAM Journal on Imaging Sciences*, vol. 3, no. 3, pp. 492–526, 2010 (cit. on p. 7).

[16] M. Holler and K. Kunisch, "On Infimal Convolution of TV Type Functionals and Applications to Video and Image Reconstruction.," *SIAM Journal on Imaging Sciences*, vol. 7, no. 4, pp. 2258–2300, 2014 (cit. on pp. 7, 8).

[17] M. Schloegl, M. Holler, A. Schwarzl, *et al.*, "Infimal Convolution of Total Generalized Variation Functionals for dynamic MRI.," *To appear in Magnetic Resonance in Medicine*, 2016 (cit. on pp. 8, 19, 20, 66).

[18] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of Mathematical Imaging and Vision*, vol. 40, no. 1, pp. 120–145, 2011 (cit. on pp. 8, 10, 11, 25, 67).

[19] J. Fessler and B. Sutton, "Nonuniform Fast Fourier Transforms Using Min-Max Interpolation," *IEEE Transactions on Signal Processing*, no. 2, pp. 560–574, Feb. 2003 (cit. on p. 10).

[20] M. Schloegl, M. Holler, K. Bredies, and R. Stollberger, "A Variational Approach for Coil-Sensitivity Estimation for Undersampled Phase-Sensitive Dynamic MRI Reconstruction.," in *Proceedings of the 23th Annual Meeting of ISMRM*, Toronto, 2015, p. 3692 (cit. on pp. 19, 23).

[21] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, tenth edition. National Bureau of Standards, 1964 (cit. on p. 20).

[22] J. Fessler. (2012). Image reconstruction toolbox, [Online]. Available: http://web.eecs.umich.edu/~fessler/code/ (cit. on pp. 21, 38).

[23] F. Knoll, A. Schwarzl, C. Diwoky, and D. Sodickson, "gpuNUFFT - An Open-Source GPU Library for 3D Gridding with Direct Matlab Interface," in *Proceedings of the 22th Annual Meeting of ISMRM*, Mailand, 2014, p. 4297. [Online]. Available: http://submissions.miracd.com/ismrm2014/proceedings/ (cit. on pp. 21, 31).

[24] M. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, pp. 409–436, 6 Dec. 1952 (cit. on p. 25).

[25] M. Freiberger, F. Knoll, K. Bredies, *et al.*, "The agile library for image reconstruction in biomedical siences using graphics card hardware acceleration.," *Computing in Science and Engineering*, pp. 34–44, 2013 (cit. on pp. 25, 28).

[26] NVIDIA. (2015). CUDA C Programming Guide, Version 7.5, [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (cit. on pp. 28, 29).

[27] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*, first edition. Addison-Wesley, 2011 (cit. on p. 28).

[28] D. Kirk and W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, first edition. Elsevier Inc., 2010 (cit. on p. 28).

[29] NVIDIA. (2015). CUFFT Library User's Guide, Version 7.5, [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf (cit. on pp. 31, 70).

[30] Hansen, Block, Griswold, *et al.* (2015). ISMRMRD Raw Data Format, [Online]. Available: http://ismrmrd.github.io/ (cit. on p. 33).

[31] NEMA. (2015). DICOM - Digital Imaging and Communications in Medicine, [Online]. Available: http://dicom.nema.org/ (cit. on p. 33).

[32] The HDF Group. (2015). HDF - Hierarchical File Format, [Online]. Available: https://www.hdfgroup.org/HDF5/ (cit. on p. 33).

[33] OFFIS computer science institute. (2015). DCMTK - DICOM Toolkit, [Online]. Available: http://dicom.offis.de (cit. on p. 33).

[34] M. Schloegl, M. Holler, K. Bredies, *et al.*, "ICTGV Regularization for Highly Accelerated Dynamic MRI.," in *Proceedings of the 23th Annual Meeting of ISMRM*, Toronto, 2015 (cit. on p. 38).

[35] Y. Framework. (2016). Center for Advanced Imaging Innovation and Research, [Online]. Available: https://yarra.rocks/ (cit. on p. 71).

[36] M. Hansen and T. Sorensen, "Gadgetron: An Open Source Framework for Medical Image Reconstruction," *Magnetic Resonance in Medicine*, vol. 69, no. 6, pp. 1768–76, 2013 (cit. on p. 71).