Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Multi-GPU Implementation of MR-STAT for Quantitative MRI Reconstruction

Author: Ivy Rui Wang (2801549)

1st supervisor:	Adam S.Z. Belloum	(University of Amsterdam)
daily supervisor:	Alessio Sclocco	(Netherlands eScience Center)
2nd reader:	Stijn Heldens	(Netherlands eScience Center)
3rd reader:	Oscar van der Heide	(UMC Utrecht)

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

June 29, 2026

"I am the master of my fate, I am the captain of my soul"
from Invictus*, by William Ernest Henley*

Abstract

Context. Quantitative MRI reconstruction is computationally challenging. This thesis extends MR-STAT, a fast method for multi-parametric quantitative MR, to multiple GPUs. The single-GPU implementation can run out of memory for large two-dimensional images.

Goal. The goal is to design, implement, and evaluate a multi-GPU version that preserves the numerical behaviour of the single-GPU reconstruction.

Method. The design partitions voxels across GPUs. Bloch simulation, derivative computations, and adjoint Jacobian application remain local to each voxel range. The simulated signal, forward Jacobian-vector product, and solver scalar quantities need synchronization.

Results. The multi-GPU implementation matches the single-GPU baseline numerically for both noiseless and noisy data. The convergence analysis shows the same optimisation behaviour across GPU configurations. Strong scaling at fixed problem size reduces runtime as GPUs are added. Problem-size scaling shows the main benefit: multi-GPU reconstruction extends the feasible image size beyond the single-GPU memory limit.

Conclusions. Voxel partitioning is a suitable multi-GPU strategy for this MR-STAT reconstruction because it preserves numerical behaviour while distributing voxel-indexed memory and computation. The implementation mainly improves capacity for larger reconstructions, while speedup and efficiency scaling are limited by frequent communications.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

1.1 Motivation

Magnetic resonance imaging (MRI) is often used qualitatively. It produces images whose intensities are weighted by tissue properties and acquisition settings but are arbitrary pixel intensities that cannot be accurately compared across different MRI machines, hospital sites, and scanning sessions. Quantitative MRI (qMRI) aims to estimate physical tissue parameters instead, such as T_1 and T_2, which have explicit physical units (e.g., seconds or milliseconds) (1). This thesis studies a qMRI reconstruction method based on MR-STAT, a new method for fast, multi-parametric quantitative MR (2). MR-STAT estimates quantitative parameter maps by directly fitting a Bloch-equation based volumetric signal model to the measured time-domain signal. The signal model includes the transient-state magnetisation dynamics, spatial gradient encoding, and receive-coil sensitivity. Spatial localisation and parameter estimation are therefore solved together in one nonlinear optimisation problem.

1.2 Computational Challenge

The reconstruction evaluates a Bloch simulation for many voxels. It then solves a nonlinear least-squares problem with an iterative trust-region method. Inside this method, an inner conjugate-gradient loop repeatedly applies matrix-free Hessian-vector products. These computations are expensive, and GPU acceleration has made this reconstruction practical on a single GPU for two-dimensional images (3).

A single GPU still limits the reconstruction in two ways. First, all intermediate arrays used by the reconstruction must fit in the memory of that GPU. For large image sizes,

the required intermediate state can exceed the available device memory. In that case, the reconstruction is not merely slow but it cannot run on one GPU.

Second, the runtime remains significant because the forward model is evaluated many times during the outer and inner iterations. Using more GPUs could therefore help both with memory capacity and with runtime.

1.3 Gap in Existing Multi-GPU MRI Reconstruction

MRI reconstruction methods expose different computational structures. Partial Fourier reconstruction exploits redundancy in k-space (4). Parallel imaging uses receive-coil sensitivities as additional spatial encoding (5, 6). Compressed sensing adds sparsity constraints to the reconstruction problem (7, 8). Deep learning methods can learn mappings from k-space to image space or combine learned components with physical data consistency (4).

Multi-GPU parallelism has been used to address the computational and memory demands of these reconstruction methods (9). Existing multi-GPU work includes field-corrected iterative reconstruction and real-time reconstruction frameworks (10, 11), compressed sensing (12, 13), and deep learning (14, 15). Many methods outside deep learning partition the work along slices, receive coils, or temporal frames (10, 11, 13, 16, 17). These strategies are effective when the reconstruction pipeline exposes dimensions that can be processed with little communication. In many cases, the remaining communication is limited to final aggregation. When dependencies occur inside the reconstruction itself, the implementation must manage them through sequential frame handling, ghost-region exchange, synchronisation buffers, or transfers through host memory (13, 14, 17, 18).

The reconstruction considered in this thesis is a single two-dimensional slice. This means that slice-level parallelism is not available inside the reconstruction itself. Existing multi-GPU MRI reconstruction work mainly exploits higher-level dimensions such as slices, receive coils, temporal frames, or model blocks. The gap is therefore at the level of the dependency structure: how to parallelise a nonlinear qMRI reconstruction when the useful work split is over voxels inside the same slice, while the measurement-domain signal still couples all voxel partitions. In this setting, much of the Bloch and derivative work is voxel-local. However, the signal predicted by the forward model and the forward Jacobian-vector product require summing contributions from all voxel partitions. Chapter 4 analyzes this dependency structure in detail.

1.4 Research Objectives

The work has three objectives.

- **O1 (Dependency analysis).** Starting from the reconstruction algorithm and the foward signal model, explain why voxel partitioning is exploited and which communications are necessary. Analyse the reconstruction to identify which computations are voxel-local and which quantities require cross-voxel summation.
- **O2 (Multi-GPU implementation).** Realise this design in code through rank-to-GPU binding, voxel range assignment, rank-local data representation, MPI reductions, matrix-free Hessian-vector products, and solver operations on partitioned vectors.
- **O3 (Empirical characterisation).** Characterise the resulting implementation in terms of reconstruction quality, numerical behaviour, runtime scaling, and the problem-size scaling.

1.5 Thesis Structure

The rest of the thesis is organised as follows. Chapter 2 introduces quantitative MRI, Bloch simulation and the forward signal model, nonlinear inverse reconstruction, GPU computing, MPI, and existing MRI reconstruction partitioning strategies. Chapter 3 gives a high-level outline of the reconstruction workflow. Chapter 4 derives the multi-GPU design from the MR-STAT, including the voxel partitioning strategy and the communication cost. Chapter 5 explains how the design is realised in Julia, CUDA, and MPI. Chapter 6 evaluates numerical equivalence, convergence behaviour, strong scaling, per-iteration timing, and problem-size scaling. Chapter 7 discusses the implications of the results. Chapter 8 analyses threats to validity. Chapter 9 compares this work with previous multi-GPU MRI reconstruction work. Chapter 10 concludes the thesis.

2

Background

This chapter gives the background for the multi-GPU reconstruction developed in this thesis.

2.1 Quantitative MRI and Forward Model

Magnetic resonance imaging measures signals from hydrogen nuclei after they are excited by radiofrequency pulses inside a static magnetic field. After excitation, the tissue relaxes through longitudinal relaxation with time constant T_1 and transverse relaxation with time constant T_2 (4). Conventional MRI usually produces weighted images, whose intensities depend on both tissue properties and acquisition settings. Quantitative MRI (qMRI) is commonly defined as extracting an image-derived characteristic whose magnitude can be expressed as a number with units or relative to a reference material (19). The qMRI reconstruction studied in this thesis follows the MR-STAT paper (2, 3). For a given tissue-parameter estimate, Bloch simulation gives the magnetisation state of each voxel over the readouts. The spatial gradient encoding and receive-coil sensitivities are then included in the signal model that maps these voxel states to the simulated signal. Spatial localisation and parameter estimation are solved together in one large nonlinear optimisation problem.

2.2 Nonlinear Inverse Problem

A nonlinear inverse problem is a mathematical challenge of deducing unknown underlying parameters from indirect measurements, where the forward model relating causes to effects depends nonlinearly on the unknowns. These problems are ill-posed, meaning their solutions lack uniqueness, existence, or stability. The reconstruction studied here is

a nonlinear inverse problem because the measured time-domain signal is predicted from the voxel-wise tissue parameters through Bloch simulation, and the parameters are estimated by minimising the least-squares mismatch between the simulated and measured signals (2). This nonlinear inverse problem can be solved iteratively (2, 20). The outer iteration uses a trust-region inexact Gauss–Newton method, which linearises the forward signal model at the current parameter estimate and chooses an update within a trusted region. The inner iteration uses the Steihaug conjugate-gradient method to approximately solve the trust-region subproblem. Each inner iteration needs a Gauss–Newton Hessian-vector product, evaluated as a forward Jacobian-vector product followed by an adjoint Jacobian application. The derivative matrices are not stored explicitly. The Jacobian and the Gauss–Newton Hessian approximation are used only through products with vectors. This matrix-free structure avoids storing very large matrices, but it repeatedly evaluates Bloch simulation and derivative information for many voxels. It also stores magnetisation-state arrays and finite-difference derivative arrays indexed by readout and voxel. These computations and arrays are the main reason why the reconstruction is both expensive and memory-demanding on a single GPU. The Bloch simulation for one voxel can be computed independently of the others. However, the simulated signal sample is formed after spatial encoding has combined contributions from all voxels. Each simulated signal sample is therefore obtained by summing the encoded contributions from all voxels.

2.3 Multi-GPU Execution and Bottlenecks

GPU acceleration is effective in MRI reconstruction when the computation exposes many independent operations (9). For example, the voxel-wise Bloch simulation in this thesis can be evaluated independently for every voxel before the voxel contributions are combined. The best way to use the GPUs depends on the problem size, the data dimensions, and the memory constraints. Murphy et al. report that image-domain SPIRiT interpolation can become memory-limited because interpolation kernels must be stored for many coil pairs (12). More recent MRI reconstruction work reports similar memory and data-movement constraints in different forms. Plummer et al. use coil sketching to lower the memory footprint of four-dimensional lung MRI by solving reduced coil subproblems while periodically using full-coil information to guide convergence (21). Shafique et al. use compressed SVD to reduce the cost of the low-rank step in dynamic MRI, but they also report that the speedup drops when CPU–GPU transfer time is included (22). Fujita et al. use gradient checkpointing for quantitative parameter mapping. During the forward

pass, selected denoiser outputs are saved as checkpoints instead of storing all intermediate gradient containing gradient information. During the backward pass, intermediate results are recomputed from the nearest checkpoint in the forward direction, trading extra computation for lower GPU memory use (23). Iakovlev et al. treat an unrolled reconstruction network as a fixed-point iteration with shared weights across blocks. Their Jacobian-free backpropagation computes the gradient approximation through the last block instead of backpropagating through all unrolled blocks, which avoids memory growth with the number of blocks (24). The memory limitation in this thesis is introduced in Section 3.1. Using several GPUs provides more aggregate memory and more parallel compute, but the benefit depends on how much data must be exchanged between GPUs. Several GPUs, especially across multiple compute nodes, do not share one memory space. MPI represents this setting with ranks, where each rank runs the same program on its own data and communicates explicitly with other ranks. Each rank contributes an array, MPI sums the arrays element by element, and every rank receives the summed result. Communication can reduce the benefit of parallel computation. Schaetz et al. observed that peer-to-peer aggregation became a bottleneck beyond two GPUs because PCIe transfers introduced contention and synchronisation delays (11). This issue is relevant here because the simulated signal and forward Jacobian-vector product must be assembled from partial results computed on different GPUs. Section 4.5 explains how this voxel summation becomes a communication cost in the multi-GPU design.

2.4 Partitioning Strategies in Multi-GPU MRI Reconstruction

Multi-GPU MRI reconstruction usually starts by choosing a dimension along which the work can be split (9). Previous work has partitioned independent image slices (10, 16, 25), receive coils (11, 21), temporal frames (13, 17, 18), subspace or block structure (26, 27), or neural network modules in deep learning reconstruction (14, 15). These strategies are effective when the selected dimension exposes mostly independent work.

Some methods only need final aggregation after independent reconstruction tasks have finished. This is different from a dependency inside the repeated reconstruction loop. When intermediate results are needed before the next computation step can proceed, the implementation must manage communication through reductions, ghost regions, synchronisation buffers, or host-memory transfers (11, 13, 14, 17).

2. BACKGROUND

The reconstruction in this thesis targets one two-dimensional slice. Slice-level parallelism is therefore not available inside this reconstruction problem. Coil or temporal partitioning would also not divide the voxel-indexed arrays that dominate memory. The natural parallel dimension is the voxel dimension, because the Bloch simulation and several derivative computations are performed independently for each voxel. With voxel partitioning, each GPU stores and computes a subset of the voxels, which divides the voxel-indexed memory and voxel-local work. The consequence is that measurement-domain quantities such as the simulated signal must still be assembled by Allreduce.

3

Overview

3.1 Motivation: Memory Limitation

The practical limitation motivating the multi-GPU design is GPU memory pressure in the single-GPU implementation. In the MR-STAT code, the memory bottleneck comes from the magnetisation-state array `echos` produced by the Bloch simulation and the finite-difference derivative array ∂`echos` used by the Jacobian operations. Here, N is the image width and height of an $N \times N$ slice, K is the number of readouts acquired for each phase-encoding line, and p is the number of MPI ranks, which is also the number of GPUs in this implementation. `echos` has shape $n_{TR} \times n_{\text{loc}}$, where $n_{TR} = N \times K$ is the total number of readouts and $n_{\text{loc}} \approx N^2/p$ is the number of voxels owned by one rank. ∂`echos` has the same readout-by-local-voxel shape, but each entry stores finite-difference derivatives with respect to T_1 and T_2. These arrays dominate memory because each rank stores readout-indexed magnetisation and derivative quantities for its local voxel slice. On a single GPU, $p = 1$, so $n_{\text{loc}} = N^2$ and the memory footprint grows as $O(N^3)$ for fixed K. This growth makes multi-GPU reconstruction necessary for large images, not only faster. The problem-size scaling experiments in Section 6.5 show that one A4000 GPU runs out of memory for $N \geq 384$, and a two-GPU configuration runs out of memory at $N = 512$. Multi-GPU configurations are therefore required to make the larger image reconstructions feasible.

3.2 High-Level Outline

The reconstruction starts from a measured time-domain signal, the pulse sequence, the sampling trajectory, coil sensitivities, voxel coordinates, and an initial estimate of the unknown parameter values. It estimates one set of parameters per voxel, including T_1,

T_2, and the real and imaginary proton-density variables used by the signal model. The multi-GPU reconstruction returns the same maps as the single-GPU reconstruction. On a single GPU, one device holds all of the voxels and runs the entire reconstruction. In the multi-GPU version, the voxels of the image are divided into several subsets, one for each GPU, and each subset is handled by its own MPI rank. Each rank carries out the computation for its own voxels. Communication occurs where measurement-domain data have to be assembled from contributions of all voxels. Each rank computes a partial result, and the ranks combine these partial results into the full quantity at the points where the simulated signal or Hessian-vector computation requires it. Figure 3.1 shows this flow.

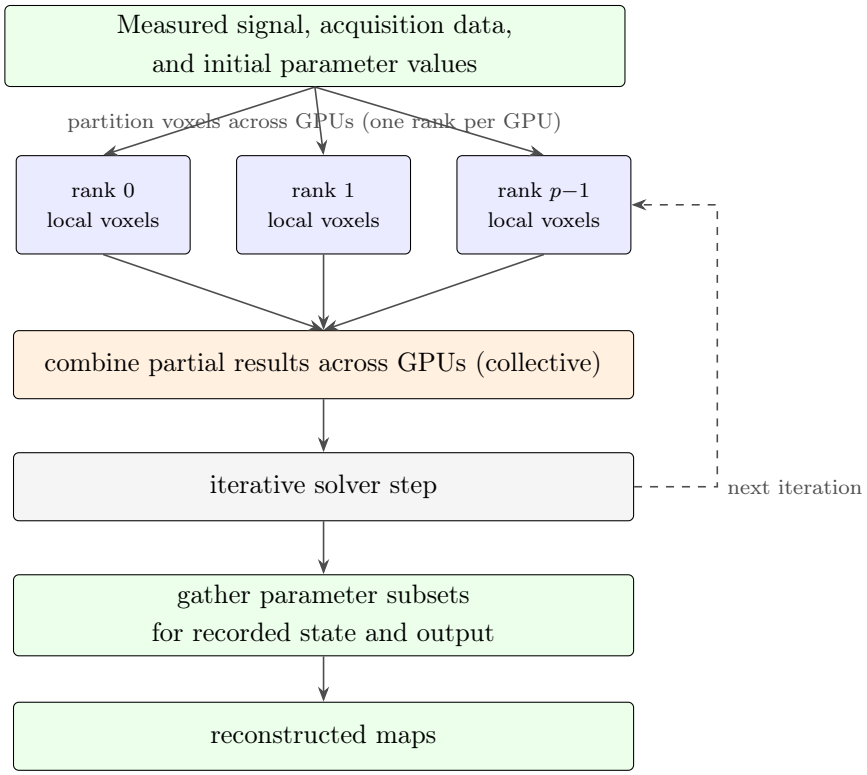


Figure 3.1: High-level outline of the multi-GPU reconstruction. The image voxels are partitioned across the GPUs, one MPI rank per GPU, and each rank computes on its own voxels. Partial measurement-domain results are combined across the GPUs by collective operations when the simulated signal or forward Jacobian-vector product is needed. Parameter subsets are gathered when full maps are needed for recorded state and output.

4

Design

This chapter explains the design for the multi-GPU qMRI reconstruction. It states the design goals(Section 4.1) and reviews the MR-STAT reconstruction algorithm (Section 4.2). Based on the algorithmic structure, the chapter introduces the multi-GPU perspective (Section 4.3), the voxel partitioning strategy (Section 4.4), and the resulting communication cost (Section 4.5).

4.1 Design Goals

The design is constrained by the following goals (Table 4.1). The G2 and G3 are the practical contribution of the work: numerical equivalence with the single-GPU reference, and relief from the single-GPU out-of-memory error that limits the single-GPU implementation at large image dimensions.

Table 4.1: Design Goals

Goal	Statement
G1	The Blcoh Simualtion, MR-STAT forward model, outer trust-region inexact Gauss–Newton iteration, and inner Steihaug-CG solver are inherited unchanged.
G2	Multi-GPU reconstruction is numerically equivalent to the single-GPU reference regardless of GPU count.
G3	Allow image dimensions that single GPU cannot host alone.

4.2 Reconstruction Problem Analysis

The MR-STAT reconstruction problem is a nonlinear inverse problem, where quantitative tissue-parameter maps are estimated from the measured time-domain MRI signal by fitting a forward signal model that uses Bloch simulation (2). In this thesis, the unknowns are the voxel-wise values of T_1, T_2, and the real and imaginary proton-density components ρ^x, ρ^y. These unknowns are concatenated into the parameter vector $\boldsymbol{\alpha}$. For a given $\boldsymbol{\alpha}$, the forward model predicts the MR signal that would be measured for those tissue parameters. The reconstruction minimises the least-squares mismatch between this simulated signal and the measured signal data. Because the forward model includes spatial encoding gradients, spatial localisation and parameter estimation are performed in the same nonlinear optimisation problem, rather than relying on an FFT as a separate spatial-localisation step (2).

Given a current parameter estimate, the forward model computes the simulated signal. The current optimisation variables define the voxel-wise values of T_1, T_2, ρ^x, and ρ^y. The Bloch simulation first produces magnetisation states for each voxel at the readout times. These magnetisation states are then phase-encoded and combined with coil sensitivities to obtain the simulated forward signal.

For Cartesian readouts, the simulated forward signal is represented per coil as a matrix $\mathbf{S} \in \mathbb{C}^{n_s \times n_r}$, where rows correspond to samples within a readout and columns correspond to readouts. For one receive coil, the simulated signal is evaluated as

$$\mathbf{S} = \mathbf{AB}, \qquad \mathbf{A} \in \mathbb{C}^{n_s \times N^2}, \quad \mathbf{B} \in \mathbb{C}^{N^2 \times n_r}.$$

Here $\mathbf{B}$ contains the phase-encoded magnetisation states, with rows corresponding to voxels and columns corresponding to readouts. The matrix $\mathbf{A}$ contains the terms used to evaluate the voxel sum at each sample point: gradient encoding along the readout, T_2 decay, complex proton density, and receive-coil sensitivity (3). The residual $\mathbf{r}(\boldsymbol{\alpha})$ is then formed as the simulated signal at the current parameter vector $\boldsymbol{\alpha}$ minus the measured signal data.

The reconstruction problem is the nonlinear least-squares problem solved by an outer trust-region inexact Gauss–Newton iteration (2). One outer iteration means one pass in which the residual and the derivatives needed for the Gauss–Newton update are evaluated at the current parameter estimate, a trial parameter update is computed, and the step is accepted or rejected while the trust-region radius is adjusted.

Let $\Delta\boldsymbol{\alpha}$ denote the parameter update computed in such an outer iteration. Let $\mathbf{J}$ denote the Jacobian of the residual with respect to the parameter vector. Since the residual

depends nonlinearly on the tissue parameters, Gauss–Newton uses a local linear approximation around the current parameter vector:

$$\mathbf{r}(\boldsymbol{\alpha} + \Delta\boldsymbol{\alpha}) \approx \mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}\Delta\boldsymbol{\alpha}.$$

The Gauss–Newton update is chosen by minimising the squared norm of this linearised residual. Without a trust-region constraint, this gives the linear least-squares subproblem

$$\min_{\Delta\boldsymbol{\alpha}} \frac{1}{2} \left\| \mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}\Delta\boldsymbol{\alpha} \right\|_2^2 .$$

Expanding the squared norm gives

$$\frac{1}{2} \left\| \mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}\Delta\boldsymbol{\alpha} \right\|_2^2 = \frac{1}{2} \left(\mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}\Delta\boldsymbol{\alpha} \right)^{\mathsf{H}} \left(\mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}\Delta\boldsymbol{\alpha} \right).$$

Differentiating with respect to $\Delta\boldsymbol{\alpha}$ gives

$$\mathbf{J}^{\mathsf{H}}\mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}^{\mathsf{H}}\mathbf{J}\Delta\boldsymbol{\alpha}.$$

Setting this derivative to zero and rearranging gives the normal equation

$$\mathbf{J}^{\mathsf{H}}\mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}^{\mathsf{H}}\mathbf{J}\Delta\boldsymbol{\alpha} = 0,$$

$$\mathbf{J}^{\mathsf{H}}\mathbf{J}\Delta\boldsymbol{\alpha} = -\mathbf{J}^{\mathsf{H}}\mathbf{r}(\boldsymbol{\alpha}),$$

where $\mathbf{J}^{\mathsf{H}}\mathbf{r}$ is the gradient term and $\mathbf{J}^{\mathsf{H}}\mathbf{J}$ is the Gauss–Newton approximation of the Hessian.

The trust-region method restricts the size of $\Delta\boldsymbol{\alpha}$ because the quadratic approximation is only reliable near the current parameter estimate. The corresponding trust-region subproblem is

$$\min_{\Delta\boldsymbol{\alpha}} \frac{1}{2} \left\| \mathbf{r}(\boldsymbol{\alpha}) + \mathbf{J}\Delta\boldsymbol{\alpha} \right\|_2^2 \quad \text{subject to} \quad \|\Delta\boldsymbol{\alpha}\|_2 \leq \Delta,$$

where Δ is the trust-region radius.

The Steihaug-CG inner loop approximately solves this trust-region subproblem. An inner iteration means one conjugate-gradient step used to approximate the current parameter update $\Delta\boldsymbol{\alpha}$. It updates the current approximation of $\Delta\boldsymbol{\alpha}$ inside the subproblem, not the MR-STAT tissue maps themselves.

The reconstruction is matrix-free: the Jacobian $\mathbf{J}$ and the Gauss–Newton approximate Hessian $\mathbf{J}^{\mathsf{H}}\mathbf{J}$ are not assembled as explicit matrices. In each inner iteration, the solver only asks for the product of this approximate Hessian with the current CG vector $\mathbf{v}$:

$$\mathbf{J}^{\mathsf{H}}(\mathbf{J}\mathbf{v}).$$

The forward Jacobian product is

$$\mathbf{Jv} = \sum_{j=1}^{N^2} \mathbf{J}_j \mathbf{v}_j,$$

where $\mathbf{v}_j$ contains the entries of $\mathbf{v}$ associated with voxel j. The corresponding Hessian-vector block for voxel j is then

$$(\mathbf{J}^{\mathsf{H}}\mathbf{Jv})_j = \mathbf{J}_j^{\mathsf{H}}(\mathbf{Jv}).$$

The parameter update $\Delta\boldsymbol{\alpha}$ returned by the Steihaug-CG solve is then applied with reflection at the feasible boundaries to enforce the per-voxel box constraints (20). Repeating the outer iterations updates the parameter vector until the max iteration count is reached. The final output is the quantitative tissue maps T_1, T_2, ρ^x, and ρ^y.

4.3 Multi-GPU Perspective

For the multi-GPU design, the voxel-local computations are the source of parallelism, since a rank that owns a subset of voxels can perform them on its assigned GPU using only its local data. The cross-voxel reductions are the unavoidable coupling. They require cross-rank synchronisation in a distributed implementation. Figure 4.1 summarises the cross-rank communication within one outer iteration, showing where MPI collectives occur.

In the multi-GPU design, we evaluate the same sum by partitioning the matrices along the voxel dimension. Partitioning $\mathbf{A}$ column-wise and $\mathbf{B}$ row-wise into p blocks yields the algebraic equivalence

$$\mathbf{S} \;=\; \mathbf{AB} \;=\; \sum_{i=1}^{p} \mathbf{A}_i \mathbf{B}_i,$$

where $\mathbf{A}_i, \mathbf{B}_i$ are rank i's voxel slice. Each rank computes its block product $\mathbf{A}_i\mathbf{B}_i$ from local data alone, and an Allreduce over the p partial products computes the distributed equivalent of $\mathbf{S}$.

In the multi-GPU design, we partition voxels into rank-local voxel subsets $\mathcal{I}_1, \ldots, \mathcal{I}_p$, this becomes

$$\mathbf{Jv} = \sum_{i=1}^{p} \left(\sum_{j \in \mathcal{I}_i} \mathbf{J}_j \mathbf{v}_j \right).$$

Each rank computes the inner sum for its local voxels, and one Allreduce over these partial measurement-domain vectors assembles $\mathbf{Jv}$ on every rank. After this Allreduce, every rank

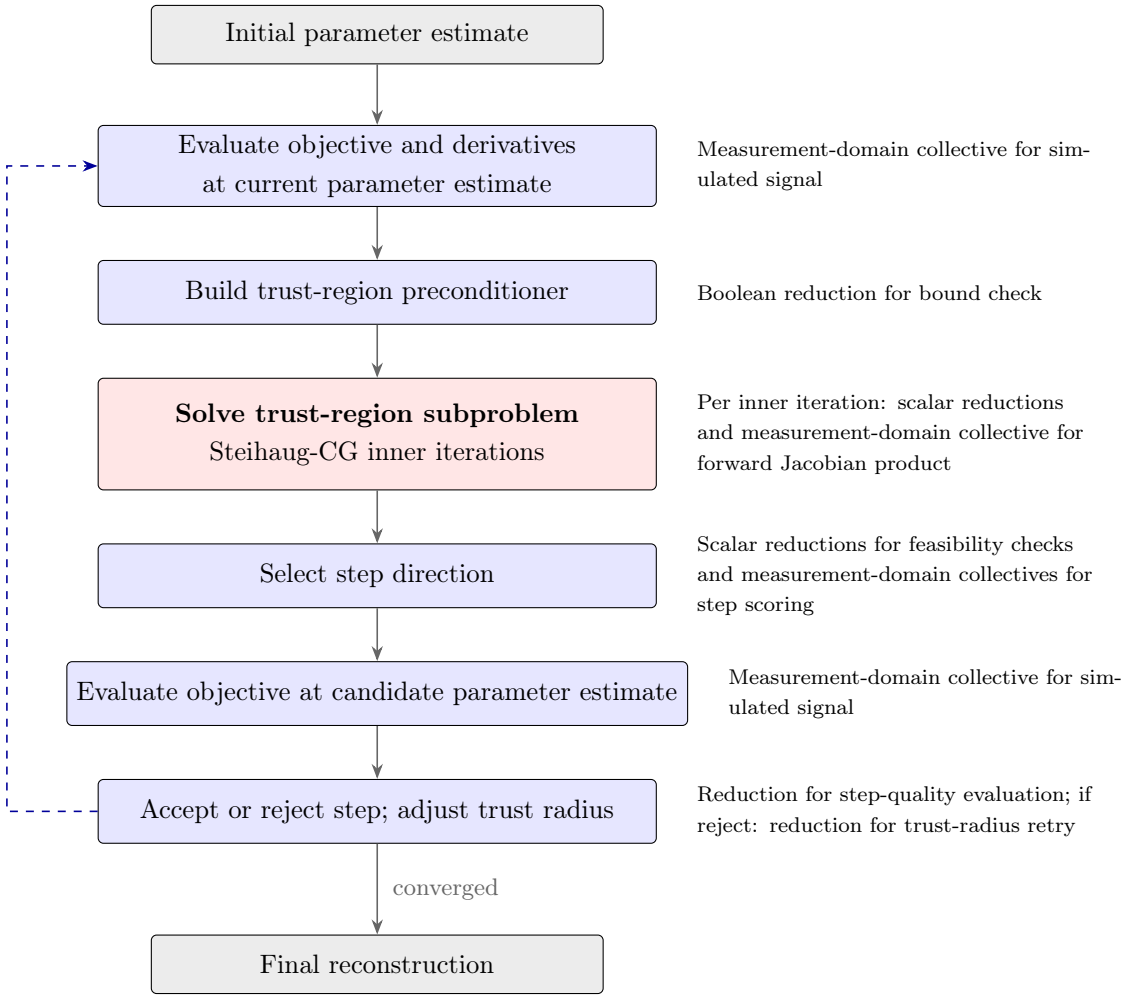


Figure 4.1: Flowchart of the multi-GPU design and its cross-rank communication within one outer iteration. The right-hand annotations list the MPI collectives associated with each phase: evaluating the objective and derivatives at the current parameter estimate assembles the simulated signal by a measurement-domain collective; building the trust-region preconditioner uses a Boolean reduction for the bound check; each Steihaug-CG inner iteration uses scalar reductions and a measurement-domain collective for the forward Jacobian product; step selection uses scalar reductions for feasibility checks and, depending on the step candidates considered, measurement-domain collectives for step scoring; evaluating the candidate parameter estimate again assembles the simulated signal; and step acceptance uses reductions for step-quality evaluation and an trust-radius retry reduction if rejected.

has the same measurement-domain vector $\mathbf{y} = \mathbf{J}\mathbf{v}$. Then the adjoint Jacobian produces independent voxel-indexed output: for voxel j,

$$\left(\mathbf{J}^{\mathsf{H}}\mathbf{y}\right)_j = \mathbf{J}_j^{\mathsf{H}}\mathbf{y}.$$

Therefore, the collective summation that assembles $\mathbf{J}\mathbf{v}$ is the synchronisation point. The subsequent adjoint Jacobian produces only the local part of $\mathbf{J}^{\mathsf{H}}\mathbf{J}\mathbf{v}$.

The same local-then-global pattern also applies when the solver needs scalar quantities over the full parameter vector. Dot products, norms, infinity norms, and minima are first computed on each rank's local voxel slice and then combined into a global scalar. For logical checks such as bound feasibility, each rank first evaluates the condition on its local voxel slice, and the resulting Boolean values are combined globally.

4.4 Voxel Partitioning

Voxel partitioning follows from the structure of the MR-STAT unknowns and computations. The unknown tissue parameters are voxel-wise quantities, and the largest intermediate arrays are also indexed by voxel. In particular, the Bloch simulation output and derivative arrays store readout-indexed magnetisation quantities for each voxel. Splitting the voxel set therefore reduces the dominant per-GPU memory footprint and divides the voxel-local Bloch, derivative, and adjoint Jacobian work across GPUs.

Partitioning by readout samples would not address this bottleneck. The measured signal is indexed by acquisition samples, readouts, and coils, but the expensive state arrays remain indexed by voxel. Moreover, applying the adjoint Jacobian for a local voxel block requires the full residual or the full forward Jacobian-vector result over the measurement domain. For this reason, the design assigns each MPI rank one contiguous voxel range on one GPU. Parameter-domain data remain local to that rank, while measurement-domain sums over voxel contributions are handled by collective summation.

Under this partitioning, the reconstruction data fall into the three classes in Table 4.2.

Table 4.2: Three Data Classes in the Multi-GPU Design

Data class	Meaning	Storage and communication
Parameter-domain data	Voxel-wise quantities, including unknown tissue parameters T_1, T_2, and the real and imaginary proton-density components ρ^x, ρ^y, gradients, and optimisation steps.	Partitioned as rank-local voxel slices. Full tissue parameter maps are assembled only for output.
Measurement-domain data	Arrays on the acquisition grid of readout samples, readouts, and receive coils, including the measured signal, simulated forward signal $\mathbf{S}$, and forward Jacobian-vector product $\mathbf{Jv}$.	The measured signal is replicated across all ranks. $\mathbf{S}$ and $\mathbf{Jv}$ are assembled by collective summation.
Global solver decisions	Checks used by the trust-region solver and Steihaug-CG inner loop, such as norms, inner products, feasibility checks, and step acceptance.	Reduced across ranks to one shared scalar or Boolean value.

4.5 Communication Cost

Figure 4.1 shows where the collective operations appear within one outer iteration. The communication cost has two different forms. Measurement-domain collectives assemble arrays with the same acquisition-grid shape as the simulated signal or the forward Jacobian-vector product. Scalar reductions combine local parameter-domain values into one global scalar or Boolean value.

Objective evaluations require the first form. At the current parameter estimate and at each candidate parameter estimate, every rank computes only the simulated-signal contribution from its local voxel range. A collective summation then assembles the full simulated signal needed to form the residual.

Hessian-vector products introduce the same measurement-domain collective inside the Steihaug-CG loop. For each Hessian-vector product, the forward Jacobian product Jv is first assembled from voxel-local contributions. After this Jv vector is available on every rank, the adjoint Jacobian computation returns only the local parameter-domain slice. Because Steihaug-CG may perform several inner iterations within one outer iteration, this repeated Hessian-vector communication is the main communication cost expected from the design. Section 6.4.2 evaluates this expectation that Steihaug-CG inner loop dominates

the per-outer-iteration runtime.

The solver also uses scalar reductions for inner products, norms, feasibility checks, trust-region checks, and step-acceptance decisions. These reductions move little data compared with the measurement-domain collectives, but they still introduce synchronisation points in the solver loop.

5

Implementation

5.1 Code Dependencies

The experiment hardware environment and setup used for evaluation is described in Section 6.1. This section only records the software dependencies required by the implementation.

The implementation builds on the `MR-STAT.jl` (3). The `MR-STAT.jl` codebase provides the original reconstruction structure, including the objective function, derivative and Jacobian operations, trust-region solver, and utilities for simulation data. The multi-GPU implementation preserves this algorithmic structure, but implements multi-GPU versions of the objective, trust-region solver, and Steihaug-CG operations for partitioned parameter vectors and MPI collectives. The local Bloch simulation remains the main numerical kernel used in the multi-GPU implementation. `MPI.jl` provides rank management and collective operations. `CUDA.jl` is used for GPU execution and for selecting the GPU device used by each rank. `ComputationalResources.jl` provides the resource dispatch interface. `BlochSimulators.jl` provides the Bloch simulation and signal evaluation based on trajectories used to form the simulated MR signal. `LinearMaps.jl` is used to pass the matrix-free Hessian-vector product to the solver without assembling a Hessian matrix. `ImagePhantoms.jl` provides the digital phantom used to generate the synthetic MR signal data used as the observed data in the reconstruction. `JLD2.jl` is used by the entry script to save reconstruction outputs and metadata.

5.2 Code Structure

The multi-GPU implementation is in `mrstat_main/src/mpi`. These files add code specific to MPI around the original MR-STAT implementation without replacing the Bloch simulation kernels. The entry script `scripts/mpi_mrstat_recon.jl` loads these files before setting up the multi-GPU reconstruction.

- `MPIResources.jl` defines GPU selection for each rank, voxel partitioning, collective summation, scalar and Boolean reductions, vector gather, and functions that copy GPU arrays to host memory before calling MPI collectives.
- `mpi_objective.jl` implements the multi-GPU objective evaluation, including local Bloch simulation calls, signal reduction, local gradient construction, and the matrix-free Hessian-vector operator.
- `mpi_solver.jl` implements the outer trust-region inexact Gauss–Newton loop for parameter-domain vectors stored on each rank and records the timing breakdown for each iteration used in the evaluation.
- `mpi_steihaug.jl` implements the Steihaug-CG inner loop for vectors stored on each rank, replacing global dot products and norms with local computations followed by MPI reductions.
- `mpi_utils.jl` computes boundary distances, feasibility checks, quadratic step scores, and step selection over partitioned vectors.

5.3 Data Representation

Figure 5.1 shows how the implementation stores measurement-domain and parameter-domain data. The figure focuses on the two main computations where local voxel contributions must be combined across ranks. When computing the simulated signal, each rank forms the contribution from its voxel range and collective summation assembles the full simulated signal. In the Hessian-vector computation, each rank forms its local forward Jacobian-vector contribution, and collective summation assembles the full forward Jacobian-vector product. The local adjoint Jacobian then returns entries for that rank's voxels.

The measured signal is replicated rather than partitioned because each rank needs the full residual when applying the adjoint Jacobian to its local voxels. Partitioning the measured

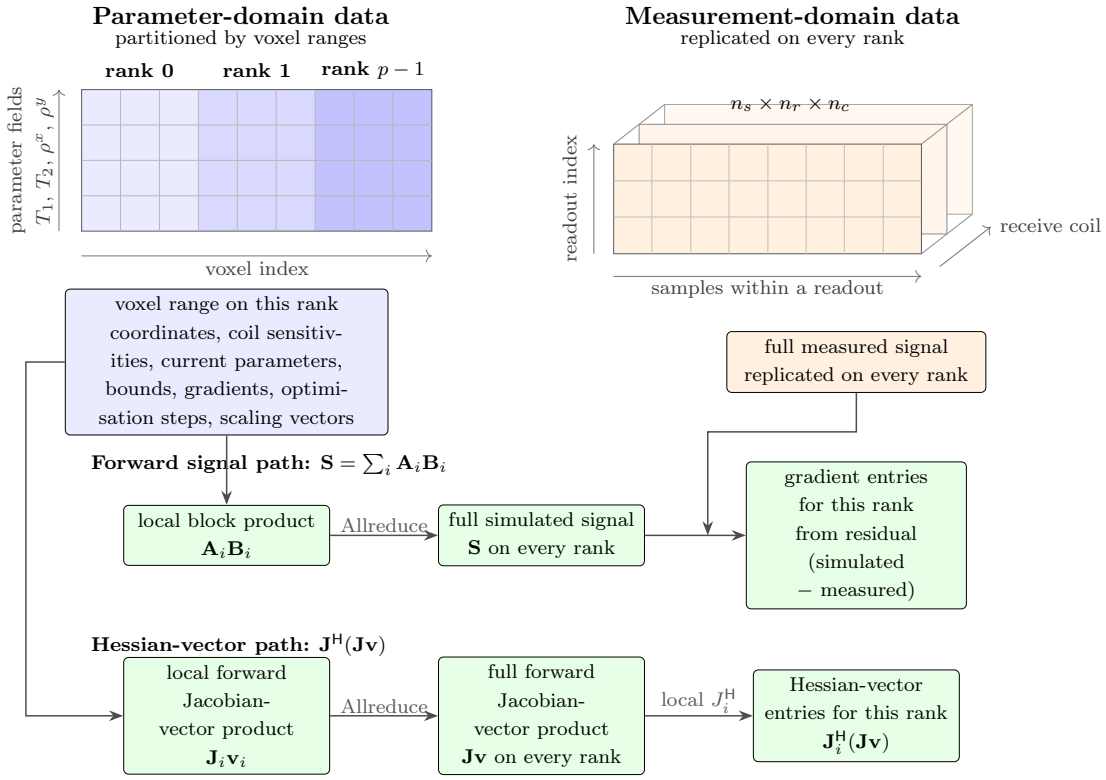


Figure 5.1: Data representation in the multi-GPU implementation. The figure focuses on the forward signal path and the Hessian-vector path because these are the main reconstruction computations where voxel contributions from different ranks must be combined. Here, i denotes the voxel block owned by rank i. n_s, n_r, and n_c denote the number of samples per readout, readouts, and receive coils, respectively. $\mathbf{S}$ is the assembled simulated forward signal. $\mathbf{A}_i$ is the encoding block for the voxel block on rank i. $\mathbf{B}_i$ contains the magnetisation states over the readouts for that voxel block. $\mathbf{J}$ is the Jacobian of the residual with respect to the voxel-wise optimisation variables, namely log-scaled T_1, log-scaled T_2, and the real and imaginary proton-density components ρ^x, ρ^y. $\mathbf{J}_i$ is the Jacobian block associated with the voxel block on rank i. $\mathbf{v}$ is the input parameter-space vector for one Hessian-vector product evaluation inside the inner Steihaug-CG solve. $\mathbf{v}_i$ contains the entries of $\mathbf{v}$ for the voxel block on rank i. $\mathbf{J}_i^{\mathsf{H}}$ is the adjoint of that local Jacobian block; it maps a measurement-domain vector back to the parameter entries for rank i's voxel block. Parameter-domain data are partitioned by contiguous voxel ranges, while measurement-domain data are replicated on every rank. In the forward signal path, each rank computes its local contribution to the simulated signal, the contributions are summed across ranks, and the summed simulated signal is compared with the replicated measured signal to form the residual. The residual is then used with local derivative information to compute the gradient entries for that rank through the adjoint Jacobian. In the Hessian-vector path, each rank computes its local contribution to $\mathbf{J}\mathbf{v}$, the contributions are summed across ranks, and the local adjoint Jacobian then produces the Hessian-vector entries for the rank's voxels.

signal by readout samples would therefore require reconstructing the full residual on every rank. The replication cost can be estimated directly from the shape of the measured signal. The measured signal has shape $n_s \times n_r \times n_c$, where $n_s = N$ is the number of samples per readout, $n_r = n_{TR} = N \times K$ is the number of readouts, and n_c is the number of receive coils. This gives $N^2 \times K \times n_c$ complex entries, which is approximately 2 MB for $N = 224$, $K = 5$, and one receive coil, and approximately 10 MB for $N = 512$. Compared with the 16 GB memory of the A4000 GPU used in the experiments, this replicated measured signal is negligible.

5.4 Signal, Gradient, and Hessian-Vector Computation

`mpi_objective.jl` is the implementation layer for the signal, residual, gradient, and Hessian-vector computations described in Section 4.2.

For the simulated signal, Section 4.3 describes the signal as a sum of rank-local voxel-block contributions. In `mpi_objective.jl`, each rank computes only its local contribution to this sum. `allreduce_sum!` from `MPIResources.jl` then assembles the full simulated signal on every rank. The measured signal is replicated, as discussed in Section 5.3. Each rank can therefore subtract the measured signal locally to form the residual. The least-squares misfit value is computed from this residual.

For the gradient, `mpi_objective.jl` uses the finite-difference derivative and runs it only for the rank's voxel range. The adjoint Jacobian combines the full residual with local derivative information, local magnetisation, coordinates, and coil sensitivities. It returns only the gradient entries for the rank's voxel range. This follows the local adjoint-Jacobian structure described in Section 4.3. No full gradient vector is gathered. Solver then use this local gradient together with local parameter, scaling, and step vectors. `MPIResources.jl` provides `mpi_dot` and `mpi_norm`. `mpi_dot` computes the local dot product and then sums the scalar contributions across ranks. `mpi_norm` uses `mpi_dot`, so the norm is also computed from a global scalar reduction.

For the Hessian-vector, `mpi_objective.jl` returns a `LinearMap`. This implements the partition-and-reduce Hessian-vector structure described in Section 4.3. The `LinearMap` is the interface used by `mpi_steihaug.jl` when the Steihaug-CG loop needs the Gauss–Newton Hessian approximation applied to a vector. The `LinearMap` avoids explicitly storing the Hessian matrix in memory. When the map is applied, the input vector contains only the entries for the rank's voxel range. Cross-rank communication occurs at the forward Jacobian-vector reduction described in Section 4.3. Then `allreduce_sum!` sums the

rank-local forward Jacobian-vector contributions into the full measurement-domain vector. The `LinearMap` then applies the local adjoint Jacobian. The returned vector contains only the Hessian-vector entries for the rank's voxel range. From the solver's point of view, the map takes a local vector and returns a local vector.

5.5 Trust-Region Solver with Partitioned Parameter Vectors

`mpi_solver.jl` implements the outer trust-region inexact Gauss–Newton loop described in Section 4.2 over partitioned parameter-domain vectors. The parameter vector, gradient, bounds, scaling vectors, and step vectors are stored only for the rank's voxel range. Global scalar or Boolean values are computed from local contributions and reduced by the primitives in `MPIResources.jl`.

At the start of an outer iteration, `mpi_solver.jl` calls the objective computation from Section 5.4. The call uses the current local parameter vector. It returns the residual and misfit value on every rank. It also returns the gradient entries for the rank's voxel range and a Hessian-vector `LinearMap`. `mpi_solver.jl` builds the trust-region scaling terms from local parameter entries, local gradient entries, and local bounds. It calls `mpi_computeDistanceToBoundaries` to compute the distance from the current local parameter entries to their feasible bounds. `mpi_steihaug.jl` runs the Steihaug-CG inner loop on local vectors. When a CG step reaches the trust-region boundary, it calls `mpi_distanceToTrustRegion` to compute the boundary step length from global dot products. `MPIResources.jl` defines the global scalar and Boolean reduction functions `allreduce_sum`, `mpi_dot`, `mpi_norm`, `mpi_norminf`, `mpi_minimum`, `mpi_all`, `mpi_any`, and `mpi_colnorms`. These functions first compute the value from the rank's local vector entries. They then use MPI collectives to return the corresponding global scalar or Boolean value on every rank. This implements the scalar and Boolean reduction pattern described in Section 4.3. When Steihaug-CG applies the Hessian approximation, it calls the `LinearMap` returned by `mpi_objective.jl`.

Step selection is implemented in `mpi_utils.jl`. After Steihaug-CG has produced the local entries of a scaled step, `mpi_solver.jl` maps this step back to the original parameter scaling. `mpi_chooseStep` then calls `mpi_withinBounds` to check whether the tentative parameter vector stays within the bounds.

If the update is feasible, this inexact Newton step is kept. The trust-region constraint has already been enforced by the Steihaug-CG solve.

If the update is not feasible, the step-selection code constructs three alternatives. (i) A shorter inexact Newton step goes in the same direction but stops inside the feasible bounds. (ii) A reflected Newton step first goes to the boundary and then continues along a reflected direction. (iii) A steepest descent step goes along the negative gradient direction. Each rank stores only the entries of these candidate steps for its own voxel range. The final choice is based on global scores.

`mpi_distanceToFeasibleRegion` computes the largest scalar multiplier that keeps the current step inside the parameter bounds. For each local parameter entry, it computes how much the current step can be scaled before that entry would move outside its lower or upper bound. `mpi_minimum` then reduces to the largest scalar multiplier that keeps all parameter entries inside their bounds. Entries whose scaling factor equals the largest scalar multiplier are marked as boundary entries. When the inexact Newton step is outside the bounds, the code multiplies the step by the largest scalar multiplier. After this multiplication, at least one parameter entry is at its bound and no entry is outside its bounds. The marked boundary entries also define which components are reversed in the reflected Newton direction.

For the reflected Newton step, `mpi_distanceToTrustRegion2` computes the trust-region limit for the reflected direction. The largest allowed reflected step length is limited by two distances: the parameter-bound distance and the trust-region distance. The smaller of these two distances defines the largest allowed reflected step length.

For the steepest descent step, `mpi_norm` computes the length of the steepest descent direction. The maximum trust-region step length is obtained by dividing the trust-region radius by the direction length. The largest allowed steepest descent step length is also limited by the parameter-bound distance and the trust-region distance. The smaller of these two distances defines the largest allowed steepest descent step length.

`mpi_evaluateQuadratic` is used when every entry of the candidate step vector has already been determined. It computes the quadratic score from the gradient term and the Hessian term. The Hessian term requires applying the Hessian-vector `LinearMap` to the candidate step. The gradient term and the Hessian term are both inner products over partitioned parameter-domain vectors, so they are computed with MPI dot products, as described in Section 5.4.

For the reflected Newton step and the steepest descent step, the code knows the direction but still has to choose the final scalar step length. `mpi_buildQuadratic1D` writes the quadratic model as a function of this scalar step length. `minimizeQuadratic1D` then

chooses the step length that gives the smallest quadratic model value while staying within the parameter bounds and the trust-region radius.

`mpi_chooseStep` then compares the quadratic scores of the shorter Newton step, the reflected Newton step, and the steepest descent step. It selects the step with the smallest score.

After a step is chosen, `mpi_solver.jl` forms the local entries of the candidate parameter vector. It then calls the signal, residual, and misfit path from Section 5.4. This recomputes the simulated signal for the candidate parameter vector. Collective summation assembles the full simulated signal. Each rank forms the residual and computes the new least-squares misfit value.

`mpi_solver.jl` accepts or rejects the step by comparing the real decrease with the decrease predicted by the quadratic approximation. The real decrease is the old misfit value minus the new misfit value. The predicted decrease is computed from the gradient, the step, and the Hessian-vector result. The required inner products are reduced across ranks.

If the real decrease is sufficient relative to the predicted decrease, the local parameter vector is updated. After an accepted step, `mpi_adjustTrustRadius` updates the trust-region radius. It increases the radius when the prediction is good and the accepted step length is close to the current trust-region radius.

If the real decrease is too small, the trust radius is decreased. The solver then checks the intermediate steps stored during the Steihaug-CG solve. If one of these CG steps fits inside the reduced trust radius, it is reused without rerunning Steihaug-CG. If no stored step is suitable, the solver stays in the same outer iteration and reruns Steihaug-CG with the smaller trust-region radius. The trust radius is not reduced indefinitely. The implementation sets a lower threshold Δ_{limit} to 10^{-10} times the initial trust-region radius. If the radius falls below this threshold, the current step is accepted to prevent from continuing indefinitely.

5.6 Evaluation Outputs

5.6.1 Result Files

Reconstruction results and intermediate outputs are saved to a structured JLD2 result file. The JLD2 file stores the solver state, the ground truth phantom, the image size, the transmit field, and run metadata. The solver state contains the parameter estimates at the initial point and after each accepted outer iteration. It also contains the misfit

value, residual vector, and elapsed time recorded at those accepted iterations. For noisy results, this file additionally stores the noise floor value. The stored parameter estimates and ground truth are used for the parameter map in figures 6.1–6.2 and table 6.2. For each accepted iteration, the stored parameter vector is compared with the ground truth T_1 and T_2 maps inside the tissue ROI mask to compute the root-mean-square relative error (RMSRE) curves in figures 6.3 and 6.4. They are also used to compute the T_n-to-noise-ratio (T_nNR) efficiency curves in figures 6.6 and 6.7. The stored misfit values are used for the cost function convergence curves in figure 6.5. The stored elapsed times are used for the strong-scaling wall-time, speedup, and efficiency figures in figures 6.8–6.10. They are also used for the problem-size-scaling figures in figures 6.12–6.14.

Rank 0 also writes the timing CSV during the reconstruction. These timing records are used to produce the per-iteration time breakdown in figure 6.11. The figure groups per-iteration times into three most time-consuming stages: objective evaluation, Steihaug-CG, and step selection. The objective evaluation category includes both the objective evaluation at the current parameter vector and the candidate parameter evaluation.

5.6.2 Noiseless and Noisy Result Analysis

The result analysis is conducted for both noiseless and noisy data. The noiseless analysis uses synthetic raw data generated from the digital phantom without added thermal noise. The noisy analysis starts from the same synthetic raw data and adds complex Gaussian noise to the simulated data with $\mathrm{SNR}_{\mathrm{dB}} = 15.36$, following the Cartesian experiment in the MR-STAT efficiency and robustness study (28). Let $\boldsymbol{\eta}$ denote the noise vector. The cost function curve uses the least-squares misfit. Therefore, the noise floor $\frac{1}{2}\|\boldsymbol{\eta}\|^2$ is plotted as a horizontal line in the cost function convergence curves. If the noisy cost curve drops below this level, the reconstruction is fitting the noise. If it stays higher, the optimisation has not reached the data fidelity.

5.6.3 Reconstruction Quality Analysis

The standard Shepp–Logan phantom mathematically models the human head using an outer boundary ellipse and 9 internal ellipses, which define a total of 10 structural components as 10 different brain regions. Each brain region Ω_m contains voxels with the same Shepp–Logan phantom value. These values are used to group voxels into different brain regions. The Shepp–Logan value 0 denotes image background and is excluded from the tissue mask. For each brain region, `make_phantom.jl` samples one T_1, T_2, ρ^x, and ρ^y value

from preset ranges and assigns to all voxels in that region. The overall tissue mask is $\Omega = \bigcup_{m=1}^{M} \Omega_m$ with background voxels excluded. The ground-truth parameter maps are generated.

Let $q_j^{(k)}$ be the reconstructed value at voxel j and iteration k, and let q_j^{gt} be the ground truth value. The root-mean-square relative error (RMSRE) over voxels in the overall tissue mask Ω at iteration k is

$$\mathrm{RMSRE}^{(k)} = \sqrt{\frac{1}{|\Omega|} \sum_{j \in \Omega} \left(\frac{q_j^{(k)} - q_j^{\mathrm{gt}}}{q_j^{\mathrm{gt}}} \right)^2}.$$

The mean of the absolute relative error (MARE) over voxels in the overall tissue mask Ω at iteration k is

$$\mathrm{MARE}^{(k)} = \frac{1}{|\Omega|} \sum_{j \in \Omega} \left| \frac{q_j^{(k)} - q_j^{\mathrm{gt}}}{q_j^{\mathrm{gt}}} \right|.$$

RMSRE is used for the iteration curves, which show how the reconstruction error changes during optimisation. The optimal iteration is identified as the iteration with the lowest RMSRE and is marked on the iteration curves. For noisy data, this mark identifies the point after which later iterations may overfit the measurement noise. In that case, the data misfit can still decrease while the parameter error increases. MARE is reported at the final iteration. If iteration curves are close across GPU configurations, this means that the multi-GPU distribution preserves reconstruction accuracy.

For each of the tissue regions $\Omega_1, \ldots, \Omega_M$, let $\mu_m^{(k)}$ and $\sigma_m^{(k)}$ denote the mean and standard deviation of the reconstructed values over voxels in the region m, the T_n-to-noise ratio (T_nNR) of region m is computed as $\mu_m^{(k)}/\sigma_m^{(k)}$. The T_nNR efficiency averages this ratio over the overall tissue regions and divides by $\sqrt{T_{\mathrm{scan}}}$:

$$E_{T_n}^{(k)} = \frac{1}{\sqrt{T_{\mathrm{scan}}}} \frac{1}{M} \sum_{m=1}^{M} \frac{\mu_m^{(k)}}{\sigma_m^{(k)}}, \qquad T_{\mathrm{scan}} = 11.2 \text{ s}.$$

T_{scan} is computed from 1120 TR periods of 10 ms each. The T_nNR efficiency compares Cartesian and radial trajectories in the MR-STAT efficiency and robustness study (28). In this thesis, the acquisition trajectory is fixed to the Cartesian case. The T_nNR efficiency is used as a consistency check. If the single-GPU and multi-GPU configurations produce overlapping T_nNR efficiency curves, this indicates that the multi-GPU implementation preserves the same reconstruction efficiency.

The parameter map is used to compare the single-GPU baseline with the four-GPU and eight-GPU configurations. For the noiseless data, we show the final iteration because

the RMSRE continues to decrease until the end of the reconstruction. For the noisy data, we show both the optimal and final iterations because final iteration can overfit the measurement noise.

5.6.4 Strong Scaling and Problem-Size Scaling Analysis

Strong scaling is evaluated at a fixed image size of $N = 224$. Let p denote the number of GPUs and t_p denote the wall time measured with p GPUs. Speedup $S(p)$ and parallel efficiency $E(p)$ relative to the single-GPU wall time t_1 are

$$S(p) = \frac{t_1}{t_p}, \qquad E(p) = \frac{S(p)}{p}.$$

The mean per-iteration time across outer iterations is summarized for each GPU configuration. The breakdown is grouped into the three most time-consuming stages: objective evaluation, Steihaug-CG, and step selection.

Furthermore, the problem-size scaling analysis evaluates how reconstruction time and scalability change with the image dimension N. The single-GPU baseline is available only for $N \leq 320$. For larger image sizes, the single-GPU reconstruction exceeds GPU memory. The speedup and efficiency analyses for problem-size scaling use one node with two GPUs as the baseline.

Weak scaling is not evaluated because increasing the problem size also increases the measurement-domain communication cost, as discussed in Section 8.3.

6

Evaluation

This chapter evaluates the multi-GPU reconstruction through four parts. The first part (Section 6.2) verifies that the multi-GPU reconstruction result is consistent with the ground truth. The second part (Section 6.3) compares convergence behaviour under noiseless and noisy data. The third part (Section 6.4) measures how reconstruction wall-time, speedup, and parallel efficiency scale with GPU count at a fixed problem size. The fourth part (Section 6.5) examines how reconstruction wall-time, speedup, and parallel efficiency scale with problem size, motivating the use of multiple GPUs for large reconstructions that exceed single-GPU memory.

6.1 Experimental Setup

This section describes the hardware, software, numerical phantom data, and reconstruction configurations shared by the evaluation.

6.1.1 Hardware and Software Environment

All experiments are conducted on the DAS-6 cluster (29). We use compute nodes equipped with NVIDIA RTX A4000 GPUs. The software stack is Julia 1.10.3, CUDA toolkit 12.3, and OpenMPI 4.1.6. For single-node runs, MPI is configured to use POSIX shared memory (`UCX_TLS=posix,sm,self`). For multi-node runs, MPI is configured to use TCP transport (`UCX_TLS=tcp,self,sm`, `-mca pml ob1 -mca btl tcp,self`).

6.1.2 Numerical Phantom

We use a synthetic numerical phantom based on the Shepp-Logan head model (30). The model defines nine tissue regions, each a set of voxels that share one tissue label. For

each region, T_1 and T_2 values are drawn independently from $[0.3, 2.5]$ s and $[0.03, 0.2]$ s respectively, with T_2 capped at $0.5\,T_1$ wherever it would otherwise exceed T_1. A fixed pseudo-random seed is set before phantom generation, so that the ground-truth T_1 and T_2 maps are reproducible across all experiments and GPU configurations.

6.1.3 Acquisition and Signal Simulation

A 22.4×22.4 cm^2 field of view (FOV) with a 224×224 matrix size ($N = 224$) gives an in-plane resolution of 1 mm $\times$ 1 mm. The acquisition uses a FISP (Fast Imaging with Steady-state Precession) sequence, implemented as `FISP2D` in `BlochSimulators.jl`. The sequence parameters are repetition time TR = 10 ms, echo time TE = 6 ms, and inversion time TI = 25 ms. The flip-angle train increases linearly from 1° to 90° over the acquisition. The EPG simulation retains 32 configuration states. The Cartesian trajectory uses $K = 5$ repetitions per phase-encoding line, giving $N \cdot K = 1120$ readouts. For experiments that operate on noisy data, complex Gaussian noise is added to the simulated k-space samples to achieve $\mathrm{SNR}_{\mathrm{dB}} = 15.36$ (28). A fixed noise seed is shared across all MPI ranks, so that every rank observes the same noise. All reconstructions use the same initial parameter values, with $T_1 = 1.0$ s, $T_2 = 0.100$ s, $\rho^x = 1.0$, and $\rho^y = 0.0$ in every voxel. The box constraints are $T_1 \in [0.1, 7.0]$ s and $T_2 \in [0.001, 3.0]$ s.

6.1.4 Configurations Compared

Different GPU configurations are compared in the experiments (Table 6.1).

Table 6.1: GPU configurations evaluated in this chapter.

Label	Description	Nodes	GPUs/node	Total GPUs	Comm.
1 GPU	single-GPU baseline	1	1	1	—
1N×2G	1 node, 2 GPUs	1	2	2	shared memory
1N×4G	1 node, 4 GPUs	1	4	4	shared memory
2N×2G	2 nodes, 2 GPUs each	2	2	4	TCP
2N×3G	2 nodes, 3 GPUs each	2	3	6	TCP
2N×4G	2 nodes, 4 GPUs each	2	4	8	TCP

Apart from the single-GPU baseline, the scaling experiments in Sections 6.4 and 6.5 use only configurations with an even number of GPUs on each node, which gives the 1, 2, 4, and 8 GPU progression. This follows common practice for balanced parallel execution. The 2N×3G configuration, with three GPUs on each node, has an odd count and is excluded

from these scaling results. It is still included in the equivalence check of Section 6.2 and the convergence analysis of Section 6.3, which do not depend on parallel performance.

6.2 Reconstruction Validation

We verify that the multi-GPU implementation produces reconstructions equivalent to the single-GPU reference before using it for the scaling experiments. We check this equivalence in two cases, first on noiseless data where differences between GPU configurations isolate numerical and implementation effects, and then on noise-corrupted data that matches the real scenario of the scaling experiments.

6.2.1 Equivalence on Noiseless Data

Without measurement noise, all configurations receive the same deterministic synthetic data. Differences between GPU configurations would therefore indicate floating-point round-off or an error in the multi-GPU implementation, not a difference in the input data. The comparison against the ground truth checks whether the shared reconstruction problem is solved accurately.

Figure 6.1 shows the reconstructed T_1 and T_2 maps for the 1-GPU, 4-GPU, and 8-GPU configurations alongside the ground truth and voxel-wise error maps. All configurations reproduce the ground-truth maps with no visually discernible difference.

Table 6.2 quantifies the visual agreement by reporting the MARE of the reconstructed T_1 and T_2 maps for each GPU configuration. All T_1 values stay below 0.005 % and all T_2 values below 0.024 %. These errors are on the order of 10^{-5} to 10^{-4} in relative units, which is consistent with single-precision floating-point round-off. The values do not increase systematically with GPU count, the expected result of random round-off rather than a bias introduced by partitioning or reduction.

Table 6.2: Mean absolute relative error (MARE, percent) of the reconstructed T_1 and T_2 maps against the ground-truth phantom for each GPU configuration on noiseless data. All values fall within the single-precision floating-point round-off regime.

Configuration	T_1 MARE (%)	T_2 MARE (%)
Single GPU	0.0040	0.0174
1N×2G (2 GPUs)	0.0042	0.0196
1N×4G (4 GPUs)	0.0034	0.0171
2N×3G (6 GPUs)	0.0039	0.0198
2N×4G (8 GPUs)	0.0045	0.0236

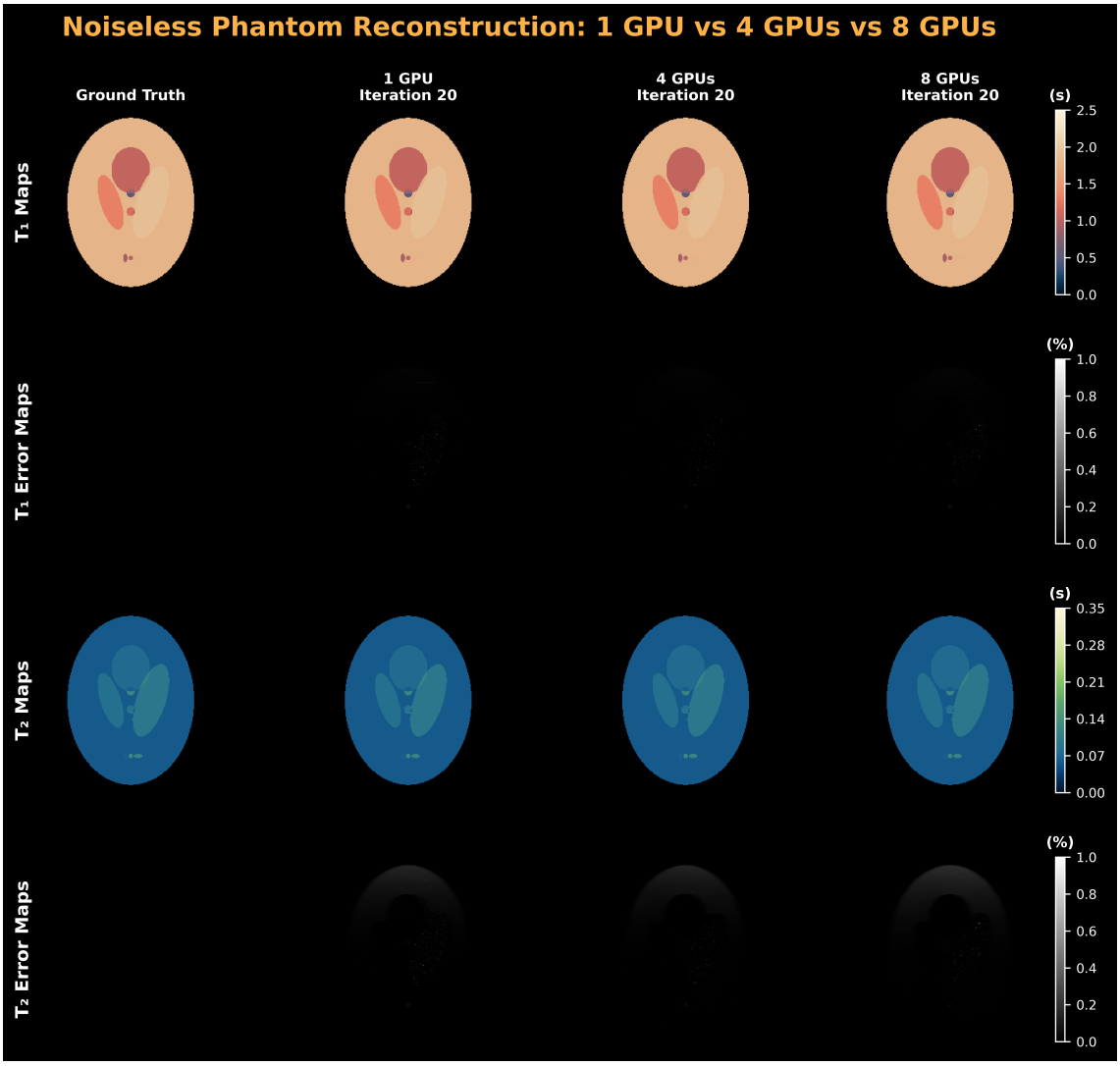


Figure 6.1: Reconstructed parameter maps from noiseless phantom data across GPU configurations. The single-GPU baseline and each multi-GPU MPI configuration produce visually identical T_1 and T_2 maps. Error maps show the voxel-wise absolute relative error computed against the ground-truth phantom.

6.2.2 Reconstruction Quality under Noise

We repeat the comparison on noise-corrupted data ($\mathrm{SNR}_{\mathrm{dB}} = 15.36$) to confirm that the equivalence persists under measurement noise. Figure 6.2 shows the reconstructed T_1 and T_2 maps for the 1-GPU, 4-GPU, and 8-GPU configurations. For noisy data, the final iteration is not the most accurate parameter estimate. As discussed in Section 5.6.3, later iterations can continue reducing the signal misfit while fitting measurement noise, which can increase the parameter-map error. The figure therefore shows both the optimal iteration and the final iteration. The optimal iteration is the accepted iteration with the lowest RMSRE, selected separately for T_1 and T_2. The reconstructed maps show similar tissue boundaries, contrast, and noise patterns across the three GPU configurations. Together, the noiseless and noisy validation results show that the GPU configurations produce equivalent reconstructions for the data used in the scaling experiments.

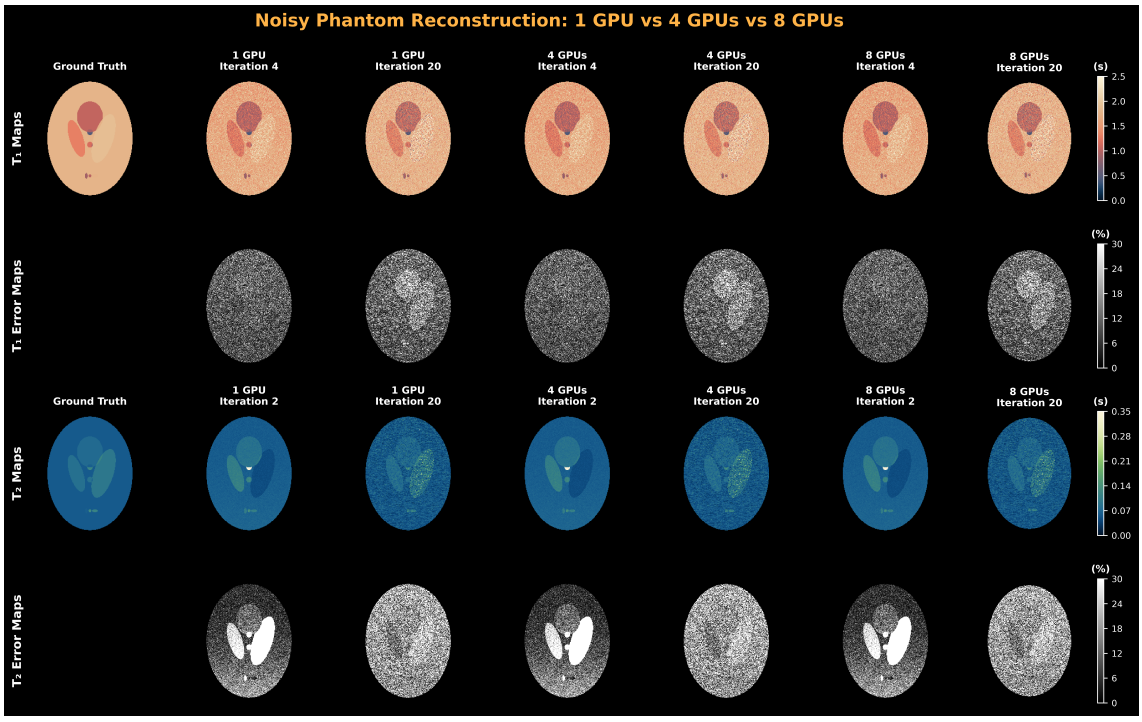


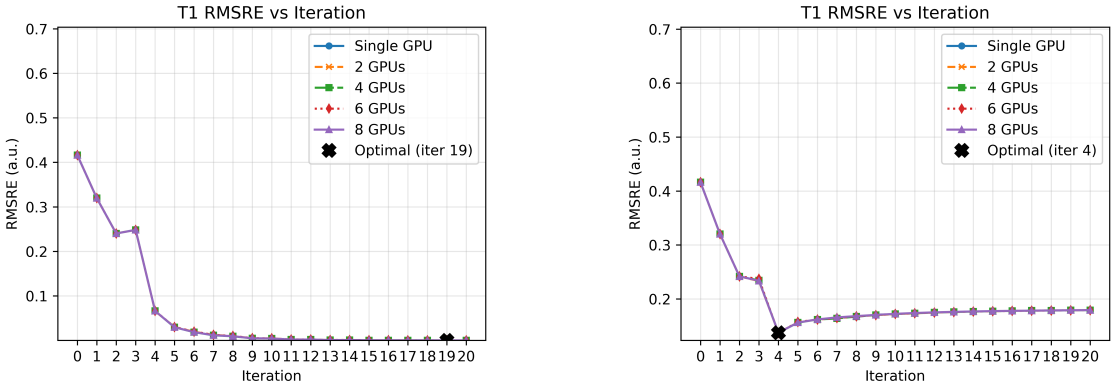
Figure 6.2: Reconstructed T_1 and T_2 maps from noise-corrupted phantom data ($\mathrm{SNR}_{\mathrm{dB}} = 15.36$) for representative GPU configurations. Both optimal and final iterations are shown because noisy data can lead to overfitting at later iterations. Differences against ground truth are noise-limited and consistent across configurations.

6.3 Convergence Behaviour

The convergence behaviour of the optimisation differs substantially between noiseless and noisy data. The curves are computed from the saved parameter estimates, misfit values, ground-truth maps, and noise floor described in Section 5.6.1. The following convergence curves are computed as described in Section 5.6.3.

6.3.1 RMSRE Convergence

Figures 6.3 and 6.4 show the root-mean-square relative error (RMSRE) of the T_1 and T_2 maps against the ground-truth phantom.

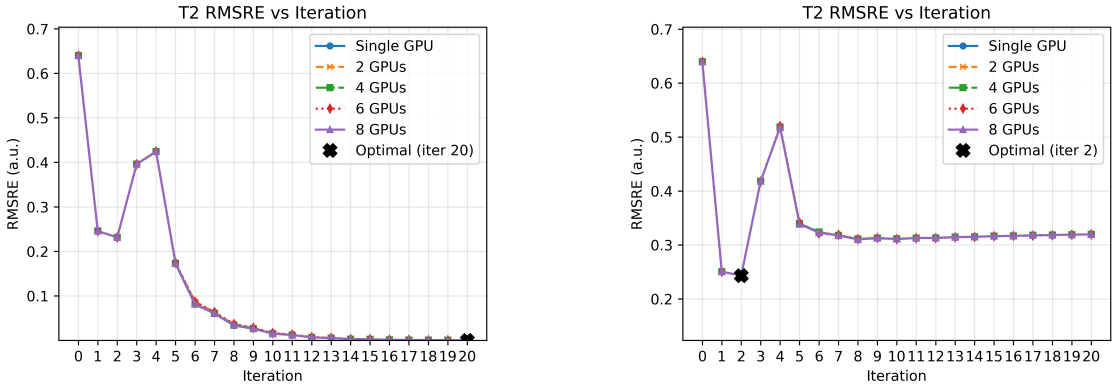


(a) Noiseless data

(b) Noisy data

Figure 6.3: T_1 root-mean-square relative error versus iteration. Noiseless reconstructions approach zero. Noisy reconstructions reach an early optimum and then show overfitting.

Noiseless reconstructions approach RMSRE near zero. Noisy reconstructions reach their lowest RMSRE early and then become worse as later iterations fit the noise. The optimal iteration is the iteration with the lowest RMSRE. For the noisy case, this occurs at iteration 4 for T_1 and iteration 2 for T_2. Across GPU configurations, the curves overlap showing numerical equivalence over the full optimisation.



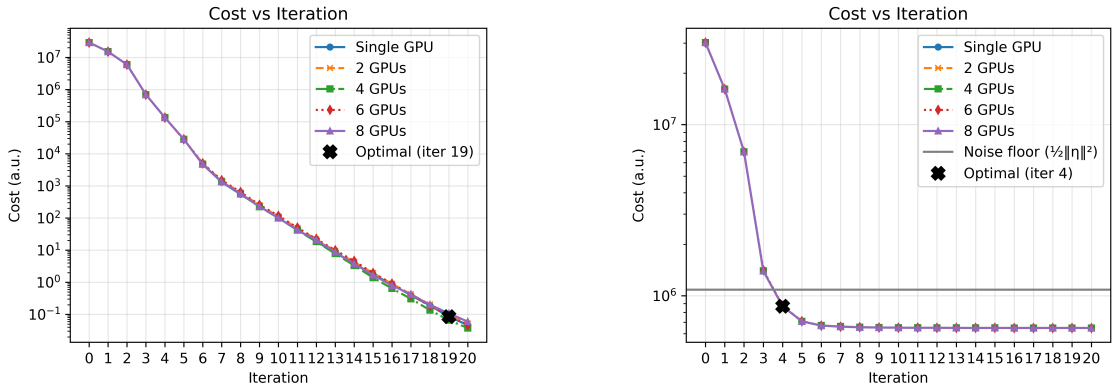
(a) Noiseless data

(b) Noisy data

Figure 6.4: T_2 root-mean-square relative error versus iteration. Noiseless reconstructions approach zero. Noisy reconstructions reach an early optimum and then show overfitting.

6.3.2 Cost Function Convergence

Figure 6.5 compares the cost-function value across iterations. Without noise, the cost decreases monotonically toward zero. With noise, the horizontal noise-floor line $\frac{1}{2}\|\boldsymbol{\eta}\|^2$ marks the expected least-squares form of the thermal noise $\boldsymbol{\eta}$. The noisy cost curve continues below this line at later iterations. Together with the RMSRE convergence curve, this indicates that the solver starts fitting the noise rather than only the underlying phantom signal.



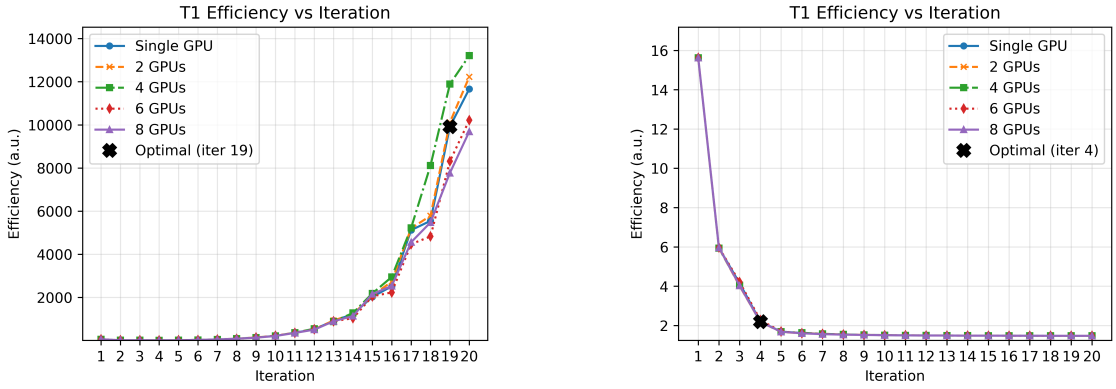
(a) Noiseless data

(b) Noisy data ($\mathrm{SNR}_{\mathrm{dB}} = 15.36$)

Figure 6.5: Cost function versus iteration. Noiseless reconstructions converge toward zero. Noisy reconstructions can continue below the noise-floor reference at later iterations.

6.3.3 T_nNR Efficiency

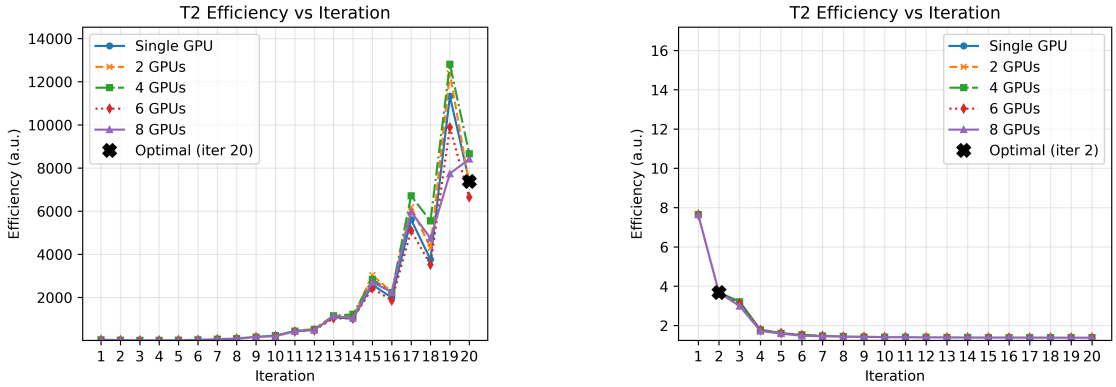
Figures 6.6 and 6.7 compare the $T_n\text{NR}/\sqrt{T_{\text{scan}}}$ efficiency across iterations. For each tissue ROI, T_nNR is computed as the mean reconstructed tissue value divided by its standard deviation. The plotted efficiency is the average over tissue ROIs divided by $\sqrt{T_{\text{scan}}}$. This metric should not be confused with the parallel efficiency in Figure 6.10. Here it checks whether the multi-GPU configurations preserve the same parameter-map precision as the single-GPU reconstruction. On noiseless data the curves separate at late iterations due to a property of the metric in the noiseless limit as described in Section 7.4.



(a) Noiseless data

(b) Noisy data

Figure 6.6: T_1NR efficiency versus iteration. The plotted value is the T_1NR averaged over tissue ROIs and divided by $\sqrt{T_{\text{scan}}}$.



(a) Noiseless data

(b) Noisy data

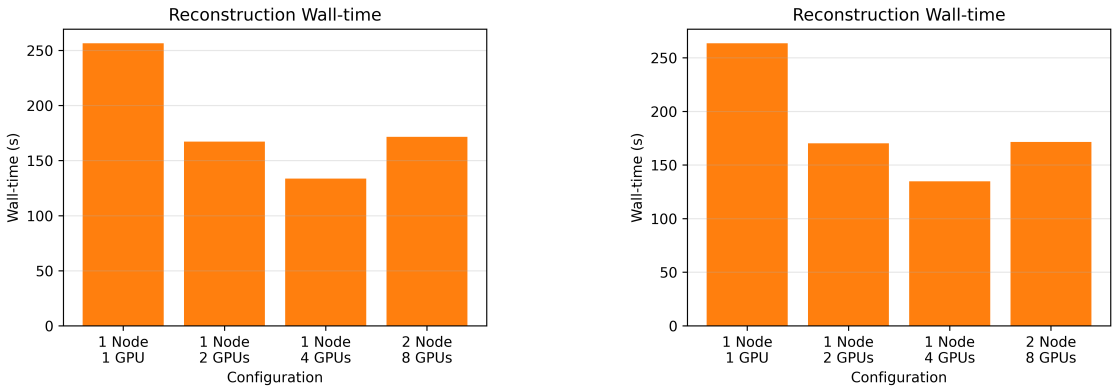
Figure 6.7: T_2NR efficiency versus iteration. The plotted value is the T_2NR averaged over tissue ROIs and divided by $\sqrt{T_{\text{scan}}}$.

6.4 Strong Scaling

This experiment fixes the problem size at $N = 224$ and varies the number of GPUs from one to eight. For both noiseless and noise-corrupted data, we measure reconstruction wall-time, speedup, parallel efficiency, and the per-iteration time breakdown for the 1, 2, 4, and 8 GPU configurations in Table 6.1.

6.4.1 Wall-time, Speedup, and Parallel Efficiency

Figure 6.8 reports the absolute wall-time of the complete reconstruction for the noiseless and noisy cases. In both cases, wall-time drops from one GPU to four GPUs within a single node. The eight-GPU run across two nodes does not improve further, showing that inter-node communication offsets the extra GPU work.



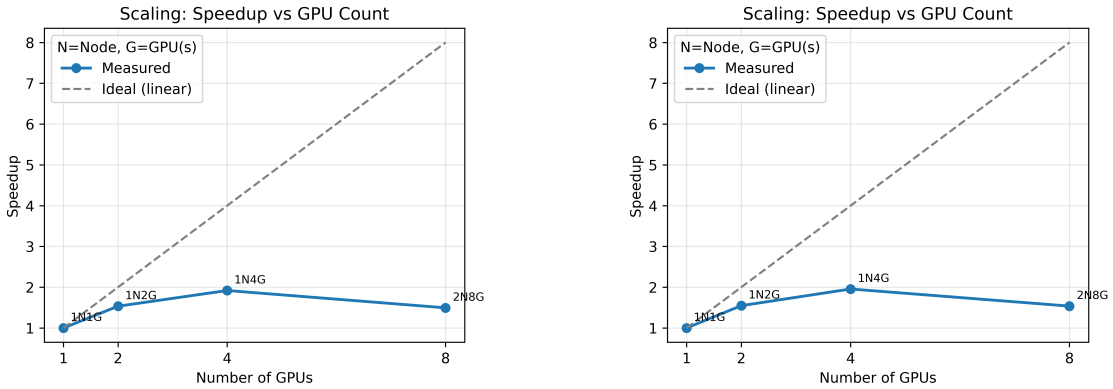
(a) Noiseless data

(b) Noisy data

Figure 6.8: Reconstruction wall-time at $N = 224$ across GPU configurations. Both noiseless and noisy reconstructions show the same pattern.

Figure 6.9 shows the same behaviour as speedup relative to the single-GPU baseline. Speedup improves within one node and then plateaus once the run crosses to two nodes.

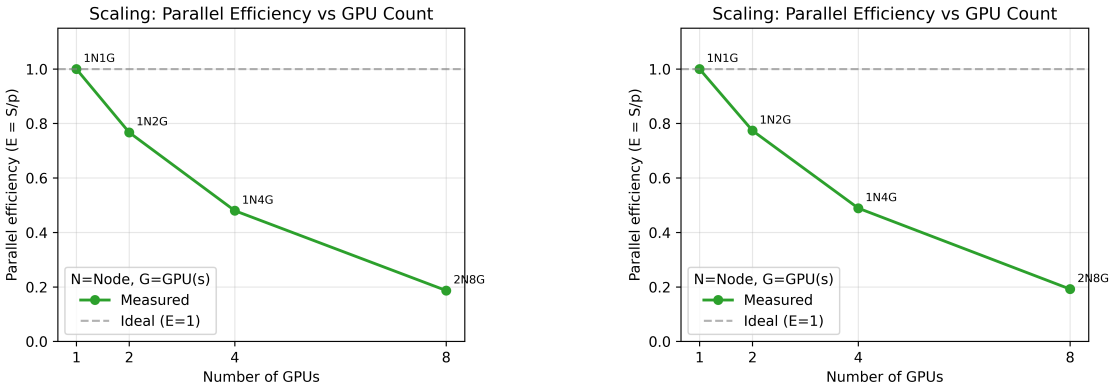
Figure 6.10 reports parallel efficiency, defined as speedup divided by the number of GPUs. It is 100% only when using p GPUs gives a $p\times$ speedup. An efficiency below 100% means that communication and synchronisation overheads prevent ideal linear scaling. In the noisy run, efficiency decreases from 77.4% on two GPUs to 48.9% on four GPUs and 19.2% on eight GPUs. The eight-GPU run does not recover efficiency from the larger GPU count, because the dominant collective operations now use TCP communication across nodes.



(a) Noiseless data

(b) Noisy data

Figure 6.9: Speedup relative to the single-GPU baseline. Both noiseless and noisy reconstructions show the same pattern. The dashed grey line marks ideal linear scaling.



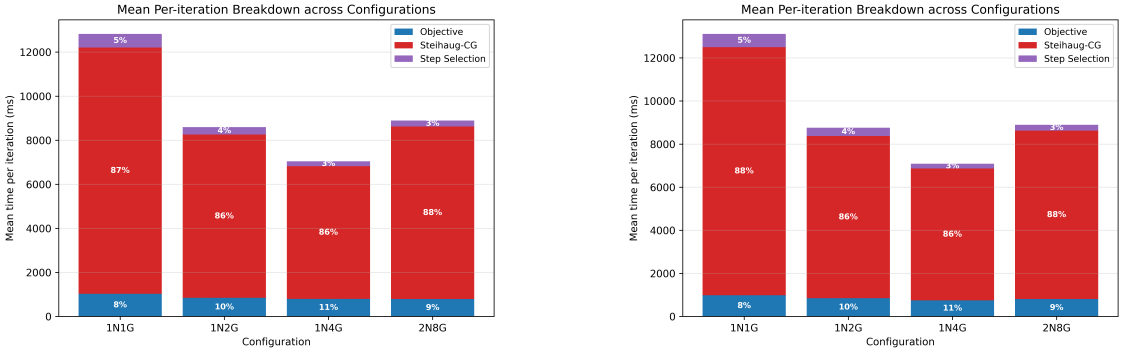
(a) Noiseless data

(b) Noisy data

Figure 6.10: Parallel efficiency across GPU configurations. Both the noiseless and noisy cases show decreasing efficiency as the number of GPUs increases. The dashed grey line marks ideal 100% parallel efficiency.

6.4.2 Per-iteration Time Breakdown

Figure 6.11 shows the three stages that dominate the recorded per-iteration runtime. The *Objective* stage evaluates the simulated signal and residual, corresponding to the signal computation in Section 5.4. The *Steihaug-CG* stage is the inner solve described in Section 5.5, and it repeatedly calls the matrix-free Hessian-vector operator from Section 5.4. The *Step Selection* stage scores candidate steps and updates the trust radius, as described in Section 5.5. Steihaug-CG accounts for most iteration time because it repeatedly applies the Hessian-vector operator.



(a) Noiseless data

(b) Noisy data

Figure 6.11: Per-iteration time breakdown across GPU configurations. Steihaug-CG is the dominant stage in both noiseless and noisy reconstructions because the inner CG loop repeatedly applies the matrix-free Hessian-vector operator.

6.5 Problem Size Scaling

The strong-scaling experiment in Section 6.4 fixes the problem size at $N = 224$. Here we vary the image dimension $N \in \{224, 320, 384, 448, 512\}$. All measurements in this experiment use noise-corrupted data ($\mathrm{SNR}_{\mathrm{dB}} = 15.36$). We use the noisy case because it better represents a realistic reconstruction scenario.

6.5.1 Setup and Baseline Selection

We compare one node with two GPUs, one node with four GPUs, two nodes with two GPUs, and two nodes with four GPUs. The two four-GPU configurations isolate the effect of communication over shared memory within one node versus TCP across two nodes. The single-GPU baseline runs out of memory for $N \geq 384$. The two-GPU single-node

configuration runs out of memory at $N = 512$. These failures occur when the required intermediate buffers exceed the available memory per A4000 GPU. For relative speedup and parallel efficiency, we use the two-GPU single-node configuration as the baseline.

6.5.2 Wall-time

Figure 6.12 reports wall-time as a function of N^2 (proportional to voxel count). All four configurations show increasing wall-time with problem size. The two-GPU single-node curve terminates at $N = 448$ because it runs out of memory at $N = 512$; the remaining three configurations cover the full range up to $N = 512$. The eight-GPU two-node configuration gives the shortest wall-time at every measured size.

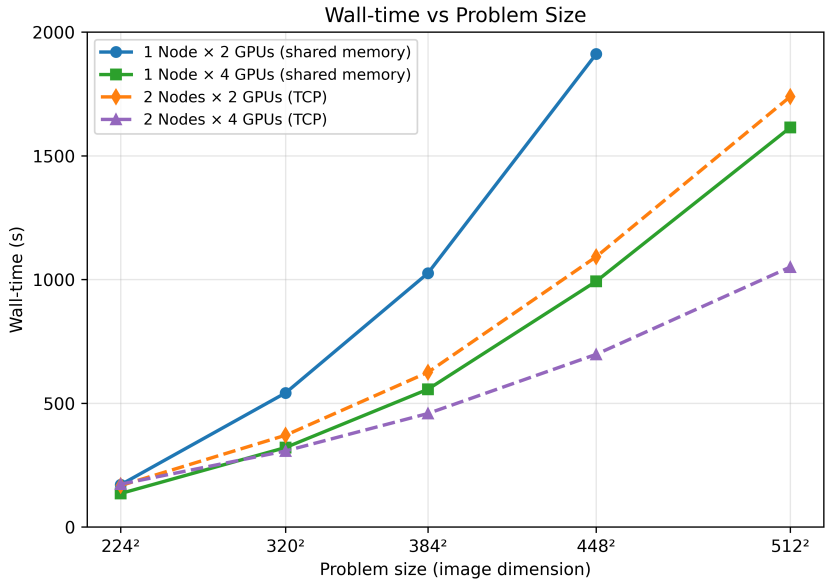


Figure 6.12: Reconstruction wall-time versus problem size N^2. In configuration labels, the letter N denotes nodes and G denotes GPUs. The 1N×2G configuration runs out of memory at $N = 512$.

6.5.3 Relative Speedup

Figure 6.13 reports the relative speedup of each configuration with respect to the two-GPU single-node baseline. The four-GPU single-node configuration is faster than the four-GPU two-node configuration because of the cost of cross-node TCP communication. The eight-GPU two-node configuration is not beneficial at $N = 224$, but becomes the fastest configuration for $N \geq 320$. This shows that crossing the node boundary becomes useful only when the problem is large enough to offset the extra communication cost.

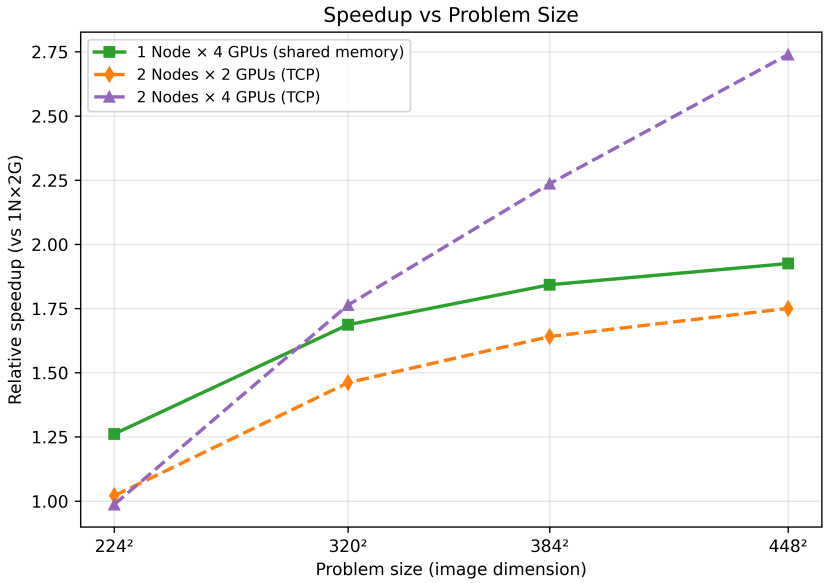


Figure 6.13: Relative speedup over the 1N×2G baseline. In configuration labels, the letter N denotes nodes and G denotes GPUs. The four-GPU single-node configuration outperforms the four-GPU two-node configuration, while the eight-GPU two-node configuration becomes beneficial for larger images.

6.5.4 Parallel Efficiency

Figure 6.14 reports parallel efficiency $E = S/(p/p_0)$, where $p_0 = 2$ is the GPU count of the two-GPU single-node baseline. The four-GPU single-node configuration has the highest efficiency because it adds GPUs without crossing the node. The cross-node configurations have lower efficiency, but their efficiency also improves with N. Larger reconstructions have more voxel-local work per GPU, so communication and synchronisation overheads take a smaller share of runtime. This confirms that multi-GPU reconstruction is more favourable for the large image sizes that motivate the implementation.

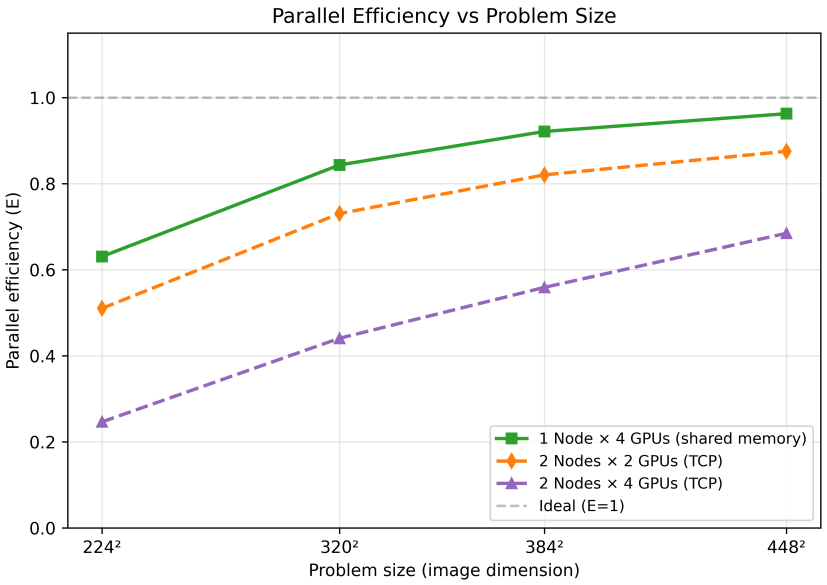


Figure 6.14: Parallel efficiency versus problem size. In configuration labels, the letter N denotes nodes and G denotes GPUs. The dashed grey line at $E = 1$ marks ideal scaling. All three configurations improve as N grows.

7

Discussion

This chapter interprets the evaluation results and explains their implications. The main result is that voxel partitioning and collective summation preserve the reconstruction behaviour of the single-GPU implementation, while allowing larger images to use the aggregate memory and compute capacity of multiple GPUs.

7.1 Numerical Equivalence of Multi-GPU Reconstruction

The validation results in Section 6.2 show that the multi-GPU implementation produces reconstructions equivalent to the single-GPU baseline. On noiseless data, the reconstructed maps are visually identical and the MARE values remain at the single-precision round-off level. On noisy data, the representative GPU configurations produce matching tissue boundaries, contrast, and noise texture. The convergence curves in Section 6.3 also overlap across GPU configurations. This result is important because the multi-GPU implementation changes where intermediate quantities are stored and where reductions occur. Parameter-domain data are partitioned by voxel range, while measurement-domain quantities are assembled by collective summation. The small residual differences are consistent with floating-point round-off from different reduction orders. The quantitative maps, RMSRE curves, and cost curves can still be interpreted using the same criteria as the single-GPU reconstruction. This is the basis for using the multi-GPU version in settings where a single GPU does not provide enough memory.

7.2 Capacity Scaling and Practical Performance

The strongest practical motivation for this work is memory capacity. The problem-size experiments in Section 6.5 show that the single-GPU baseline runs out of memory for $N \geq$

384, while multi-GPU configurations can reconstruct larger images. The two-GPU single-node configuration is also memory-limited at $N = 512$, so the largest tested reconstruction requires more aggregate GPU memory. The performance results should therefore be read as capability scaling as well as speedup. At the original $N = 224$ problem size, the strong-scaling results in Section 6.4.1 show that adding more GPUs gives limited speedup because the communication cost is already visible. For larger images, the problem-size efficiency results in Section 6.5.4 show better parallel efficiency because each GPU receives more voxel-local work. This makes the multi-GPU implementation more useful precisely where the single-GPU implementation becomes out of memory. The implication for qMRI reconstruction is that multi-GPU is most valuable for high-resolution reconstructions. It is less compelling as a pure throughput optimisation for small images.

7.3 Communication Cost and Deployment Recommendations

The strong-scaling and problem-size results in Sections 6.4 and 6.5 show that communication, not only GPU count, determines performance. The main communication costs follow from the forward model: the simulated signal and the forward Jacobian-vector product both require summation over voxel contributions. The per-iteration breakdown in Section 6.4.2 shows why this matters. Steihaug-CG dominates runtime because the inner CG loop repeatedly applies the matrix-free Hessian-vector operator. Each such application contains a measurement-domain collective summation. The deployment recommendation is therefore to prefer single-node multi-GPU execution when the reconstruction fits in one node. Within one node, shared-memory communication is faster than TCP communication across nodes. This is visible in the problem-size results in Sections 6.5.2 and 6.5.3, where the four-GPU single-node configuration outperforms the four-GPU two-node configuration at the same GPU count. Cross-node execution is still useful when the problem is large enough. In the problem-size experiment in Section 6.5.2, the eight-GPU two-node configuration is not beneficial at $N = 224$, but becomes the fastest configuration for $N \geq 320$. This indicates that crossing the node boundary should be treated as a capacity and larger-problem option.

7.4 Convergence Behaviour and Metric Interpretation

The convergence results in Section 6.3 show different behaviour for noiseless and noisy data. On noiseless data, RMSRE and cost both continue to decrease toward zero. This

is expected because the synthetic measurements contain no added noise term. On noisy data, the RMSRE reaches an early optimum and then increases at later iterations. At the same time, the cost can continue to decrease because the solver is fitting the noisy measured signal more closely. This means that a later iteration can improve the signal misfit while making the parameter maps less accurate with respect to the ground truth. For noisy data, the optimal iteration is therefore more informative than the final iteration.

The T_nNR efficiency curves in Section 6.3.3 add a view of convergence. They measure within-region precision rather than error against the ground truth. On noisy data the curves overlap across all configurations throughout the optimisation. On noiseless data, the T_nNR curves separate at late iterations. This separation is a property of the metric in the noiseless case. The T_nNR of a tissue region is the mean reconstructed value in that region divided by its standard deviation. With fixed scan time, a smaller within-region standard deviation gives a larger plotted efficiency value. In the noiseless phantom, every voxel in a tissue region is assigned the same ground-truth value. As the reconstruction converges, the standard deviation within each region collapses toward the floating-point floor. Once the denominator reaches this floor, the metric becomes sensitive to tiny round-off differences. The reduction order differs between GPU configurations, so the late-iteration denominator can differ slightly between configurations. The resulting T_nNR separation should therefore not be read as a difference in reconstruction quality.

8

Threats To Validity

This chapter discusses threats to validity using the classification by Wohlin *et al.* (31).

8.1 Internal Validity

Internal validity concerns whether the observed results are caused by the implementation and experiment design, rather than by uncontrolled errors.

The main internal threat is implementation correctness. The multi-GPU code changes data layout, which rank stores which voxels, and how reductions are performed, so an error in partitioning or gathering could produce incorrect maps. We mitigate this threat by comparing all GPU configurations against the single-GPU baseline on noiseless data. The parameter maps, MARE values, RMSRE curves, and cost curves match to the single-precision round-off level.

Another internal threat is randomness in phantom and noise generation. The experiments use fixed pseudo-random seeds for the phantom and for the noise vector. All ranks generate or receive the same synthetic raw data. This prevents rank-dependent input differences from being mistaken for numerical or performance differences.

Timing measurements are affected by system noise on the cluster. JIT compilation, CUDA launch overhead, job placement, and network load can change wall-time between runs. This threat is strongest for small problem sizes, where fixed overheads are a larger fraction of total runtime. For this reason, the problem-size analysis is limited to the selected image sizes in Section 6.5. It does not establish scaling behaviour for very small images, where fixed overheads could dominate runtime.

8.2 External Validity

External validity concerns whether the results generalise beyond the experimental setting.

The experiments use a synthetic Shepp–Logan based phantom. This gives known ground-truth maps and makes RMSRE and MARE possible, but it does not capture all effects present in real patient data. Motion, scanner imperfections, more complex anatomy, and model mismatch may change both reconstruction quality and convergence behaviour.

The acquisition setting is also limited. The evaluation uses a 2D Cartesian FISP sequence with fixed sequence parameters, fixed $K = 5$, and one tested noise level. The conclusions therefore apply most directly to this acquisition and signal model. Other trajectories, 3D acquisitions, stronger multi-coil settings, or different sequence designs may change the balance between voxel-local compute and measurement-domain communication.

The hardware setting is DAS-6 with NVIDIA RTX A4000 GPUs. Within a node, MPI uses shared-memory communication. Across nodes, the experiments use TCP communication. The DAS-6 cluster also provides InfiniBand, but CUDA-aware MPI with the cluster GID-table configuration was not reliable in this environment. The cross-node results may therefore underestimate performance on a cluster with a stable high-performance MPI path for GPU data.

8.3 Construct Validity

Construct validity concerns whether the measured quantities capture the concepts that the thesis claims to evaluate.

The reconstruction-quality metrics capture different aspects of accuracy. RMSRE measures the relative error over the tissue mask across iterations. MARE measures a scalar final-iteration error summary. T_nNR efficiency measures within-region precision relative to scan time. These metrics together support the equivalence claim.

The T_nNR efficiency metric is weak in the noiseless data because the within-region standard deviation would be very close to the floating-point floor. The denominator of T_nNR then becomes very small and sensitive to round-off. For this reason, noiseless T_nNR curves show separation at late iterations. The noiseless equivalence conclusion relies instead on parameter maps, RMSRE, and MARE.

Weak scaling is not evaluated. Weak scaling would increase GPU count and problem size together while keeping the work per GPU approximately constant. In this thesis, increasing image size also increases the measurement-domain communication payload. Each entry of

the simulated signal array, indexed by sample, readout, and receive coil, is assembled by summing encoded contributions from every voxels. Therefore weak scaling would mix increased local work with increased measurement-domain communication payload.

8.4 Conclusion Validity

Conclusion validity concerns whether the reported conclusions follow from the data.

The scaling conclusions are based on a finite set of GPU configurations. The main scaling figures use the balanced 1, 2, 4, and 8 GPU progression. The six-GPU configuration with three GPUs on each node is included in the reconstruction equivalence and convergence analyses, but not in the scaling curves. This choice keeps the scaling results focused on balanced configurations and avoids presenting an odd cross-node layout as part of a smooth scaling trend.

The number of repeated timing runs is limited by cluster availability. The large timing differences between single-node and cross-node configurations are robust enough to support the main conclusions. Smaller timing differences should be read more cautiously. This is especially true when two configurations differ by only a few percent.

The problem-size conclusions are also bounded by the tested range $N \in \{224, 320, 384, 448, 512\}$. The observed improvement in parallel efficiency with N supports the claim that larger reconstructions amortise communication better. It does not prove that the same trend continues indefinitely. At still larger image sizes, memory limits, MPI buffer sizes, or different solver behaviour may introduce new bottlenecks.

9

Related Work

This chapter compares the thesis against previous multi-GPU MRI reconstruction work about how the reconstruction is partitioned, which dependencies remain after partitioning, and which performance bottlenecks follow from those dependencies.

9.1 Spatial and Coil Partitioning

Spatial or channel dimensions are partitioned where much of the computation can remain independent until final aggregation. Most designs avoid most inter-GPU synchronisation because each slice can be reconstructed without intermediate data from other slices. Zhuo et al. assign different two-dimensional slices to different GPUs for iterative MR image reconstruction with field correction (10). Saybasili et al. use a multi-node, multi-GPU framework in which independent slices are processed asynchronously across reconstruction nodes (25). Murphy et al. present a scalable implementation of ℓ_1-SPIRiT in which decoupled two-dimensional slices and channel-wise work are distributed across devices (12). Schaetz et al. distribute 8-10 compressed coil channels across 4 GPUs in a real-time reconstruction but include temporal regularisation and peer-to-peer all-reduce operations, which introduce synchronisation overhead (11). More recent work also uses the coil dimension to reduce memory or computation. Plummer et al. use coil sketching for four-dimensional lung MRI reconstruction, where the reconstruction works with random subsets of coils while periodic full-coil information guides convergence (21). Schauman et al. split the forward operator by coil and subspace components to reduce the memory footprint of volumetric spatio-temporal subspace reconstruction (26). Although this thesis also uses a spatial partition, the partition is at a different level. The reconstruction target is one two-dimensional slice. Here, one slice can already exceed the memory capacity of a single

GPU. The slice itself therefore has to be partitioned. Voxel partitioning is the natural way to divide this work. Each rank stores a voxel range inside the same reconstructed slice and computes the voxel-local Bloch and adjoint-Jacobian work for that range. The simulated signal and forward Jacobian-vector product still have to sum contributions from all voxel ranges, so communication remains inside the repeated reconstruction loop.

9.2 Temporal and High-Dimensional Partitioning

Dynamic and high-dimensional MRI reconstructions often partition the temporal dimension. Schaetz et al. discuss real-time nonlinear inverse reconstruction where temporal decomposition is used but also limited because first l frames require strict sequential processing (18). Quan et al. partition dynamic compressed sensing MRI into temporal chunks with interior, exterior, and ghost regions (13). The ghost regions carry boundary information between chunks, and CUDA streams are used to overlap transfer and computation. These works show that temporal partitioning is effective when most work is local, but boundary dependencies still need explicit communication. Xue et al. use a cloud-based Gadgetron framework for distributed MRI reconstruction (16). Their system combines slice-wise distribution with temporal distribution across cardiac phases, and a gateway node manages job distribution and result aggregation. They also discuss partitioning along the readout direction after a one-dimensional inverse Fourier transform, where the resulting pieces can be reconstructed independently before final assembly. Higher-dimensional MRI introduces further partitioning choices. Lecoeur et al. accelerate four-dimensional reconstruction for MR-guided radiotherapy by sorting data into respiratory bins and distributing slices along the spatial axis (32). López-Ales et al. partition undersampled five-dimensional cardiac CINE reconstruction along the cardiac phase dimension (17). Their implementation has dependencies between adjacent cardiac and respiratory volumes, so synchronisation is required. Recent reconstruction work also reduces high-dimensional memory or compute cost by changing the algorithmic representation. Chen et al. reconstruct three-dimensional radial MPnRAGE with a self-supervised deep factor model (27). The memory issue in that work comes from training a high-dimensional model at full spatial resolution and across many inversion times. They reduce this cost with progressive training, where training starts at a lower spatial resolution before moving to the full problem. Fujita et al. study accelerated quantitative parameter mapping without fully sampled ground truth (23). Their memory issue comes from storing intermediate network outputs during training. They use checkpointing, so some intermediate values are recomputed during backpropagation instead

of being kept in GPU memory. Iakovlev et al. address unrolled cardiac cine MRI reconstruction (24). Standard unrolled training stores intermediate states for every unrolled block, so memory grows with the number of blocks. Their method treats the unrolled network as an implicit fixed-point problem. It shares weights across unrolled blocks and computes gradients without backpropagating through every stored block. This avoids the linear memory growth of standard end-to-end unrolled training. Shafique et al. accelerate a low-rank plus sparse model for dynamic MRI (22). They replace the expensive SVD step with compressed SVD and implement the dominant matrix operations with CUDA kernels. Their results also show that CPU–GPU transfer time can substantially reduce the apparent speedup. Together, these studies show that high-dimensional MRI reconstruction is often limited by how the problem is partitioned, which intermediate states must be stored, and where data must move. The temporal and volumetric partitioning studies mainly communicate boundary information between neighbouring blocks. For example, temporal chunks exchange ghost-region information, and adjacent cardiac or respiratory volumes require synchronisation at their boundaries (13, 17). In this thesis, the dominant dependency is different. Every voxel partition contributes to the same measurement-domain simulated signal and forward Jacobian-vector product, so these quantities require collective summation over all voxel partitions.

10

Conclusion

10.1 Summary of the Work

This thesis studied a multi-GPU implementation of qMRI reconstruction based on Bloch simulation. The reconstruction is solved by an outer trust-region inexact Gauss–Newton iteration, with an inner Steihaug-CG solve for the trust-region subproblem. The motivation was not only runtime. The single-GPU implementation runs out of memory for large two-dimensional images because the magnetisation-state and finite-difference derivative arrays grow with both the number of readouts and the number of voxels.

The design started from the data dependencies of the reconstruction. Voxel-local work, including Bloch simulation, derivative computations, and adjoint Jacobian application, can be divided across GPUs. Measurement-domain quantities are different. The simulated signal and the forward Jacobian-vector product both require summing encoded contributions from all voxel partitions. This leads to the central design choice of the thesis: partition parameter-domain data by voxel range, keep those data local to each rank, and assemble measurement-domain quantities by collective summation only where the reconstruction requires it.

The implementation realised this design in Julia, CUDA, and MPI. Each MPI rank controls one GPU and owns a contiguous voxel range. The implementation keeps parameter-domain vectors partitioned during the main reconstruction loop. It uses MPI reductions for measurement-domain arrays and for scalar or Boolean solver quantities. It also keeps the matrix-free Hessian-vector product structure of the original reconstruction, while making the forward Jacobian-vector summation use collective summation across GPUs.

10.2 Main Findings

The evaluation shows that the multi-GPU implementation preserves the numerical behaviour of the single-GPU baseline. On noiseless data, parameter maps and final error metrics match at the single-precision round-off level. On noisy data, the representative GPU configurations produce matching parameter maps, RMSRE curves, cost curves, and T_nNR efficiency curves. This supports the validity and correctness claim.

The problem-size-scaling results show the strongest benefit of the multi-GPU implementation. The single-GPU baseline runs out of memory for $N \geq 384$, and the two-GPU single-node configuration runs out of memory at $N = 512$. Larger multi-GPU configurations can reconstruct these larger images. Parallel efficiency and speedup improve with image size.

The strong-scaling results show that the implementation provides useful acceleration, but not ideal scaling. The per-iteration timing breakdown explains that Steihaug-CG dominates runtime because it repeatedly applies the matrix-free Hessian-vector operator, and each application contains a measurement-domain collective summation.

10.3 Future Work

The evaluation was performed with synthetic data generated from a two-dimensional Cartesian FISP sequence. Future work should test the same multi-GPU design on real measured data, other acquisition settings, and three-dimensional or multi-slice reconstructions. These settings may change the balance between voxel-local computation and measurement-domain communication. They would also show whether the same voxel partitioning strategy remains the best choice when the reconstruction problem contains additional spatial, coil, or temporal dimensions.

References

[1] G. Saltarelli, G. Di Cerbo, A. Innocenzi, C. De Felici, A. Splendiani, and E. Di Cesare. **Quantitative MRI in neuroimaging: a review of techniques, biomarkers, and emerging clinical applications**. *Brain Sciences*, **15**(10):1088, 2025. 1

[2] Oscar van der Heide, Alessandro Sbrizzi, Peter R. Luijten, and Cornelis A. T. van den Berg. **High-resolution in vivo MR-STAT using a matrix-free and parallelized reconstruction algorithm**. *NMR in Biomedicine*, **33**(4):e4251, 2020. 1, 5, 6, 12

[3] Oscar van der Heide, Cornelis A. T. van den Berg, and Alessandro Sbrizzi. **GPU-accelerated Bloch simulations and MR-STAT reconstructions using the Julia programming language**. *Magnetic Resonance in Medicine*, **92**(2):618–630, 2024. 1, 5, 12, 19

[4] Mehmet Akcakaya, Mariya Ivanova Doneva, and Claudia Prieto. *Magnetic Resonance Image Reconstruction: Theory, Methods, and Applications*. Academic Press, 2022. 2, 5

[5] Klaas P. Pruessmann, Markus Weiger, Markus B. Scheidegger, and Peter Boesiger. **SENSE: sensitivity encoding for fast MRI**. *Magnetic Resonance in Medicine*, **42**(5):952–962, 1999. 2

[6] Daniel K. Sodickson and Warren J. Manning. **Simultaneous acquisition of spatial harmonics (SMASH): fast imaging with radiofrequency coil arrays**. *Magnetic Resonance in Medicine*, **38**(4):591–603, 1997. 2

[7] Emmanuel J. Candès, Justin Romberg, and Terence Tao. **Robust uncertainty principles: exact signal reconstruction from highly incomplete fre-**

quency information. *IEEE Transactions on Information Theory*, **52**(2):489–509, 2006. 2

[8] Michael Lustig, David Donoho, and John M. Pauly. **Sparse MRI: the application of compressed sensing for rapid MR imaging**. *Magnetic Resonance in Medicine*, **58**(6):1182–1195, 2007. 2

[9] Haifeng Wang, Hanchuan Peng, Yuchou Chang, and Dong Liang. **A survey of GPU-based acceleration techniques in MRI reconstructions**. *Quantitative Imaging in Medicine and Surgery*, **8**(2):196–208, 2018. 2, 6, 7

[10] Yue Zhuo, Xiao-Long Wu, Justin P. Haldar, Wen-Mei W. Hwu, Zhi-Pei Liang, and Bradley P. Sutton. **Multi-GPU implementation for iterative MR image reconstruction with field correction**. In *Proc. Int. Soc. Mag. Reson. Med.*, page 2942, 2010. 2, 7, 51

[11] Sebastian Schaetz and Martin Uecker. **A multi-GPU programming library for real-time applications**. In *Algorithms and Architectures for Parallel Processing*, pages 114–128. Springer, 2012. 2, 7, 51

[12] Mark Murphy, Marcus Alley, James Demmel, Kurt Keutzer, Shreyas Vasanawala, and Michael Lustig. **Fast ℓ_1-SPIRiT compressed sensing parallel imaging MRI: scalable parallel implementation and clinically feasible runtime**. *IEEE Transactions on Medical Imaging*, **31**(6):1250–1262, 2012. 2, 6, 51

[13] Tran Minh Quan, Sohyun Han, Hyungjoon Cho, and Won-Ki Jeong. **Multi-GPU reconstruction of dynamic compressed sensing MRI**. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, **9351**, pages 484–492. Springer, 2015. 2, 7, 52, 53

[14] Chi Zhang, Davide Piccini, Omer Burak Demirel, Gabriele Bonanno, Burhaneddin Yaman, Matthias Stuber, Steen Moeller, and Mehmet Akcakaya. **Distributed memory-efficient physics-guided deep learning reconstruction for large-scale 3D non-Cartesian MRI**. In *2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI)*, pages 1–5. IEEE, 2022. 2, 7

[15] Batu Ozturkler, Arda Sahiner, Tolga Ergen, Arjun D. Desai, Christopher M. Sandino, Shreyas Vasanawala, John M. Pauly, Morteza Mardani,

and Mert Pilanci. **GLEAM: Greedy learning for large-scale accelerated MRI reconstruction**. arXiv:2207.08393, 2022. 2, 7

[16] Hui Xue, Souheil Inati, Thomas Sangild Sørensen, Peter Kellman, and Michael S. Hansen. **Distributed MRI reconstruction using Gadgetron-based cloud computing**. *Magnetic Resonance in Medicine*, **73**(3):1015–1025, 2015. 2, 7, 52

[17] Emilio López-Ales, Rosa-María Menchón-Lara, Federico Simmross-Wattenberg, Manuel Rodríguez-Cayetano, Marcos Martín-Fernández, and Carlos Alberola-López. **Multi-device parallel MRI reconstruction: efficient partitioning for undersampled 5D cardiac CINE**. *Sensors*, **24**(4):1313, 2024. 2, 7, 52, 53

[18] Sebastian Schaetz, Dirk Voit, Jens Frahm, and Martin Uecker. **Accelerated computing in magnetic resonance imaging: real-time imaging using nonlinear inverse reconstruction**. *Computational and Mathematical Methods in Medicine*, **2017**:1–11, 2017. 2, 7, 52

[19] Kathryn E. Keenan, Joshua R. Biller, Jana G. Delfino, Michael A. Boss, Mark D. Does, Jeffrey L. Evelhoch, Mark A. Griswold, Jeffrey L. Gunter, R. Scott Hinks, Stuart W. Hoffman, Geena Kim, Riccardo Lattanzi, Xiaojuan Li, Luca Marinelli, Gregory J. Metzger, Pratik Mukherjee, Robert J. Nordstrom, Adele P. Peskin, Elena Perez, Stephen E. Russek, Berkman Sahiner, Natalie Serkova, Amita Shukla-Dave, Michael Steckner, Karl F. Stupic, Lisa J. Wilmes, Holden H. Wu, Huiming Zhang, Edward F. Jackson, and Daniel C. Sullivan. **Recommendations towards standards for quantitative MRI (qMRI) and outstanding needs**. *Journal of Magnetic Resonance Imaging*, **49**(7):e26–e39, 2019. 5

[20] Thomas F. Coleman and Yuying Li. **An interior trust region approach for minimization subject to bounds**. *SIAM Journal on Optimization*, **6**(3):418–445, 1996. 6, 14

[21] Joseph W. Plummer, Pierre Daudé, Anastasia Tsakirellis, Jordan Taylor, Joel Moss, Rajiv Ramasawmy, Ahsan Javed, and Adrienne E. Campbell-Washburn. **Coil sketching for fast and efficient 4D lung MRI reconstruction**. *Magnetic Resonance in Medicine*, **95**(4):2241–2253, 2026. 6, 7, 51

[22] Muhammad Shafique, Sohaib Ayaz Qazi, and Hammad Omer. **Compressed SVD-based L+S model to reconstruct undersampled dynamic MRI data using parallel architecture**. *Magnetic Resonance Materials in Physics, Biology and Medicine*, **37**(5):825–844, 2024. 6, 53

[23] Naoto Fujita, Suguru Yokosawa, Toru Shirai, and Yasuhiko Terada. **Ground-truth-free deep learning approach for accelerated quantitative parameter mapping with memory efficient learning**. *PLOS One*, **20**(6):e0324496, 2025. 7, 52

[24] Nikolay Iakovlev, Florian A. Schiffers, Santiago L. Tapia, Daming Shen, KyungPyo Hong, Michael Markl, Daniel C. Lee, Aggelos K. Katsaggelos, and Daniel Kim. **Computationally efficient implicit training strategy for unrolled networks (IMUNNE): a preliminary analysis using accelerated real-time cardiac cine MRI**. *IEEE Transactions on Biomedical Engineering*, **72**(1):187–197, 2025. 7, 53

[25] Haris Saybasili, Daniel A. Herzka, Kestutis Barkauskas, Nicole Seiberlich, and Mark A. Griswold. **A generic, multi-node, multi-GPU reconstruction framework for online, real-time, low-latency MRI**. *Proc. 21st Meet. Int. Soc. Magn. Reson. Med.*, page 3838, 2013. 7, 51

[26] S. Sophie Schauman, Siddharth S. Iyer, Christopher M. Sandino, Mahmut Yurt, Xiaozhi Cao, Congyu Liao, Natthanan Ruengchaijatuporn, Itthi Chatnuntawech, Elizabeth Tong, and Kawin Setsompop. **Deep learning initialized compressed sensing (Deli-CS) in volumetric spatio-temporal subspace reconstruction**. *Magnetic Resonance Materials in Physics, Biology and Medicine*, **38**(2):221–237, 2025. 7, 51

[27] Yan Chen, Steve R. Kecskemeti, James H. Holmes, Curtis A. Corum, Nima Yaghoobi, Vincent A. Magnotta, and Mathews Jacob. **Accelerating 3D radial MPnRAGE using a self-supervised deep factor model**. *Magnetic Resonance in Medicine*, **94**(3):1191–1201, 2025. 7, 52

[28] Oscar van der Heide, Alessandro Sbrizzi, and Cornelis A. T. van den Berg. **Cartesian vs radial MR-STAT: An efficiency and robustness study**. *Magnetic Resonance Imaging*, **99**:1–13, 2023. 26, 27, 30

[29] **DAS-6: The Distributed ASCI Supercomputer 6**. `https://www.cs.vu.nl/das6/`. Accessed: 2026-04-29. 29

[30] Lawrence A. Shepp and Benjamin F. Logan. **The Fourier reconstruction of a head section**. *IEEE Transactions on Nuclear Science*, **21**(3):21–43, 1974. 29

[31] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2012. 47

[32] Bastien Lecoeur, Marco Barbone, Jessica Gough, Uwe Oelfke, Wayne Luk, Georgi Gaydadjiev, and Andreas Wetscherek. **Accelerating 4D image reconstruction for magnetic resonance-guided radiotherapy**. *Physics and Imaging in Radiation Oncology*, **27**:100484, 2023. 52