Real-time tomographic reconstruction

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C. J. J. M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op 1 juli 2020
klokke 11:15 uur

door

Jan-Willem Buurlage

geboren te Heerenveen
in 1991

Promotor:
Prof. dr. K. J. Batenburg

Copromotor:
Prof. dr. R. H. Bisseling — Universiteit Utrecht

Samenstelling promotiecommissie

Voorzitter:
Prof. dr. P. Stevenhagen

Secretaris:
Prof. dr. S. J. Edixhoven

Overige leden:
Prof. dr. S. Bals — Universiteit Antwerpen
Prof. dr. P. C. Hansen — Technical University of Denmark
Dr. B. Uçar — École Normale Supérieure de Lyon

The research presented in this dissertation was carried out at Centrum Wiskunde & Informatica (CWI) in Amsterdam.

Financial support was provided by The Netherlands Organisation for Scientific Research (NWO), project number 639.073.506.

Contents

Chapter 1

Introduction

The ability to look inside an object without destroying it is useful for many applications in, e.g., science, industry, and medicine. In tomographic imaging, *projection images* of the object are acquired along different directions using some kind of penetrating beam. From these projection images, the 3D interior can be computed using *tomographic reconstruction* methods [Her09; KS01].

Tomography is the technique behind many 3D imaging devices and techniques. Well known examples include medical CT scanners and μ-CT (laboratory) setups. At synchrotron radiation facilities, a (highly luminiscent) beam of X-rays is generated by accelerating electrons to high speeds along a circular trajectory, and this beam can be used to perform tomographic experiments. Electron microscopes penetrate samples with an electron beam instead of X-rays, and by tilting the sample a *tilt series* of projection images of, e.g., a nanoparticle can be produced.

The imaging techniques outlined above rely on the same basic principle. A *source* generates some form of penetrating radiation, for example X-rays. A 3D object is placed in front of the source. Inside this object, the radiation beam loses its intensity through interaction with the material. In other words, the beam is *attenuated*, and how much it gets attenuated depends on the kind of beam, and certain volumetric properties of the material (e.g., its density).

A *detector* captures a 2D profile of the radiation beam after it has passed through the object. The resulting 2D images are called *projection images*, and correspond in some sense to shadows of the object. This process is re-

peated for a number of combinations of source/detector positions. *Tomographic reconstruction algorithms* deal with the problem of obtaining a 3D profile of the interior of the imaged object, from a set of projection images.

This reconstruction step is typically performed *offline*, i.e., after the scan has completed. If instead the reconstruction can be performed *online* and in real time, then the insight gained from the 3D reconstruction of the object can be used to immediately steer the experiment. Acquisition parameters such as source and detector positioning could be optimized based on the internal structure of the specific object being imaged. Furthermore, dynamic processes in the imaged object could be followed as they occur. Consider an experiment where the behaviour is investigated of an object under changes in external parameters, such as the temperature or pressure. Real-time access to reconstructions would aid the on-the-fly adjustment of these parameters. For example, the operator can choose to stop heating the object as soon as a phase transition is observed.

The runtime of conventional reconstruction algorithms is typically much longer than the time it takes to acquire the projection images, and this prohibits the real-time reconstruction and visualization of the imaged object. The research in this dissertation introduces various techniques (in particular: new parallelization schemes, data partitioning methods, and a quasi-3D reconstruction framework) that significantly reduce the time it takes to run conventional tomographic reconstruction algorithms without affecting image quality. The resulting methods and software implementations put reconstruction times in the same ballpark as the time it takes to do a tomographic scan, so that we can speak of *real-time tomographic reconstruction*.

1.1 Tomographic reconstruction

Before discussing our novel techniques for real-time tomography, I will first give a general overview of mathematical methods that are required for a complete understanding of the work. For simplicity, we will initially consider 2D tomography in our mathematical description, where we aim to reconstruct the interior of a 2D object from its 1D projections. All ideas apply directly to 3D tomography, i.e., the reconstruction of a 3D object from its 2D projections.

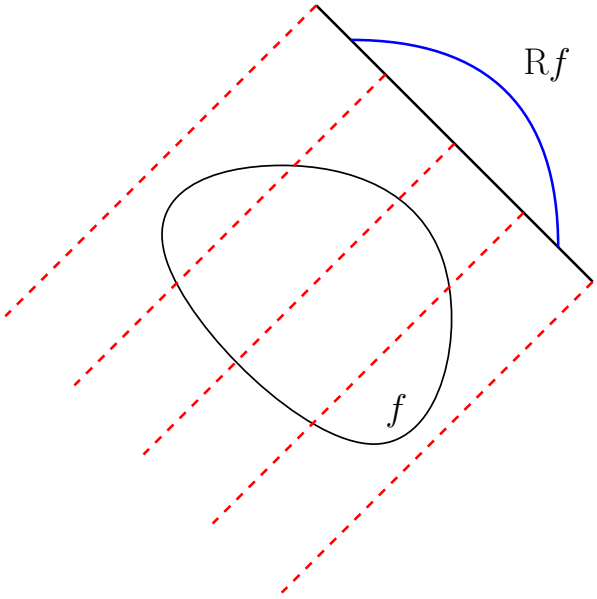


Figure 1.1: The Radon transform of a function are its line integrals. Here, we visualize 5 lines passing through the domain of a function f. The blue line represents the line integral values along this specific angle, assuming the function f is constant. This corresponds to a projection image in a tomographic experiment.

The interior of an object can be modelled as a function $f : \mathbb{R}^2 \to \mathbb{R}$, with $f(\mathbf{x})$ representing the value of some volumetric property of the object at location $\mathbf{x}$. The total attenuation of a ray ℓ passing through the object, can be written as a line integral over the function f. Therefore, it is natural to consider the function $\mathrm{R}f : L \to \mathbb{R}$, where L is the set of all lines in the plane, and

$$\mathrm{R}f(\ell) = \int_\ell f(\mathbf{x})\, d\mathbf{x}.$$

The function $\mathrm{R}f$ is called the *Radon transform* of f, and is closely related to projection images in tomography. If we have a ray SD passing through the imaged object from source position S to detector pixel D, then $\mathrm{R}f(SD)$ represents the measurement of the detector at pixel D for source position S. This is visualized in Figure 1.1.

Tomographic reconstruction methods aim to retrieve f from its Radon

transform Rf.

1.1.1 Projection matrix

Instead of considering the set of all lines, we consider a specific set of lines G that is specific to real-world experimental conditions. These conditions are captured in the *acquisition geometry*, corresponding to a collection of projection parameters. These parameters are (1) the detector position, (2) the detector tilt, (3) the position of the source, (4) the physical size of the detector, and (5) the detector shape in pixels. The set G is the set of lines defined by all source–detector pixel pairs over all projections.

Furthermore, we look at a discretized version of the real-valued function f. We assume the object is contained in a rectangular box, and discretize this box in a number of volume elements, or *voxels*.

The discrete analog of the Radon transform, is the linear transformation defined by the *projection matrix* W. This matrix has a row for each line in G, and a column for each voxel of the volume. An element W_{ij} of the matrix corresponds to the length of the line from G at index i, through the voxel with index j. This matrix is sparse, since each line only intersects a small minority of the voxels.

The product $\mathbf{y} \equiv W\mathbf{x}$ is called the *forward projection* of the image $\mathbf{x}$, and corresponds to the imaging experiment, with $\mathbf{y}$ representing the values of the projection images. A matrix–vector product with the adjoint transformation, $\mathbf{x} \equiv W^T\mathbf{y}$, is called the *backprojection* of $\mathbf{y}$. Visually, this corresponds to *smearing out* the measured values over the volume.

Tomographic reconstruction is a linear inverse problem: given measurements $\mathbf{y}$ find an image $\mathbf{x}$ such that:

$$W\mathbf{x} = \mathbf{y}. \tag{1.1}$$

The matrix W is generated by an acquisition geometry. To deal with noise in the data and errors in the model, the system is often solved in the least-squares sense:

$$\mathbf{x} = \underset{\tilde{\mathbf{x}} \in \mathbb{R}^n}{\arg\min} \; ||\mathbf{y} - W\tilde{\mathbf{x}}||_2. \tag{1.2}$$

There are two important systems that are used to compute least squares

solutions to underdetermined and overdetermined systems, respectively:

$$WW^T\mathbf{z} = \mathbf{y}, \qquad \mathbf{x} = W^T\mathbf{z}. \tag{1.3}$$

$$W^TW\mathbf{x} = W^T\mathbf{y}. \tag{1.4}$$

The system (1.4) is known as the *normal equations*. A solution to the normal equations is also a solution to (1.2).

1.1.2 Direct methods

Filtered backprojection

A common acquisition geometry is *parallel beam*, where the source is (conceptually) infinitely far away. In this case, the incoming rays in each detector pixel are perpendicular to the detector. Because of this property, the 3D reconstruction actually corresponds to a series of 2D reconstructions: one for each pixel row on the detector.

A fast reconstruction method for parallel beam geometries is *filtered backprojection* (FBP). We split the data into blocks, one for each one-dimensional projection:

$$\mathbf{y} = [\mathbf{y}^{[1]} | \cdots | \mathbf{y}^{[P]}]^T.$$

This data is first *filtered*, which in this case means a one-dimensional convolution $C_\mathbf{h}$ with kernel $\mathbf{h}$ is applied to each block. Next, the filtered data is backprojected:

$$\mathbf{x} = W^T \left(\bigoplus_{p=1}^{P} C_\mathbf{h} \right) \mathbf{y}. \tag{1.5}$$

Here, the direct sum indicates that $C_\mathbf{h}$ acts upon each block separately. One choice for $\mathbf{h}$ is the Ram–Lak filter, where for the ith element of the Fourier transformed filter we have $\mathscr{F}(\mathbf{h})_i \sim |\xi_i|$ with ξ_i the corresponding frequency. This filter yields an exact reconstruction in case of unlimited and noise-free data.

Using the convolution theorem, each row can be filtered efficiently:

$$C_\mathbf{h}\mathbf{y}^{[i]} \equiv \mathbf{h} * \mathbf{y}^{[i]} = \mathscr{F}^{-1}(\mathscr{F}(\mathbf{h}) \odot \mathscr{F}(\mathbf{y}^{[i]})).$$

Here, $\odot$ denotes pointwise multiplication.

FDK

If we have a point source at some finite distance from the object, the incoming beam is not parallel but cone shaped. When the source and detector move in a circular trajectory around the object (or equivalently, the object rotates around a single axis), we speak of a *circular cone-beam* geometry.

The FDK method [FDK84] is a method similar to filtered backprojection for such geometries. The data is still filtered in the same manner as for FBP, with a 1D convolution for each row on the detector. In addition, the data is weighted, with rows away from the center of the detector being dampened before backprojecting.

Let M be the number of pixel rows on the detector. As before, we split our data into blocks, one for each pixel row of our (now two-dimensional) projections. An FDK reconstruction is of the form:

$$\mathbf{x} = W^T \left(\bigoplus_{p=1}^{PM} C_{\mathbf{h}} \right) Z\mathbf{y}. \tag{1.6}$$

Here, $Z = \mathrm{diag}(z_i)$ is a diagonal matrix. If an element y_i is part of the pth projection, then it has an associated detector pixel position $\mathbf{f}_i$, source position $\mathbf{s}_p$, and detector plane D_p. The FDK weights are given by:

$$z_i \equiv \frac{d(\mathbf{s}_p, \mathbf{f}_i)}{d(\mathbf{s}_p, D_p)},$$

where $d(\cdot, \cdot)$ denotes the Euclidean distance in $\mathbb{R}^3$.

1.1.3 Iterative methods

Direct inversion methods such as FBP and FDK effectively approximate the inverse of the projection matrix in a single step. An alternative class of reconstruction methods are iterative methods. As the name implies, iterative methods refine an image over a number of iterations. Here, we will list a number of iterative methods that are commonly used in tomographic reconstruction. We roughly distinguish between two kinds of iterative methods: row-action methods relax equations defined by the rows of the projection matrix, while column-action methods act on the matrix columns instead.

ART

The prototypical row-action method is the Kaczmarz method [Kac37] (also known as ART [GBH70] in the tomography literature).

Each row $W_{i:}$ of the matrix W, together with the component y_i defines an equation that $\mathbf{x}$ should satisfy:

$$W_{i:} \cdot \mathbf{x} = y_i.$$

Geometrically, this equation defines a hyperplane. A simple, but (perhaps surprisingly) effective iterative way of solving (1.1), is to make the current iterate satisfy these equations in turn. Let $\mathbf{x}^{(k)}$ be the current iterate, $\mathbf{s}$ the step to take, and say we want the new iterate to satisfy the ith equation. Then we have for the next iterate:

$$\begin{aligned} W_{i:} \cdot \mathbf{x}^{(k+1)} &= y_i, \\ W_{i:} \cdot (\mathbf{x}^{(k)} + \mathbf{s}) &= y_i, \\ W_{i:} \cdot \mathbf{s} &= y_i - W_{i:} \cdot \mathbf{x}^{(k)}. \end{aligned}$$

We can view this as an extremely underdetermined system for the vector $\mathbf{s}$. The shortest step can be computed using the generalized inverse, since it yields the minimum-norm least squares solution:

$$\mathbf{s} = W_{i:}^{\dagger}(y_i - \mathbf{r}_i \cdot \mathbf{x}^{(k)}) = \frac{W_{i:}}{||W_{i:}||_2^2}(y_i - W_{i:} \cdot \mathbf{x}^{(k)}).$$

Geometrically speaking, the next iterate is the projection of the current iterate on the hyperplane defined by the ith equation.

ART corresponds to Gauss–Seidel iteration on the system (1.3).

ICD

Iterative coordinate descent (ICD) [Wat94], a column-action method, updates only one of the components x_j of the image x at each iteration:

$$\mathbf{x} \leftarrow \mathbf{x} + \delta \mathbf{e}_j,$$

where the vector $\mathbf{e}_j$ has a 1 in the jth position, and zeros elsewhere. It is natural to choose δ in such a way that the residual is minimized:

$$\begin{aligned} \mathbf{y} - W(\mathbf{x} + \delta\mathbf{e}_j) &= 0, \\ W\delta\mathbf{e}_j &= \mathbf{y} - W\mathbf{x}, \\ \delta W_{:j} &= \mathbf{y} - W\mathbf{x}. \end{aligned}$$

Here, $W_{:j}$ is the jth column of W. We view this as an overdetermined system for δ, and obtain the least-squares solution using the generalized inverse:

$$\delta = W_{:j}^{\dagger}(\mathbf{y} - W\mathbf{x}) = \frac{W_{:j}^T}{||W_{:j}||_2^2}(\mathbf{y} - W\mathbf{x}).$$

ICD corresponds to Gauss–Seidel iteration on the system (1.4).

SART

Simultaneous ART (SART) [And84] is a modification of ART to update the current iterate in order to (attempt to) satisfy a block of equations at the same time. The vector $\mathbf{y} = [\mathbf{y}^{[1]} | \cdots | \mathbf{y}^{[B]}]^T$ is split into B blocks. The update for a block $\mathbf{y}^{[i]}$ can be written as:

$$\mathbf{x} \leftarrow \mathbf{x} + \omega W^{[i]T} M^{[i]} (\mathbf{y}^{[i]} - W^{[i]}\mathbf{x}).$$

where ω is an optional relaxation parameter, and $M = \mathrm{diag}(\mathbf{m})$, $m_i = ||W_{i:}||_2^{-2}$. The blocks are updated in a sequential, cyclic manner.

SIRT

SIRT [Gil72; BG05] is an iterative method that makes use of full forward and backprojection operations, and is a simultaneous row-action method. We define a SIRT update to be:

$$\mathbf{x} \leftarrow \mathbf{x} + \omega C W^T R(\mathbf{y} - W\mathbf{x}),$$

where ω is an optional relaxation parameter, and:

$$C = \mathrm{diag}(\mathbf{c}), \quad c_j^{-1} = \sum_{i=1}^{m} W_{ij};$$

$$R = \mathrm{diag}(\mathbf{r}), \quad r_i^{-1} = \sum_{j=1}^{n} W_{ij}.$$

If instead $C = R = \mathrm{Id}$ is chosen, SIRT reduces to Landweber iteration which is equivalent to solving (1.4) using gradient descent.

Krylov

The k-dimensional Krylov subspace generated with W and $\mathbf{y}$ is defined as:

$$\mathcal{K}_k(W, \mathbf{y}) = \mathrm{span}\{\mathbf{y}, W\mathbf{y}, \ldots, W^{k-1}\mathbf{y}\}.$$

These subspaces are of interest since even for low k they contain good approximate solutions. An intuitive way to see why this is, is by considering the Cayley–Hamilton theorem that states that a matrix is a root of its own characteristic polynomial. This means the inverse can be expanded in terms of powers of W:

$$\begin{aligned} & a_0\mathrm{Id} + a_1 W + a_2 W^2 + \ldots + a_n W^n = 0, \\ \Longrightarrow \qquad & W^{-1} = b_0\mathrm{Id} + b_1 W + b_2 W^2 + \ldots + b_{n-1} W^{n-1}. \end{aligned}$$

We see that if large powers of W (can be coerced to) tend to zero, we can approximate the inverse using only small powers of W, which is equivalent to choosing solutions from the Krylov subspaces.

The kth iteratate generated by a *Krylov method* is the optimal element from $\mathcal{K}_k(W, \mathbf{y})$ for solving the least squares problem (1.2). There are multiple notions of optimality, and here we consider two. The first notion, and perhaps the most obvious, is to minimize the residual:

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathcal{K}_k(W,\mathbf{y})}{\arg\min} ||W\mathbf{x} - y||_2.$$

GMRES is a Krylov method of this kind. The second notion of optimality is to let the kth residual be perpendicular to $\mathcal{K}_k(W, \mathbf{y})$. The conjugate gradient (CG) method [HS52] is of this kind, and can be applied to symmetric positive definite systems. CGLS amounts to applying CG to the normal equations (1.4). Similarly, CGNE is obtained when applying CG to (1.3).

Variational methods

If prior knowledge on the image is available, then reconstruction quality can be significantly improved. A general way to do this is by adding penalty

terms, leading to *regularized* least squares problems. For example, if the image is piecewise constant then the gradient image will be sparse. In other words, we expect the 1-norm of the gradient magnitude $|\nabla \mathbf{x}|$ of the image to be small. This can be incorporated into the least squares problem:

$$\arg\min_{\mathbf{x}\in\mathbb{R}^n} ||\mathbf{y} - W\mathbf{x}||_2^2 + \lambda|| \, |\nabla \mathbf{x}| \, ||_1. \tag{1.7}$$

Algorithms for such variational formulations typically work by minimizing each successive term in turn. Examples of such algorithms include FISTA [BT09] and Chambolle–Pock [CP10].

Many iterative algorithms, including SIRT, Krylov methods, FISTA, and Chambolle–Pock, have one key aspect in common: *they alternate between forward projection and backprojection operations*. Furthermore, these are typically the most computationally expensive steps in the algorithm. Algorithms that use the forward projection and backprojection as subroutines are usually computationally more efficient than, e.g., ART that operate on individual equations, as they enable parallel updates.

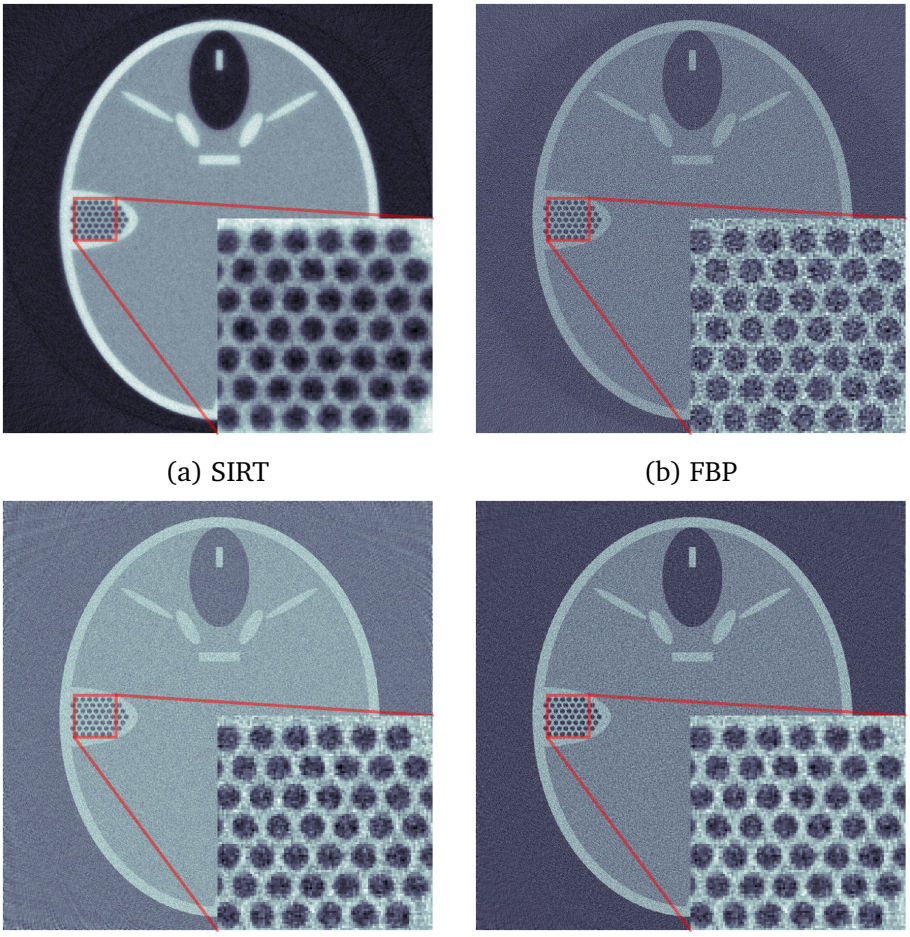
An important distinction between iterative and direct reconstruction methods, is that direct methods are *local*: each volume element can be reconstructed independently and efficiently from the (filtered) projection data.

A selection of reconstruction methods are compared in Figure 1.2.

1.2 Low-communication partitionings

Modern computers, and in particular computer systems that are used for large-scale scientific computations, are massively parallel. They typically have a high number of largely independent *processing elements*, such as processors, nodes, GPUs, or cores.

When designing algorithms that run on such systems, choosing the right data distribution is key. Specifically, data needed by one of the processing elements should be *local* to that element, so that it is easy to retrieve the necessary data during the execution of an algorithm. In other words, we should *partition* the input data and distribute the parts over the processing elements appropriately. A partitioning for p processing elements is defined as follows.



(a) SIRT (b) FBP

(c) SART (d) CGLS

Figure 1.2: Reconstructions of a 2D FORBILD phantom for a selection of different methods.

Definition 1. A p-way *partitioning* π_V of a set V is a collection of subsets $V_i \subset V$:

$$\pi_V = \{V_1, \ldots, V_p\},$$

such that

(i) the parts are non-empty: $V_i \neq \emptyset$,

(ii) the partitioning is complete: $\bigcup_i V_i = V$, and,

(iii) the parts are mutually disjoint: $i \neq j \implies V_i \cap V_j = \emptyset$.

A relevant example that is closely related to the partitioning problems treated in this dissertation is partitioning for the parallel sparse matrix–vector product (SpMV). For a sparse matrix A, the relevant input data V is a set of row–column pairs indicating the location of nonzero elements:

$$V \equiv \{(i, j) \mid A_{ij} \neq 0\}.$$

When we compute an SpMV $\mathbf{y} = A\mathbf{x}$, we are computing a series of inner products: one for each component of $\mathbf{y}$, since $y_i = A_{i:} \cdot \mathbf{x}$. During this process, the nonzeros in the jth column are multiplied with the component x_j.

In a distributed-memory setting the data involved in an SpMV, i.e., the vector components x_j and y_i, and the nonzero elements A_{ij}, are partitioned over the set of processing elements. Let $P(\cdot)$ be the part an element is assigned to. If for a nonzero $P(A_{ij}) \neq P(x_j)$, then the component x_j has to be communicated. If $P(A_{ij}) \neq P(y_i)$, then a partial sum has to be communicated for y_i [Bis04; CA01].

A good partitioning minimizes the total communication, as this is often the bottleneck in distributed-memory implementations, under the constraint that roughly the same number of elements are assigned to each part. This constraint is referred to as *load balancing*. (Bi)partitioning a sparse matrix for low communication volume is a combinatorial problem. If the nonzeros in the ith row (jth column) are assigned to the same part, then there is no communication for component y_i (component x_j). The total communication is therefore the total number of non-uniform rows and columns, see Figure 1.3.

When executing a distributed memory SpMV operation, each processor needs to be aware of relevant information of the global partitioning. For example, if processor s owns nonzeros in row i but it does not own the corresponding element y_i, then it needs to know the processor $P(y_i)$ in order to send its contribution to the final value of the component. This information is stored in *communication data structures*, which are precomputed for each processor and subsequently stored.

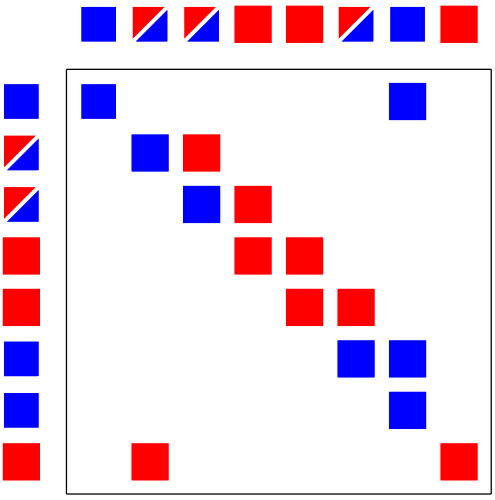


Figure 1.3: Sparse matrix partitioning. Each nonzero element of the 8×8 sparse matrix is indicated by a colored square. The elements are partitioned in two: a red part and a blue part. On the left and top of the matrix, the colors that are present in each row and column are indicated. The communication volume (CV) is the total number of rows and columns that have both red and blue elements. In this example, $CV = 5$.

1.3 Outline

The following chapters each correspond to a research article that was published during my time as a PhD student. Although they have been edited slightly, each chapter can still be read more or less independently. The dissertation can be split into three parts.

The first part consists only of Chapter 2. There, the BSP model is discussed, which is the basis of the performance analysis in subsequent chapters. Furthermore, we introduce the Bulk interface for implementing HPC software on top of the BSP model. The reference implementations used for the numerical experiments of the algorithms and methods introduced in later chapters, as well as the open-source reconstruction pipeline that resulted from the research presented in this dissertation, make extensive use of the Bulk interface.

In the second part, consisting of Chapter 3 and Chapter 4, we consider partitioning algorithms for tomographic reconstruction. Communication

in distributed-memory SpMVs involving projection matrices, i.e., the forward projection and backprojection operations, is of key importance for the overall execution time of tomographic reconstruction. In particular, successful partitioning strategies have the potential to make iterative reconstruction algorithms scale to dozens of GPUs, enabling them to run in real time. Previously developed SpMV partitioning methods are difficult to apply to tomographic reconstruction because of the data sizes involved, and do not make use of the geometric structure of the projections operators.

In Chapter 3, we formulate a low-communication partitioning problem for tomographic reconstruction. This partitioning problem is based on the underlying acquisition geometry that generates the projection matrix, instead of operating directly on the nonzero pattern of the matrix. We first give an exact geometric characterisation of the communication volume and load balance. We develop an efficient geometric recursive coordinate bisection (GRCB) partitioning method for the imaged 3D volume, and show that this can be translated into an implicit column partitioning of the projection matrix.

In Chapter 4 we further develop our geometric model for the communication volume and load balance in tomography. Our refined method works directly on the (cone-shaped) projections, removing the need to consider the discrete set of rays that correspond to the rows of the projection matrix W. In this continuous setting, we can still approximate the load balance and communication volume by considering a subdivision of the detector defined by the overlapping shadows of the parts in the partitioning. We also modify the GRCB algorithm to this continuous setting, resulting in a partitioning method that can run in real time. This enables the partitioning algorithm to run as part of a real-time reconstruction pipeline.

In the third part, consisting of Chapter 5 and Chapter 6, we propose a method for realizing live 3D reconstruction for real-time tomographic imaging, by exploiting the local property of direct methods such as FBP discussed before. We call this method quasi-3D reconstruction, and also demonstrate its use in imaging experiments.

In Chapter 5 we describe the core idea of our method. Instead of reconstructing the full 3D image, we develop a method for reconstructing arbitrary oblique slices at a fraction of the cost. By combining multiple slices, and allowing them to be chosen on-the-fly without reprocessing the

projection data, we maintain the illusion of having a full reconstructed 3D volume available. We also introduce the RECAST3D software, which is a full-stack reference implementation for quasi-3D tomographic reconstructions.

Finally, in Chapter 6 we show the feasibility of quasi-3D reconstruction in practice, by demonstrating real-time reconstruction capabilities at the TOMCAT beamline of the Swiss Light Source synchrotron radiation facility.

Chapter 2

A modern interface for BSP programs

The bulk-synchronous parallel (BSP) model was introduced as a bridging model for parallel programming by Valiant in 1989 [Val90]. It enables a way to structure parallel computations, which aids in the design and analysis of parallel programs.

The BSP model defines an abstract computer, the BSP computer, on which BSP algorithms can run. Such a computer consists of p identical processors, each having access to their own local memory. A communication network is available which can be used by the different processors to communicate data. During the execution of an algorithm, there are points at which bulk synchronizations are performed. The time of such a synchronization, the *latency*, is denoted by l. The communication cost per data word is denoted by g. The parameters l and g are usually expressed in number of floating-point operations (FLOPs). They can be related to *wall-clock time* by considering the computation rate r of the individual processors which is measured in floating-point operations per second (FLOP/s). A BSP computer is captured completely by the parameter tuple (p, g, l, r).

This chapter is based on:

Bulk: A Modern C++ Interface for Bulk-Synchronous Parallel Programs. *JW Buurlage, TR Bannink, RH Bisseling*. European Conference on Parallel Processing, 519-532, 2018

At a high level, a BSP algorithm is a series of supersteps that each consist of a *computation phase* and a *communication phase*. The processors of a BSP computer can simultaneously send and receive data, and they can do so independently. This means that the cost of communication is dominated by the maximum number of words sent or received by any processor. At the end of each superstep a bulk synchronization is performed ensuring that all outstanding communication has been resolved. Each processor runs the same program, but on different data, which means that BSP algorithms adhere to the Single Program Multiple Data (SPMD) paradigm.

The BSP cost of a BSP algorithm can predict the runtime of that algorithm when it is run on a BSP computer. This cost can be expressed completely in the parameters of a BSP computer. For each superstep, the cost depends on 1) $w_i^{(s)}$, the amount of work, measured in FLOPs, performed by processor s in the ith superstep, 2) $r_i^{(s)}$, the number of data words received, and 3) $t_i^{(s)}$, the number of data words transmitted (sent) by processor s in superstep i. The runtime of a parallel algorithm is dominated by the processor that takes the longest time, both for computation and communication. In the case of communication, we therefore require the concept of an h-relation, defined as the maximum number of words transmitted or received by any processor during the superstep, i.e., $h_i = \max_{0 \leq s < p} \max\{t_i^{(s)}, r_i^{(s)}\}$. This leads naturally to the following cost, the BSP cost, of a BSP algorithm consisting of k supersteps:

$$T = \sum_{i=0}^{k-1} \left(\max_{0 \leq s < p} w_i^{(s)} + g\, h_i + l \right).$$

The BSP model has inspired many parallel programming interfaces. BSPlib [Hil+98] describes a collection of a limited set of primitives which can be used for writing BSP programs in the C programming language. Libraries that implement the BSPlib standard include BSPonMPI [Sui] and MulticoreBSP for Java [YB12] and C [Yze+14]. Paderborn University BSP (PUB) [Bon+03] is an alternative BSP library that includes features not included in BSPlib such as subset synchronization and non-blocking collective operations. A functional BSP library is provided in BSML [LGB05] for the multi-paradigm programming language Objective CAML. Big data frameworks based on the BSP model include the popular MapReduce [DG04] and Pregel [Mal+10] frameworks introduced by Google. These frameworks have open-source implementations in respectively Apache Hadoop

and Apache Giraph, the latter of which is used for large scale graph computing by, e.g., Facebook [Chi+15]. Apache Hama [Sid+16] is a recent BSPlib alternative for the Java programming language.

For the C++ programming language, high-level parallel programming libraries include HPX [Hel+17], whose current interface focuses on asynchronous and concurrent applications, UPC++ [Zhe+14], which provides a generic and object-oriented partitioned global address space (PGAS) interface, and BSP++ [HFE10] which targets hybrid SMP architectures and implements direct remote memory access but not bulk-synchronous message passing.

Modern hardware is increasingly hierarchical. In a typical HPC cluster there are many nodes, each consisting of (several) multi-core processors together with accelerators such as GPUs or many-core coprocessors. Furthermore, there are multiple layers of random-access memory and caches which all differ in, e.g., size, latency, and read and write speed. In 2011, Valiant introduced Multi-BSP [Val11], a hierarchical execution model based on BSP. The nested execution of BSP programs is available in, e.g., the PUB, MulticoreBSP, and NestStep [Keß00] libraries.

In this chapter we introduce Bulk, a library for the C++ programming language. The current version is based on the C++17 standard [ISO17]. By leveraging common idioms and features of modern C++ we increase memory safety and code reuse, and we are able to eliminate boilerplate code from BSP programs. Furthermore, the flexible backend architecture ensures that programs written on top of Bulk are able to simultaneously target systems with shared memory, distributed memory, or even hybrid systems. The remainder of this chapter is structured as follows. In Section 2.1 we introduce the Bulk library, and highlight the differences with previous BSP libraries. In Section 2.2 we discuss two applications, *regular sample sort* and the *fast Fourier transform (FFT)*. In Section 2.3, we provide experimental results for these applications. Finally, in Section 2.4, we present our conclusions and discuss possibilities for future work.

2.1 The Bulk library

The Bulk library is a modern BSPlib replacement which focuses on the memory safety, portability, code reuse, and ease of implementation of BSP

```
bulk::backend::environment env;
env.spawn(env.available_processors(), [](auto& world) {
    world.log("Hello world from %d / %d\n",
              world.rank(), world.active_processors());
});
```

Listing 2.1: The entry point for parallelism using Bulk. We create an environment, where `backend` is a placeholder for a concrete backend such as MPI or C++ threads. Next, we spawn an SPMD block using all the available processors.

algorithms. Additionally, Bulk provides the possibility to program hybrid systems and it has several new features compared to existing BSP libraries. First, we present all the concepts of the library that are necessary to implement classic BSP algorithms.

Bulk interface Here, we give an overview of the Bulk C++ interface[1]. We use a `monospace` font in the running text for C++ code and symbols. A BSP computer is captured in an `environment`. This can be an object encapsulating, e.g., an MPI cluster, a multi-core processor or a many-core coprocessor. Within this BSP computer, an SPMD block can be spawned. Collectively, the processors running this block form a *parallel world* that is captured in a `world` object. This object can be used to communicate, and for obtaining information about the local process, such as the processor identifier (PID, in Bulk denoted `rank`) and the number of active processors. In all the code examples, `s` refers to the local rank, and `t` to an arbitrary target rank.

A simple program written using Bulk first instantiates an environment object, which is then used to spawn an SPMD block (in the form of a C++ function) on each processor, to which the local world is passed. See Listing 2.1 for a code example, and Table 2.1 for a table with the relevant methods.

The spawned SPMD section, which is a function that takes the world as a parameter, consists of a number of supersteps. These supersteps are delimited with a call to `world::sync`. The basic mechanism for communication

[1] Although we try to be as complete as possible, we do not give a detailed and exhaustive list of all the methods and functions provided by the library. For such a list, together with all the function signatures and further examples we refer to the online documentation which can be found at `https://jwbuurlage.github.com/Bulk/`.

Table 2.1: Available methods for `environment` and `world` objects.

class	*method*	*description*
`environment`	`spawn`	starts an SPMD block
	`available_processors`	returns maximum p
`world`	`active_processors`	returns chosen p
	`rank`	returns local processor ID s
	`next_rank`	returns $s+1 \pmod p$
	`prev_rank`	returns $s-1 \pmod p$
	`sync`	ends the current superstep
	`log`	logs a string message

revolves around the concept of a distributed variable, which is captured in a `var` object. These variables should be constructed in the same superstep by each processor. Although each processor defines this distributed variable, its value is generally different on each processor. The value contained in the distributed variable on the local processor is called the *local value*, while the concrete values on remote processors are called the *remote values*.

A distributed variable is of little use if it does not provide a way to access remote values of the variable. Bulk provides encapsulated references to the local and remote values of a distributed variable. We call these references *image* objects. Images of remote values can be used for reading, e.g., `auto y = x(t).get()` to read from processor `t`, and for writing, e.g., `x(t) = value`, both with the usual bulk-synchronous semantics. See Listing 2.2 for a more elaborate example. Since the value of a remote image is not immediately available upon getting it, it is contained in a `future` object. In the next superstep, its value can be obtained using `future::value`, e.g., `y.value()`.

In this simple example, we already see some major benefits of Bulk over existing BSP libraries; 1) we avoid accessing and manipulating raw memory locations in user code, making the code more memory safe and 2) the resulting code is shorter, more readable and therefore less prone to errors. Note that these benefits do not come at a performance cost, since it can be seen as syntactic sugar that resolves to calls to internal functions that resemble common BSP primitives.

```
bulk::var<int> x(world);

auto t = world.next_rank();
x(t) = 2 * world.rank();
world.sync();
// x now contains two times the ID of the previous logical processor

auto b = x(t).get();
world.sync();
// b.value() now contains two times the local ID
```

Listing 2.2: The basic usage of a distributed variable. The variable is created on each processor running the SPMD block. Its images can then be written to by using the convenient syntax `x(processor) = value`. Remote values are obtained by using the syntax `x(processor).get()`.

When restricting ourselves to communication based on distributed variables, we lose the possibility of performing communication based on slices or arrays. Distributed variables whose images are arrays have a special status in Bulk, and are captured in `coarray` objects. The functionality of these objects is inspired by Coarray Fortran [NR98]. Coarrays provide a convenient way to share data across processors. Instead of manually sending and receiving individual data elements, coarrays model distributed data as a 2D array, where the first dimension is over the processors, and the second dimension is over local 1D array indices. The local elements of a coarray can be accessed as if the coarray were a regular 1D array. Images to the remote arrays belonging to a coarray `xs` are obtained in the same way as for variables, by using the syntax `xs(t)`. These images can be used to access the remote array. For example, `xs(t)[5] = 3` puts the value 3 in the array element at index 5 of the local array at processor `t`. Furthermore, convenient syntax makes it easy to work with slices of coarrays. A basic slice for the element interval `[start, end)`, i.e., including `start` but excluding `end`, is obtained using `xs(t)[{start, end}]`. See Listing 2.3 for examples of common coarray operations. We summarize the most important put and get operations for distributed variables and coarrays in Table 2.2.

```
auto xs = bulk::coarray<int>(world, 4);

auto t = world.next_rank();
xs[0] = 1;
xs(t)[1] = 2 + world.rank();
xs(t)[{2, 4}] = {123, 321};

world.sync();
// xs is now [1, 2 + world.prev_rank(), 123, 321]
```

Listing 2.3: The basic syntax for dealing with coarrays.

Instead of using distributed variables, it is also possible to perform one-sided *mailbox communication* using message passing, which in Bulk is carried out using a `queue`. The message passing syntax is greatly simplified compared to previous BSP interfaces, without losing power or flexibility. This is possible for two reasons. First, it is possible to construct several queues, removing a common use case for *tags* to distinguish different kinds of messages. Second, messages consisting of multiple components can be constructed on demand using a syntax based on variadic templates. This gives us the possibility of *optionally* attaching tags to messages in a queue, or even denoting the message structure in the construction of the queue itself. For example, `queue<int, float[]>` is a queue with messages that consist of a single integer, and zero or more real numbers. See Listing 2.4 for the basic usage of these queues.

In addition to distributed variables and queues, common communication patterns such as `gather_all`, `foldl`, and `broadcast` are also available. The Bulk library also has various utility features for, e.g., logging and benchmarking. We note furthermore that it is straightforward to implement generic skeletons on top of Bulk, since all distributed objects are implemented in a generic manner.

Backends and nested execution Bulk has a powerful backend mechanism. The initial release provides backends for *distributed memory* based on MPI [MPI94], *shared memory* based on the standard C++ threading library, and *data streaming* for the Epiphany many-core coprocessor [ONU14]. Note that for a shared-memory system, only standard C++ has to be used.

Table 2.2: An overview of the syntax for puts and gets in Bulk. Here, `x` and `xs` are a distributed variable and a coarray, respectively, e.g., `auto x = bulk::var<int>(world)`, `auto xs = bulk::coarray<int>(world, 10)`

object	*image*	*description*	*code*
`var`	local (*)	set	`x = 5`
		use	`auto y = x + 3`
	remote	put	`x(t) = 5`
		get	`auto y = x(t).get()`
`coarray`	local (*)	set	`xs[idx] = 5`
		use	`auto y = xs[idx] + 3`
	remote	put	`xs(t)[idx] = 5`
		get	`auto y = xs(t)[idx].get()`
		put slice(**)	`xs(t)[{start, end}] = {values...}`
		get slice(**)	`auto ys = xs(t)[{start, end}].get()`

(*): a local image of a value of type `T` gets implicitly cast to a `T&` reference to the underlying value.
(**): subarrays corresponding to slices are represented using `std::vector` containers.

This means that a parallel program written using Bulk can run on a variety of systems, simply by changing the environment that spawns the SPMD function. No other changes are required. In addition, libraries that build on top of Bulk can be written completely independently from the environment, and only have to manipulate the world object.

Different backends can be used together. For example, distinct compute nodes can communicate using MPI while locally performing shared-memory multi-threaded parallel computations, all using a single programming interface. Hybrid shared/distributed-memory programs can be written simply by nesting `environment` objects with different backends.

```
// queue containing simple data
auto numbers = bulk::queue<int>(world);
numbers(t).send(1);
numbers(t).send(2);
world.sync();
for (auto value : numbers)
    world.log("%d", value);

// queue containing multiple components
auto index_tuples = bulk::queue<int, int, float>(world);
index_tuples(t).send({1, 2, 3.0f});
index_tuples(t).send({3, 4, 5.0f});
world.sync();
for (auto [i, j, k] : index_tuples)
    world.log("(%d, %d, %f)", i, j, k);
```

Listing 2.4: The use of message passing queues. The local inbox acts as a regular container, so we can use a range-based for loop. The messages can be accessed in a concise way using structured bindings.

2.2 Applications

2.2.1 Parallel regular sample sort

Here, we present our BSP variant of the parallel regular sample sort proposed by Shi and Schaeffer in 1992 [SS92]. Hill, Donaldson, and Skillicorn [HDS97] presented a BSP version, and Gerbessiotis [Ger15] studied variants with regular oversampling. Our version reduces the required number of supersteps by performing a redundant mergesort of the samples on all processors.

Our BSP variant is summarized in Algorithm 1. Every processor first sorts its local block of size $b = n/p$ by a quicksort of the interval $[sb, (s+1)b-1]$, where s is the local processor identity. The processor then takes p regular samples at distance b/p and broadcasts these to all processors. We assume for simplicity that p divides b, and, for the purpose of explanation, that there are no duplicates (which can be achieved by using the original ordering as a secondary criterion). All processors then synchronize, which

ends the first superstep. In the second superstep, the samples are concatenated and sorted. A mergesort is used, since the samples originating in the same processor were already sorted. Thus, p parts have to be merged. The start of part t is given by $start[t]$ and the end by $start[t+1]-1$. From these samples, p splitters are chosen at distance p, and they are used to split the local block into p parts. At the end of the second superstep, a local contribution X_{st} is sent to processor $P(t)$. In the third and final superstep, the received parts are concatenated and sorted, again using a mergesort because each received part has already been sorted. See Listing 2.5 for an illustration of Bulk implementations of the two communication phases of Algorithm 1.

Shi and Schaeffer have proven that the block size at the end of the algorithm is at most twice the block size at the start, thus bounding the size by $b_s \leq 2b$. A small optimization made possible by our redundant computation of the samples is that not all samples need to be sorted, but only the ones relevant for the local processor. The other samples merely need to be counted, separately for those larger and for those smaller than the values in the current block.

The total BSP cost of the algorithm, assuming p is a power of two, is

$$T_{\text{sort}} \leq \frac{n}{p}\log_2\frac{n}{p} + p^2\log_2 p + \frac{2n}{p}\cdot\log_2 p + \left(p(p-1)+2\frac{n}{p}\right)g + 2l. \quad (2.1)$$

This is efficient in the range $p \leq n^{1/3}$, since the sorting of the array data then dominates the redundant computation and sorting of the samples.

2.2.2 Fast Fourier Transform

The discrete Fourier transform (DFT) of a complex vector $\mathbf{x}$ of length n is the complex vector $\mathbf{y}$ of length n defined by

$$y_k = \sum_{j=0}^{n-1} x_j e^{-2\pi ijk/n} = \sum_{j=0}^{n-1} x_j {\omega_n}^{jk}, \quad \text{for } 0 \leq k < n, \quad (2.2)$$

where we use the notation $\omega_n = e^{-2\pi i/n}$. The DFT can be computed in $5n\log_2 n$ floating-point operations by using a radix-2 Fast Fourier Transform (FFT) algorithm assuming that n is a power of two.

Algorithm 1 Regular sample sort for processor $P(s)$, with $0 \leq s < p$.

input: **x**: vector of length n, n mod $p^2 = 0$, block distributed with block size $b = n/p$.

output: **x** sorted in increasing order, block distributed with variable block size $b_s \leq 2b$.

Quicksort($\mathbf{x}, sb, (s+1)b-1$);
for $i := 0$ **to** $p-1$ **do**
  $sample_s[i] := x[sb + i \cdot \frac{b}{p}]$;

for $t := 0$ **to** $p-1$ **do**
  put $sample_s$ in $P(t)$;
Sync;

for $t := 0$ **to** $p-1$ **do**
  $start[t] := tp$;
  for $i := 0$ **to** $p-1$ **do**
    $sample[tp+i] := sample_t[i]$;
$start[p] := p^2$;
Mergesort($sample, start, p$);

for $t := 0$ **to** $p-1$ **do**
  $splitter[t] := sample[tp]$;
$splitter[p] := \infty$;

for $t := 0$ **to** $p-1$ **do**
  $X_{st} := \{x_i : sb \leq i < (s+1)b \wedge splitter[t] \leq x_i < splitter[t+1]\}$;
  put X_{st} in $P(t)$;
Sync;

$\mathbf{x}_s := \cup_{t=0}^{p-1} X_{ts}$;

$start_s[0] := 0$;
for $t := 1$ **to** p **do**
  $start_s[t] := start_s[t-1] + |X_{t-1,s}|$;
$b_s := start_s[p]$;
Mergesort($\mathbf{x}_s, start_s, p$);

Listing 2.5: Two communication phases in the regular sample sort algorithm.

```
auto samples = bulk::coarray<T>(world, p * p); // Broadcast samples
for (int t = 0; t < p; ++t)
    samples(t)[{s * p, (s + 1) * p}] = local_samples;
world.sync();

auto q = bulk::queue<int, T[]>(world); // Contribution from P(s) to P(t)
for (int t = 0; t < p; ++t)
    q(t).send(block_sizes[t], blocks[t]);
world.sync();
```

Our parallel algorithm for computing the DFT uses the *group-cyclic distribution* with cycle $c \leq p$, and is based on the algorithm presented in [IB01] and explained in detail in [Bis04]. The group-cyclic distribution first assigns a block of the vector $\mathbf{x}$ to a group of c processors and then assigns the vector components within that block cyclically. The number of processor groups (and blocks) is p/c. The block size of a group is nc/p. Here, we assume that n, p, c are powers of two. For $c = 1$, we retrieve the regular block distribution, and for $c = p$ the cyclic distribution.

The parallel FFT algorithm starts and ends in a cyclic distribution. First, the algorithm permutes the local vector with components

$$x_s, x_{s+p}, x_{s+2p}, \ldots, x_{s+n-p},$$

by swapping pairs of components with bit-reversed local indices. The resulting storage format of the data can be viewed as a block distribution, but with the processor identities bit-reversed. The processor numbering is reversed later, during the first data redistribution. After the local bit reversal, a sequence of butterfly operations is performed, just as in the sequential FFT, but with every processor performing the pairwise operations on its local vector components. In the common case $p \leq \sqrt{n}$, the BSP cost of this algorithm is given by

$$T_{\text{FFT},\ p\leq\sqrt{n}} = \frac{5n\log_2 n}{p} + 2\frac{n}{p}g + l. \tag{2.3}$$

Table 2.3: Speedups of parallel sort (top) and parallel FFT compared to `std::sort` from libstdc++, and the sequential algorithm from FFTW 3.3.7, respectively. Also given is the sequential time t_{seq}.

		$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$	t_{seq}(s)
Sort	$n=2^{20}$	0.93	1.95	3.83	6.13	8.10	12.00	0.08
	$n=2^{21}$	1.01	2.08	4.11	7.28	10.15	15.31	0.19
	$n=2^{22}$	0.88	1.82	3.58	5.99	10.27	13.92	0.33
	$n=2^{23}$	0.97	1.90	3.63	6.19	11.99	16.22	0.72
	$n=2^{24}$	0.93	1.79	3.21	6.33	8.47	14.76	1.39
FFT	$n=2^{23}$	0.99	1.07	2.08	2.77	5.60	5.51	0.20
	$n=2^{24}$	1.00	1.26	2.14	3.07	5.68	6.08	0.45
	$n=2^{25}$	1.00	1.23	2.22	3.09	5.80	6.05	0.96
	$n=2^{26}$	0.99	1.24	2.01	3.28	5.48	5.97	1.93

2.3 Results

We evaluate the performance of Bulk implementations of the BSP algorithms sample sort and FFT outlined in the previous section. The numbers presented are obtained on a single computer with two Intel Xeon Silver 4110 CPUs, each with 8 cores and 16 hardware threads for a total of 32 hardware threads, using the C++ threads backend. The benchmark programs are compiled with GCC 7.2.1. The results are shown in Table 2.3. The parallel sort implementation is a direct translation of Algorithm 1, except that we opt for a three-phase communication protocol instead of relying on bulk-synchronous message passing to avoid potentially superfluous buffer allocations. The parallel FFT implementation is as described in Section 2.2.2, where we use FFTW [FJ98] as a sequential kernel[2]. The input arrays for both algorithms have size n, and they are run on p processors.

For the parallel sorting algorithm, the array contains uniformly distributed random integers between 0 and 2×10^5. We observe that good speedups are obtained compared to the sequential implementation. The maximum speedup seen is about 16× with $p = 32$ and $n = 2^{23}$.

For the FFT results, the vector has size n. We observe good scalability up to $p = 16$, where we seem to hit a limit presumably because of the shared floating-point unit (FPU) between two logical threads on the same physical core, and possibly also due to the memory requirements in the

[2]We use plans with the so-called planning-rigor flag `FFTW_MEASURE`.

redistribution phase.

Various other algorithms and applications have been implemented on top of Bulk. The current library release includes a number of examples, such as simple implementations for the *inner product*, or the *word count* problem. Future releases of the library are planned to have additional components. One such component is support for arbitrary data distributions, which is already available as an experimental feature. Furthermore, an open-source application in computed tomography, Tomos, has been developed on top of Bulk, illustrating that the library can be used for the implementation of more complicated software.

2.3.1 Bulk vs. BSPlib

We believe the main goal of Bulk, which is to improve memory safety, portability, code reuse, and ease of implementation compared to BSPlib, has been largely achieved. In Listing 2.6, we show a Bulk and a BSPlib implementation of a common operation. The Bulk implementation avoids the use of raw pointers, uses generic objects, requires significantly fewer lines of code, and is more readable.

We compare the performance of Bulk to a state-of-the-art BSPlib implementation, MulticoreBSP for C (MCBSP) [YR14], version 2.0.3 released in May 2018. We use the implementations of BSPedupack [Bis04], version 2.0.0-beta, as the basis of our BSPlib programs.

Table 2.4 shows the performance of Bulk compared to BSPlib. For sorting, the Bulk implementation is significantly faster, presumably because the internal sorting algorithm used is different. The Bulk implementation uses the sorting algorithm from the C++ standard library, whereas the BSPlib implementation uses the quicksort from the C standard library. The BSPedupack FFT implementation has been modified to use FFTW for the sequential kernel. For the FFT, MCBSP outperforms Bulk slightly on larger problem sizes.

In Table 2.5, the BSP parameters are measured for Bulk and MCBSP. The computation rate r is measured by applying a simple arithmetic transformation involving two multiplications, one addition and one subtraction, to an array of 2^{23} double-precision floating-point numbers. The latency l is measured by averaging over 100 bulk synchronizations without communication. The communication-to-computation ratio g is measured by

```
// BSPlib
int* xs = malloc(10 * sizeof(int));
bsp_push_reg(xs, 10 * sizeof(int));
bsp_sync();

int ys[3] = {2, 3, 4};
bsp_put((s + 1) % p, ys, xs, 2, 3 * sizeof(int));
bsp_sync();
...
bsp_pop_reg(xs);
free(xs);

// Bulk
auto xs = bulk::coarray<int>(world, 10);
xs(world.next_rank())[{2, 5}] = {2, 3, 4};
world.sync();
```

Listing 2.6: A comparison between Bulk and BSPlib for putting a subarray.

Table 2.4: Comparing implementations of BSPedupack running on top of MCBSP, to our implementations on top of Bulk.

Sort size	t_{MCBSP} (s)	t_{Bulk} (s)	FFT size	t_{MCBSP} (s)	t_{Bulk} (s)
$n = 2^{20}$	24.49	13.80	$n = 2^{22}$	0.153	0.144
$n = 2^{21}$	53.00	28.76	$n = 2^{23}$	0.305	0.320
$n = 2^{22}$	113.6	62.42	$n = 2^{24}$	0.629	0.694
$n = 2^{23}$	237.2	142.8			

communicating subarrays of various sizes, consisting of up to 10^7 double-precision floating-point numbers, between various processor pairs.

The MCBSP library uses a barrier based on a spinlock mechanism by default. This barrier gives better performance, leading to a low value for l. Alternatively, a more energy-efficient barrier based on a mutex can be used, which is similar to the barrier that is implemented in the C++ backend for Bulk. With this choice, the latency of MCBSP and Bulk are comparable. MCBSP is able to obtain a better value for g. We plan to include a spin-

Table 2.5: The BSP parameters for MCBSP and the C++ thread backend for Bulk.

Method	r (GFLOP/s)	g (FLOPs/word)	l (FLOPs)
MCBSP (spinlock)	0.44	2.93	326
MCBSP (mutex)	0.44	2.86	10484
Bulk	0.44	5.65	11702

lock barrier in a future release of Bulk, and to improve the communication performance further[3].

2.4 Conclusion

We present Bulk, a modern BSP interface and library implementation with many desirable features such as memory safety, support for generic implementations of algorithms, portability, and encapsulated state, and show that it allows for clear and concise implementations of BSP algorithms. Furthermore, we show the scalability of two important applications implemented in Bulk by providing experimental results. Even though both algorithms have $\mathcal{O}(n \log n)$ complexity, and nearly all input data have to be communicated during the algorithm, we still are able to obtain good speedups with our straightforward implementations. The performance of Bulk is close to that of a state-of-the-art BSPlib implementation, except for the mutex-based barrier.

[3]Spinlock barriers were introduced in Bulk version 1.1.0 which was released after the publication on which this chapter is based. With this implementation the measured latency l for Bulk is reduced to 467 FLOPs.

Chapter 3

Geometric partitioning for tomography

Tomography is a technique for creating 3D images of the interior of an object in a noninvasive way. Using some form of photon or particle beam, two-dimensional projections of the object are acquired, corresponding to integrals of some scalar volumetric property of the object (e.g., density, chemical concentration, etc.). Using *computed tomography* (CT) techniques, the measurements can then be used to perform a *tomographic reconstruction* of the three-dimensional profile of this property [Her09; KS01].

The projection measurements are performed by a two-dimensional detector containing a grid of pixels. In a tomographic scan, a finite number of *projection images* are acquired. The source position, detector position, and detector orientation vary for each projection image. Without loss of generality, we consider the source and the detector to move around a stationary object. Each source–pixel pair defines a line segment through the volume. All the source–pixel pairs for all projection images together combine to form a set of line segments. We call this set the *acquisition geometry*, and

This chapter is based on:

A geometric partitioning method for distributed tomographic reconstruction. *JW Buurlage, RH Bisseling, KJ Batenburg*. Parallel Computing 81, 104-121, 2019

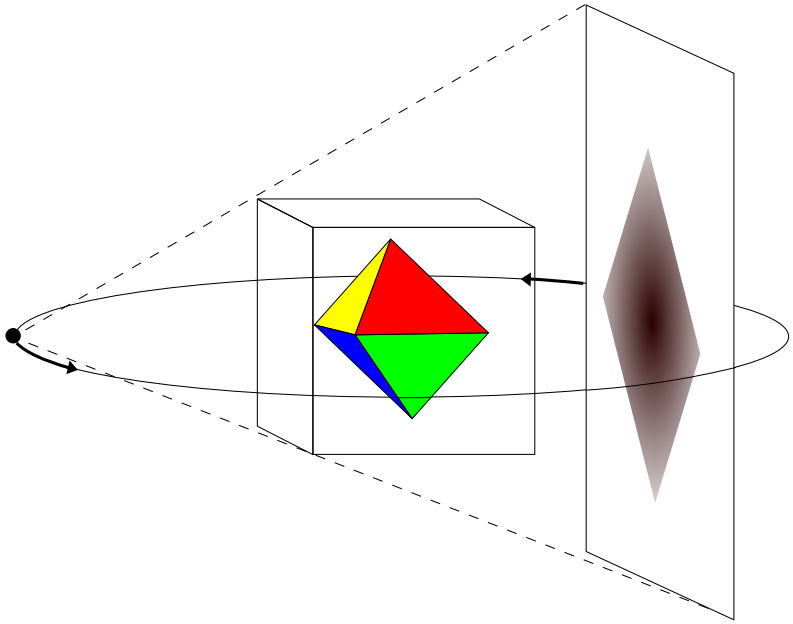


Figure 3.1: Schematic overview of a 3D tomography setup. Here, we show a single projection. On the left, we have a point source marked by a disk, which is emitting penetrating radiation. A cone-shaped collection of rays penetrates a cubic region of space shown in the center. As an example, we let it contain an object shaped as a octahedron. On the other side of the object we have the detector, shown as a square region, which performs intensity measurements of the rays. The projection of the object is shown in gray. The source and detector move opposite to each other along, for example, a circular path. Projection images are acquired at a finite number of source and detector positions.

denote it by $\mathscr{G}$. A common example is that the source positions correspond to equidistant points along a circle or helix, with the detector positioned on the opposite side of the object. An illustration of a basic tomography setup is shown in figure 3.1.

The scanned object is contained in a region $\mathscr{V} \subset \mathbb{R}^3$ which we always take to be a cuboid. We call this region the *object volume*.

Tomographic reconstruction methods aim to recover a function from a finite set of line integrals. Here, we list a number of commonly used methods. *Analytic methods* are based on discretizations of continuous in-

version formulas, and include filtered back projection type methods, such as FBP, FDK [FDK84] and Katsevich's algorithm for helical CT [Kat02]. An alternative is to formulate the reconstruction task as a linear inverse problem involving the tomographic system matrix. *Iterative methods* are then employed to solve this system; examples include ART [Kac37; GBH70], SART [And84], SIRT [Gil72], and Krylov subspace methods such as CGLS [HS52]. Most of these methods are row-action methods, and access a subset of the rows in each iteration. Column-action methods access a subset of the columns in each iteration instead [Wat94]. Other iterative methods include statistical reconstruction methods such as ML-EM [LC+84] and MBIR [SB93]. While analytic methods are typically easy to implement and are computationally efficient, they can lead to poor image quality if the reconstruction problem is underdetermined, if the measured data contains substantial noise, or if the acquisition geometry is non-standard. In these cases, iterative methods perform better, but they are computationally more expensive. With *variational methods*, tomographic reconstruction is viewed as a more general optimization problem, which allows for sophisticated noise models, as well as a priori knowledge of properties of the object to be incorporated through regularization terms. Methods such as FISTA [BT09], Chambolle–Pock [CP10] are popular for solving optimization problems in image reconstruction.

An important subset of these reconstruction methods performs matrix–vector products with the tomographic system matrix as their most computationally expensive subroutine. These methods include SIRT, CGLS and other Krylov methods, ML-EM, FISTA and Chambolle–Pock. The focus of the present work is to accelerate distributed-memory implementations of these methods by computing an appropriate data distribution. This data distribution depends heavily on the acquisition geometry that is used for the experiment.

Advances in acquisition technology, such as a rapidly increasing number of detector pixels operating at high frame rates, as well as a growing interest in multi-modal and multi-scale tomography, make reconstruction tasks increasingly computationally expensive. In particular, typical data sets that are acquired are quickly growing in size. Object volumes consisting of 2000^3 or even 4000^3 volume elements (voxels) are no longer uncommon, which means that reconstruction algorithms have to deal with vectors of sizes up to 64×10^9.

It is highly desirable to perform large-scale tomography in reasonable time. We consider this to be one of the main goals for the next generation of reconstruction techniques and algorithms. We distinguish between two approaches that are being taken in algorithm research for fast tomography. First, alternative reconstruction algorithms are being developed that approximate advanced but slow iterative methods, by faster and lighter methods [BB02; PB13; Kun+07; Nik+17; Zen12]. Second, techniques are being developed that take advantage of advances in computer hardware. Modern computing systems are increasingly parallel. By using the increased hardware capabilities to their full extent, reconstruction times can be greatly reduced. Modern implementations of common operations in tomographic reconstruction that are accelerated on multi-core processors or GPUs can give order-of-magnitude speedups over more conventional approaches [Chi+11; PBS11; SH14; Aar+15; Xu+10]. Additionally, with distributed implementations even higher reconstruction speeds can be obtained, but so far these implementations target only standard acquisition geometries for relatively low node counts [BG05; Pal+17; Ros+13]. In particular, for single-axis parallel-beam geometries, where conceptually the source is infinitely far away, efficient reconstruction is easy to realize because the task is trivially parallel [Mar+17; Wan+17]. The partitioning method we present here is flexible, and can be applied to arbitrary acquisition geometries.

In this chapter, we consider distributed-memory parallel methods for tomographic reconstruction. The main contribution of this chapter is to introduce an effective and efficient method for partitioning these data sets with respect to the matrix–vector products. The resulting partitioning depends only on the acquisition geometry, and is therefore reusable. The method can be used to automatically distribute the computational load over any number of processing elements. Furthermore, the resulting partitionings give insight into the computational structure of distributed-memory parallel methods in tomography.

The remainder of this chapter is structured as follows. In Section 3.1, we introduce the discretized tomographic reconstruction problem and the projection operations. In Section 3.2, we discuss distributed-memory parallel implementations of the projection operators, and introduce an associated geometric partitioning problem. In Section 3.3, we present an algorithm that solves the geometric partitioning problem. In Section 3.4, we

give the results of our numerical experiments. In Section 3.5, we discuss these results and the applicability of our method. Finally, in Section 3.6, we present our conclusions.

3.1 Projection operations

By discretizing the object volume $\mathcal{V}$ into *n voxels*, and linearizing the underlying physical model, we can represent the tomographic reconstruction problem as a linear system of equations:

$$W\mathbf{x} = \mathbf{b}. \tag{3.1}$$

Here, the vector $\mathbf{x}$ of size n is the image that is to be reconstructed, and the vector $\mathbf{b}$ of size m represents the measurements for each of the m line segments in the acquisition geometry. Matrix element w_{ij} of W is a *weight* related to the length of line $\ell_i \in \mathcal{G}$ in the jth voxel of the object volume. The $m \times n$ matrix W is sparse because every line intersects only a limited number of voxels.

The matrix W, called the *system matrix*, is usually not formed explicitly, because for any realistic number of voxels it quickly becomes prohibitively large. Instead, it is generated row-by-row by a *discrete integration method* (DIM), also called a kernel or projector, whenever W is used to, e.g., transform a vector. That is to say, tomography implementations are typically *matrix-free*. Common choices for a DIM are the slice-interpolated [XM06], and distance-driven [MB04] DIMs. In this chapter, we assume that the weights correspond exactly to the length of a line in a voxel. See figure 3.2 for an example of the construction of a tomography matrix.

The matrix–vector product $W\mathbf{x}$ is typically called *forward projection* in tomography literature, while a matrix–vector product with the transpose of the system matrix, i.e., $W^T\mathbf{y}$, is called the *back projection*. For a number of reconstruction methods, including SIRT and those based on Krylov subspaces, these projection operations make up the dominant part of the computational cost.

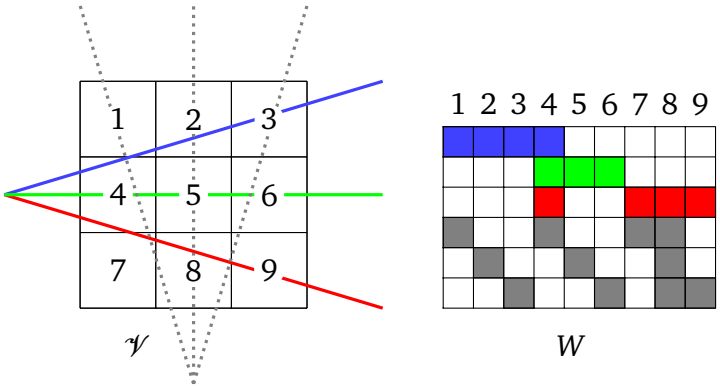


Figure 3.2: Construction of a tomography matrix in two dimensions. On the left, the object volume is shown together with two sets of three lines, corresponding to two projection images. One of these sets is shown in red, green and blue. The other projection is shown as dotted gray lines. The corresponding nonzero pattern, corresponding to nonzero lengths of the lines through the voxels, is shown on the right.

3.2 Distributed projection operations

The nonzero pattern of a typical tomography matrix is visualized in figure 3.3. There are some special aspects of a tomography matrix that distinguish it from a typical sparse matrix as we encounter them in for example the SuiteSparse matrix collection [DH11]. First, as mentioned in the previous section, it is too large to store explicitly. Instead, it is typically generated row-by-row from the acquisition geometry each time it is used. Second, the underlying structure is geometrical in nature, and this geometric information can be exploited for efficient implementations of operations involving the matrix. Third, if the object volume consists of n voxels, then there are $\mathcal{O}\left(n^{1/3}\right)$ nonzeros per row, since each row corresponds to a line intersecting a 3D volume (often a cube), so that the matrix has a relatively high density.

Running SpMV in parallel is an extensively studied problem [Bis04; CA99; Wil+09; YR14]. In order to compute a general SpMV $\mathbf{u} = A\mathbf{v}$ in parallel, the sparse matrix A has to be *partitioned*, i.e., its nonzeros should be assigned to one of the p available processors. This defines a (local) submatrix $A^{(s)}$ for each processor s. In addition, the vectors $\mathbf{v}$ and $\mathbf{u}$ need to be

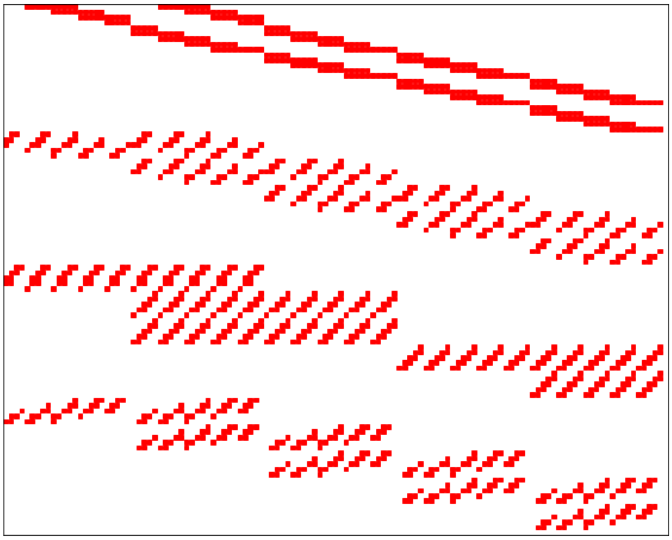


Figure 3.3: The nonzero pattern of the matrix W for a very small tomographic reconstruction problem. We consider a discretized object volume of $5 \times 5 \times 5$ voxels, with a detector shape of 5×5 pixels. The matrix was generated using a slice-interpolated DIM and a standard parallel geometry with 4 projections taken. The matrix has 100 rows, 125 columns and 1394 nonzeros.

partitioned. Generally, communication is required to obtain the necessary nonlocal components v_j, or to send nonzero contributions for components u_i that are not assigned to the local processor. Trying to minimize the total *communication volume* (not to be confused with the object volume) by finding a good partitioning gives rise to a rich optimization problem, and various methods and software packages have been specifically designed to treat this problem [CA99; Dev+06; VB05].

3.2.1 Partitionings

Because the system matrix W is not explicitly available, it is not easy to see how conventional partitioning methods can be applied. However, we do have access to the underlying geometric structure of the tomography problem, of which W is a discrete representation. Therefore, we can indirectly partition the matrix W by considering only the acquisition geometry

processors	*slab*	`onedimrow`	`onedimcol`	`mediumgrain`
16	111248	139216	108741	101402
32	233095	292833	210330	188294
64	3928222	3987888	2604930	2210671

Table 3.1: Communication volumes found by Mondriaan for different splitting methods. The imposed maximum imbalance is 0.05. The partitioned matrix corresponds to a typical circular cone beam acquisition geometry (see figure 3.6(a)) with 128^2 pixels on the detector, and an object volume of 128^3 voxels. `onedimrow` corresponds to a 1D row partitioning, `onedimcol` to a 1D column partitioning, and `mediumgrain` [PB14] to a 2D matrix partitioning. The communication volume of a slab partitioning, which is a 1D column partitioning corresponding to the object volume being split into p equal parts along the rotation axis, is shown as a reference.

$\mathcal{G}$ and the object volume $\mathcal{V}$.

We identify multiple options. First, we can partition the object volume $\mathcal{V}$. Each processor is then assigned a *subvolume* $\mathcal{V}^{(s)}$, and the local operations are restricted to the voxels in this subvolume. This corresponds to a 1D matrix column partitioning of W. Second, we can partition the geometry $\mathcal{G}$, i.e., assign a collection of lines to each processor. In this case, each processor is assigned a *subgeometry* $\mathcal{G}^{(s)}$, and the local operations are restricted to the lines in this subgeometry. This corresponds to a 1D matrix row partitioning. Third, we could consider 2D matrix partitionings. However, because of the matrix-free implementation of tomographic projection operations, using general 2D matrix partitionings seems to be infeasible.

We have investigated the performance of 1D column and row partitionings for a small tomographic problem for which the system matrix can still be formed explicitly, by a combinatorial approach using the Mondriaan partitioning software [VB05]. The results are shown in table 3.1, and suggest that 1D column partitionings perform much better than 1D row partitionings, and that limited further gains can be obtained with 2D partitioning if it would be possible to use them.

An intuitive explanation of the superior performance of 1D column partitioning compared to 1D row partitioning is that for any projection a small part of the volume will forward project to a small region of the detector, whereas any small region of the detector will back project to a larger part

of the volume.

Based on these considerations and numerical results, we shall focus exclusively on 1D matrix column partitionings. Thus, we assume that there is some partitioning of the volume:

$$\pi = \{\mathcal{V}^{(s)} \mid 0 \leq s < p\}. \tag{3.2}$$

so that for all $s \neq t$ the interiors of $\mathcal{V}^{(s)}$ and $\mathcal{V}^{(t)}$ are disjoint, and $\cup_{s=0}^{p-1} \mathcal{V}^{(s)} = \mathcal{V}$. Here, s and t are indices corresponding to one of the p processors. Let us derive how to express the parallel forward projection in this distributed setting. The forward projection $\mathbf{y} = W\mathbf{x}$ can be expressed as

$$y_i = \sum_{w_{ij} \in W(i,:)} w_{ij} x_j.$$

Here, $W(i,:)$ denotes the ith row of the matrix W. When performing this sum in parallel over a volume partitioned according to π, each processor s can *contribute* to component y_i, so that these components are no longer necessarily computed by a single processor. Each component y_i is the sum of local contributions:

$$y_i = \sum_{s=0}^{p-1} \left(\sum_{w_{ij} \in W^{(s)}(i,:)} w_{ij} x_j \right).$$

Here, $W^{(s)}$ is the local submatrix induced by the local volume $\mathcal{V}^{(s)}$. For a good partitioning, many rows of these submatrices should be empty, leading to only a limited number of contributions for each component y_i. For each component y_i, one of the contributors, the *owner* $\phi(i)$ of the ith component, is selected to receive all nonzero contributions and perform the outer sum. After the forward projection, the computed value of y_i will thus be stored exclusively on processor $\phi(i)$.

We summarize the resulting parallel algorithm for the forward projection in algorithm 2. It is given in single program multiple data (SPMD) form, and is parametrized on the processor number s. It is a *bulk-synchronous parallel* (BSP) [Val90] program, see [Bis04] for an introduction. In short, computations in BSP programs are carried out in supersteps. Communication is staged: it is prepared during a superstep, but carried out only at the end of that superstep. Communication is represented in the text

by PUT statements. In between the supersteps, there is a communication point where outstanding communication is resolved, followed by a global synchronization. This boundary is represented by a SYNC statement.

For locally storing and computing $\mathbf{y}$, we only need to consider the relevant (local) part, i.e., those components y_i for which the ith line ℓ_i intersects the local volume. This means that a volume partitioning induces subgeometries, given by the subset of the acquisition geometry with only lines that intersect the local subvolume. We will write $\mathcal{G}|_{\mathcal{V}^{(s)}}$ for these subgeometries.

The back projection operation can be implemented in a similar way. To back project into its local volume, a processor requires only the values y_i to which it contributes. If a back projection follows a forward projection, then this means that the owner $\phi(i)$ should communicate the computed value of y_i to all of its contributors at the beginning of the back projection operator. In particular, the communication volume for the back projection is the same as for the forward projection.

Algorithm 2 Parallel forward projection algorithm for processor s.

Input: $\mathbf{x}^{(s)}$, $W^{(s)}$, ϕ.
Output: $\mathbf{y}^{(s)}$

$\mathbf{z}^{(s)} = W^{(s)}\mathbf{x}^{(s)}$

for all i s.t. $z_i^{(s)} \neq 0$ **do**
  PUT $z_i^{(s)}$ in $\phi(i)$

–SYNC–

$\mathbf{y}^{(s)} \leftarrow 0$
for all i s.t. $\phi(i) = s$ **do**
  for all t s.t. $z_i^{(t)} \neq 0$ **do**
    $y_i^{(s)} \leftarrow y_i^{(s)} + z_i^{(t)}$

We end this section with two observations that are relevant for the matrix-free implementation of distributed projection operations, and illustrate how these implementations differ from general SpMV implementations. First, if the local subvolume $\mathcal{V}^{(s)}$ is a convex region, such as a cuboid, then the submatrix $W^{(s)}$ can be generated efficiently by the same

DIM as is used for W. Second, since a component y_i corresponds to a line segment for a source–pixel pair, we can efficiently find at once the set of contributors for groups of lines in the following way. We consider in turn each projection image, for each of which the position of the source is fixed. For each projection image, we look at the region to which the subvolume projects, i.e., the shadow of the subvolume on the detector. The regions where two or more shadows overlap, correspond to a group of lines with the same set of two or more contributors.

3.2.2 Partitioning the object volume

What is a *good* partitioning? The communication volume of the distributed forward projection operation arises because several subvolumes can contribute to the same component y_i. Geometrically, this can be interpreted as a line of the acquisition geometry intersecting several subvolumes associated with different processors. Before we give an expression for the total communication volume of the algorithm, we define:

$$\lambda_\ell(\pi) = |\{s \mid \ell \in \mathcal{G}|_{\mathcal{V}(s)}\}|,$$

i.e., the *line cut* $\lambda_\ell(\pi)$ is equal to the number of subvolumes in π that are intersected by the line ℓ. We assume that each line ℓ has a non-empty intersection with the full volume, so that we have $\lambda_\ell(\pi) \geq 1$.

We can express the communication volume of the forward projection and back projection operations directly in terms of the line cut:

$$V(\pi) = \sum_{\ell \in \mathcal{G}} (\lambda_\ell(\pi) - 1).$$

We will also put a load balancing constraint on the partitioning. To this end, we define the *computational weight* $\omega(j)$ of a voxel as the number of lines in the acquisition geometry that intersect the voxel. This computational weight equals the number of times a voxel is used during the forward projection. The *computational load* is the sum of the computational weights over all voxels in the local volume:

$$T^{(s)} = \sum_{j\,:\,x_j \in \mathcal{V}^{(s)}} \omega(j).$$

We define the *load imbalance* as:

$$\epsilon(\pi) = \max_{0 \le s < p} \frac{T^{(s)}}{T_{\text{avg}}} - 1.$$

Here, T_{avg} is the average computational load, i.e., the sum of the computational weights over the entire volume divided by the number of processors. To ensure that each processor performs roughly the same number of computations, the load imbalance should be kept close to zero. With these definitions in place, we can state the tomographic partitioning problem associated to distributed tomographic reconstruction:

> *Let $\mathcal{G}$ be an acquisition geometry, $\mathcal{V}$ the object volume, ϵ_{max} the maximum allowed load imbalance, and p the number of processors. Let Π denote the set of p-way volume partitionings, as given by* (3.2)*. The tomographic partitioning problem (TOMPP) is the following optimization problem:*
>
> $$\begin{aligned} \text{minimize}_{\pi \in \Pi} \quad & V(\pi) \\ \text{subject to} \quad & \epsilon(\pi) < \epsilon_{\max}. \end{aligned}$$

Since an acquisition geometry $\mathcal{G}$ is simply a set of line segments, we obtain a purely geometric problem: *partition a cuboid to minimize the total line cut for a given set of lines.*

3.3 Geometric recursive coordinate bisection

We look only at a specific class of partitionings, where each subvolume is a rectangular cuboid that is aligned with the coordinate axes. This restriction is motivated by the following considerations. First, partitioning problems are notoriously hard. Similar partitioning problems for graphs and hypergraphs have been shown to be NP-hard [BJ92; Len90]. Therefore we ought to reduce the search space considerably. Second, axis-aligned subvolumes are well suited for GPU computations. In particular, efficient GPU implementations rely on texture and index spaces that are rectangular. Third, the resulting partitionings should be easy to describe. The method we present will produce a binary space partitioning of the volume $\mathcal{V}$. This

means that the resulting partitionings can be used without any reference to the method that produced it.

In the following, when we write $\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1 \cup \ldots \cup \mathcal{V}_{p-1}$, all volumes $\mathcal{V}$ and $\mathcal{V}_i$ are assumed to be axis-aligned rectangular cuboids. In addition, the interiors for all pairs $\mathcal{V}_i$ and $\mathcal{V}_j$ with $i \neq j$ are disjoint. This union implies a partitioning π. We call such a partitioning a *cuboid partitioning*. Below, we write $V(\mathcal{V}_0, \ldots, \mathcal{V}_{p-1})$ for the communication volume $V(\pi)$.

We will first present the following observation, which informally states that the communication volume for a bipartitioning is equal to the number of lines crossing the interface between the two parts. This is illustrated in figure 3.4.

Lemma 2. *Let $\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1$, be a cuboid partitioning as above. The communication volume $V(\mathcal{V}_0, \mathcal{V}_1)$ for any acquisition geometry $\mathcal{G}$ is equal to the number of lines in $\mathcal{G}$ that have a non-empty intersection with the interface between $\mathcal{V}_0$ and $\mathcal{V}_1$.*

The core result that is used by our algorithm is a geometric version of theorem 2.2 in [VB05], and generalizes an observation from [CA99]. The result states that the communication volume is additive.

Theorem 3. *Let $\mathcal{V} = \mathcal{V}_0 \cup V_1 \cup \ldots \cup \mathcal{V}_{p-1}$ be a cuboid partitioning as above. Then for any acquisition geometry $\mathcal{G}$ we have:*

$$V(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-1}) = V(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-2} \cup \mathcal{V}_{p-1}) + V(\mathcal{V}_{p-2}, \mathcal{V}_{p-1}). \quad (3.3)$$

The proofs of lemma 2 and theorem 3 are straightforward and are given at the end of this chapter.

3.3.1 GRCB algorithm

With these results, we are ready to describe a *geometric recursive coordinate bisectioning (GRCB) algorithm* for the TOMPP. Taking an arbitrary acquisition geometry as input, it results in a cuboid partitioning of the object volume.

Recursive coordinate bisectioning (RCB) and generalizations of this method have proven to be successful partitioning strategies [BB87; Dev+16] for finite-element and finite-difference computations.

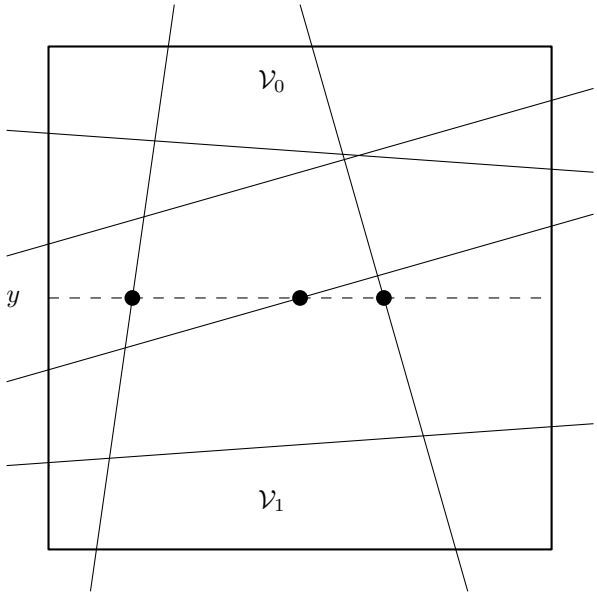


Figure 3.4: A set of lines through a square two-dimensional object volume $\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1$. The lines intersecting both subvolumes are exactly those lines that cross the horizontal interface at height y, shown here with a dashed line, between $\mathcal{V}_0$ and $\mathcal{V}_1$. In this case, three of the six lines have an intersection point (shown as $\bullet$) with the interface.

For the sake of presentation, we will restrict ourselves in this subsection in the following two ways. First, the number of processors is assumed to be a power of two. That is to say, we partition the volume into $p = 2^q$ parts, for some q. Second, the computational weights ω are assumed to be uniform over the object volume, so that we only have to consider the number of voxels of a part for load balancing considerations. We will describe later how it is possible to lift both of these restrictions.

The GRCB algorithm works as follows. We start with the full volume $\mathcal{V}$, and recursively split it into two parts, using an appropriate axis-aligned splitting plane that is to be computed. Theorem 3 ensures that each time we split, we only have to consider the subvolume being split and the lines intersecting this subvolume to obtain the change in communication volume. Furthermore, by lemma 2 we can compute this communication volume by counting the number of intersections in the splitting plane.

The overall form of the GRCB algorithm is given in algorithm 3. We

represent the resulting binary space partitioning as a balanced binary tree (the partitioning tree). We represent the tree recursively using nodes of the form $\langle n_1, v, n_2 \rangle$, where n_1 is the left child node, v is the value contained in the node, and n_2 is the right child node. With $\langle - \rangle$, we denote an empty node (a leaf of the tree has two empty child nodes). Each node of the tree has as its value a pair (d, a), with $1 \leq d \leq 3$ the axis along which the volume splits, and $a \in \mathbb{R}$ the position of the splitting plane along this axis. When splitting results in two computationally unequal parts, the load imbalance for the smaller part can be relaxed. We take the same approach as the Mondriaan partitioning method [VB05], and choose $\epsilon_{\max}$ dynamically and separately for the newly introduced subvolumes, depending on the current load imbalance and the total computational weight of the volume that is split.

Algorithm 3 Geometric recursive coordinate bisectioning (GRCB).

Subroutine: PARTITION
Input: $\mathcal{V}, \mathcal{G}, q, \epsilon_{\max}$
Output: the root node n of the partitioning (sub)tree

if $q = 0$ **then**
  return $\langle - \rangle$

$(d, a), \mathcal{V}_1, \mathcal{V}_2 \leftarrow$ SPLIT($\mathcal{V}$, $\mathcal{G}$, $\epsilon_{\max}/q$)

$\omega_{\max} \leftarrow (1 + \epsilon_{\max})\omega(\mathcal{V})/2^q$
$\epsilon_1 \leftarrow \omega_{\max} \cdot 2^{q-1}/\omega(\mathcal{V}_1) - 1$
$\epsilon_2 \leftarrow \omega_{\max} \cdot 2^{q-1}/\omega(\mathcal{V}_2) - 1$

$n_1 \leftarrow$ PARTITION($\mathcal{V}_1$, $\mathcal{G}|_{\mathcal{V}_1}$, $q-1$, ϵ_1)
$n_2 \leftarrow$ PARTITION($\mathcal{V}_2$, $\mathcal{G}|_{\mathcal{V}_2}$, $q-1$, ϵ_2)
return $\langle n_1, (d, a), n_2 \rangle$

The *splitting subroutine* shown in algorithm 4 computes a split for a volume $\mathcal{W}$ and a set of lines $\mathcal{H}$ through this volume. At the beginning of this subroutine, we compute for each line in $\mathcal{H}$ the two intersection points with the boundary of the volume $\mathcal{W}$. We call these pairs of intersection points belonging to the same line *partners*. All the intersection points

together make up a set E which we call the *event points*.

Next, we perform three plane sweeps, one for each of the three axes. Before we sweep along the dth axis, we preprocess the set of event points. First, we sort the event points by their dth coordinate. Second, for each event point, we decide if it is an *incoming* event or an *outgoing* event with respect to the dth axis. An event point is incoming if its partner has a larger dth component. If its partner has a smaller dth component, then it is outgoing. If their dth components are equal, the events can be safely ignored for this sweep, since the line will always be completely contained in one of the two subvolumes.

We are now ready to describe the plane sweep, which is illustrated in figure 3.5. Conceptually, we move a sweeping plane (perpendicular to the dth axis) that starts outside of the volume, by slowly increasing its dth coordinate. This plane will represent a candidate split of the volume $\mathscr{W}$. Since it starts outside of the volume, initially there are no lines crossing the interface. We stop at each event point. If the event is incoming, then the corresponding line will begin intersecting the sweeping plane. If the event is outgoing, then the corresponding line will no longer intersect the sweeping plane. This means that during the sweep, the number of lines intersecting the sweeping plane increases or decreases by one at each event point. In particular, it is very easy to keep track of the communication volume that would be incurred if the current sweeping plane would be taken as a splitting plane.

At each of the event points, the load balance constraint is checked. If it is satisfied, and the communication volume is the lowest among all valid splits encountered so far, we store the current sweeping plane as the current split candidate. After the third plane sweep, the split that is currently stored as the best one is returned.

After performing $p-1$ splits, the GRCB algorithm terminates. The splitting routine consists of the following computational steps. First, we compute the intersections in $O(m)$ time, where m is the number of lines. Second, we sort these intersections for each axis in $O(m \log m)$ time to obtain the events for the plane sweeps. Finally, the plane sweeps each consist of a loop over the $O(m)$ events, and the body of this loop runs in constant time. We conclude that sorting the intersections dominates the computational costs of the splitting procedure. Therefore, the full GRCB algorithm runs in $O(pm \log m)$ time. To put this into context, a single SpMV involving

Algorithm 4 Bisecting a volume $\mathscr{W}$ to minimize the line cut for a set of lines $\mathscr{H}$.

Subroutine: SPLIT.
Input: $\mathscr{H}$, $\mathscr{W}$, $\epsilon_{\max}$
Output: (d, a), $\mathscr{W}_1$, $\mathscr{W}_2$

compute set E of intersections of $\mathscr{H}$ with $\mathscr{W}$
$V_{\min} \leftarrow \infty$
$(d_{\text{best}}, a_{\text{best}}) \leftarrow (\infty, \infty)$

for d in $\{1, 2, 3\}$ **do**
  sort E by dth coordinate
  $V \leftarrow 0$
  for x in E **do**
    if event **x** is incoming **then**
      $V \leftarrow V + 1$
    else if event **x** is outgoing **then**
      $V \leftarrow V - 1$
    if load imbalance $\epsilon_{\max}$ is satisfied with split (d, a), and $V < V_{\min}$ **then**
      $V_{\min} \leftarrow V$
      $(d_{\text{best}}, a_{\text{best}}) \leftarrow (d, x_d)$

Let $\mathscr{W}_1$ and $\mathscr{W}_2$ be the two subvolumes for the split $(d_{\text{best}}, a_{\text{best}})$

return $(d_{\text{best}}, a_{\text{best}}), \mathscr{W}_1, \mathscr{W}_2$

a tomographic projection matrix runs in $O(mn^{1/3})$ time. The GRCB algorithm is efficient, and the resulting partitionings can be reused when the same acquisition geometry is employed for multiple scans. This is the case, for example, with a lab scanner that has fixed source and detector positions.

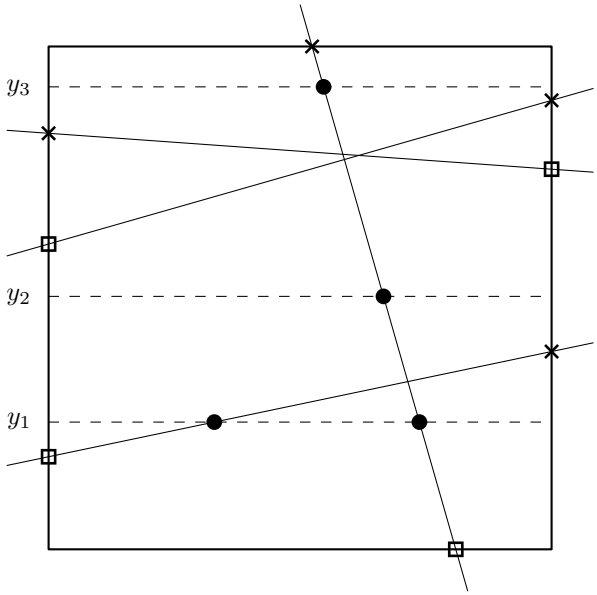
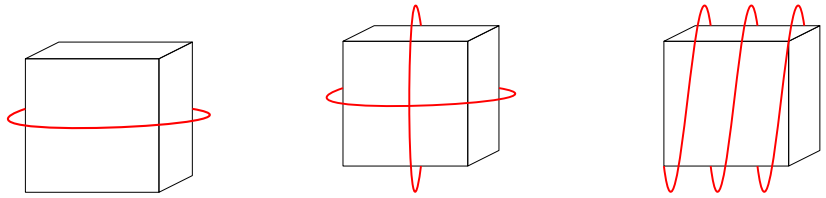


Figure 3.5: Visualization of the 2D equivalent of the 3D plane sweep described in algorithm 4. Imagine that we are considering a horizontal candidate interface which we are moving upwards, i.e., we gradually increase the y coordinate of the interface. If we were to split the volume according to the current candidate interface, the communication volume would be given by the number of lines crossing that interface. The only y coordinates where this number changes correspond to the y coordinates of intersection events, i.e., points where a line intersects the object volume boundary. Outgoing intersection events (shown as $\times$), and incoming intersection events (shown as $\square$) are marked. We illustrate candidate interfaces (shown as a dotted line) together with the interface intersections (shown as $\bullet$), for three different y coordinates.

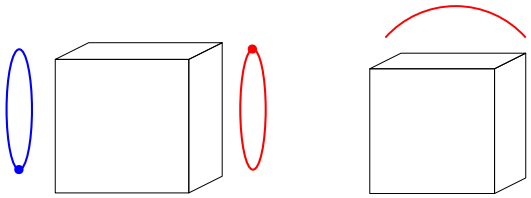
3.3.2 Removing restrictions

For partitioning into $p \neq 2^q$ parts, we can use a modified SPLIT subroutine that allows for splitting into two parts by a different ratio than $1:1$.

If we have non-uniform computational weights, we can still efficiently compute the total weight of a (candidate) subvolume. For this, we perform one preprocessing step, and store for each voxel at coordinate (i, j, k) the *cumulative sum* of the cube with lower corner $(0, 0, 0)$ and upper corner



(a) CCB, SAPB (b) DAPB (c) HCB



(d) LAM (e) TSYN

Figure 3.6: Schematic overview of the acquisition geometries that we consider. Here, the source trajectory is shown with a fat red line. The center of the detector is assumed to be at the antipodal point, except in (d) where the detector position is shown in blue. In (a) and (b), we indicate both parallel-beam and cone-beam geometries. In (d), the fat points indicate the positions of the detector and source, which are always one half rotation out of phase and move with the same angular velocity.

(i, j, k), requiring only $\mathcal{O}(n)$ memory and time, where n is the number of voxels in the full object volume. When we want to compute the total weight of a cuboid with lower corner (i_1, j_1, k_1) and upper corner (i_2, j_2, k_2), we can retrieve this in $\mathcal{O}(1)$ time using the principle of inclusion–exclusion with the cumulative sums that have been precomputed.

3.4 Results

The 3D acquisition geometries that we study in this work are all commonly used. They are illustrated in figure 3.6, and are listed below. The parameters for these geometries are given in the appendix to this chapter.

1. *Single-axis parallel-beam* (SAPB). The (point) source, conceptually infinitely far away, and the detector rotate in a circular trajectory

around the object. Example uses are tomography at synchrotron sources [Mar+17] and electron tomography [MD09]. In this acquisition geometry, each line is contained in a single slice, making it trivial to partition the volume.

2. *Dual-axis parallel-beam* (DAPB). Similar to SAPB, but after completing one circle, an alternative axis is chosen and another rotation is made [Mas97; Pen+95]. This acquisition geometry is commonly used in imaging for life sciences.

3. *Circular cone-beam* (CCB). Similar to SAPB, but the source is at some fixed distance. We distinguish between two cases (a) *wide*: the source is close to the sample. Here, wide means that the cone angle is large. (b) *narrow*: the source is far away, which is closer to the parallel-beam case. Circular cone-beam is the usual acquisition geometry for laboratory CT scanners.

4. *Helical cone-beam* (HCB). Here, the setup is the same as for CCB, but the source and detector also move along the rotation axis. This corresponds to a helical trajectory. Helical cone-beam is often used in a medical setting, but it is also used for the analysis of rock samples [She+14].

5. *Laminography* (LAM). The source and detector array follow *different* circular trajectories which are parallel to, say, the $z = 0$ plane. The source and central point on the detector are always one half rotation out of phase, and move with the same angular velocity [MPS10]. Laminography is a common technique for imaging flat objects such as paintings or semiconductor wafers.

6. *Tomosynthesis* (TSYN). The detector array is placed statically under a sample, while the source follows a circular trajectory around a given axis for some limited arc. Among other applications, it is used for breast cancer screening, and the inspection of passenger luggage [Hel10; Rei+11].

3.4.1 Resulting partitionings

For each geometry, we have run the GRCB algorithm for a varying number of processors. We consider processor counts between 16 and 256, and for each geometry we compare against a 1D block partitioning of the volume, which we will call the *standard partitioning*. In this standard partitioning, equal *slabs* of adjacent slices along one of the three dimensions are distributed among the processors, which is current practice for distributed-memory methods in tomography [Pal+17; Ros+13]. Because the vast majority of acquisition geometries have a preferred direction, this partitioning serves as a better base case than, e.g., performing a recursive bisection along the longest dimension. For an example of a standard partitioning, see the resulting GRCB partitioning of the SAPB acquisition geometry in figure 3.7(a) which happens to coincide with the standard partitioning.

We note that we expect the GRCB partitionings to be valid also for ultra-high resolutions, as long as the geometric structure does not change significantly. We chose to keep the problem sizes limited to object volumes consisting of 512^3 voxels to allow our experiments to be done in reasonable time. We employ a simple DIM for the evaluation, that attributes equidistant sampling points completely to the *closest* voxel.

We have always chosen the axis for the standard partitioning that gives the lowest communication volume. The load imbalance for GRCB partitioned object volumes is kept under $\epsilon_{\max} = 0.05$. We do not assume constant weights, and use the cumulative sum approach outlined before. We summarize the results in table 3.2. We visualize the resulting partitionings for $p = 64$ in figure 3.7. A 3D animation visualizing the partitionings and associated acquisition geometries is available as supplementary material to the publication on which this chapter is based. Each part is given a separate color, but because of the high number of parts, some colors may look similar. It is immediately clear from table 3.2 that when considering a large number of processors, which also implies more freedom in having partitionings with rich structures, a large reduction in communication volume can be obtained by using GRCB partitioned object volumes.

The negative gains for the helical cone-beam geometries in the case of low processor counts are most likely caused by the strict load balance constraint we employ. In particular, the standard partitioning is not always balanced. For example, we have computed the load imbalance of

the standard partitioning for HCB_w and HCB_n, and found that it is always above 0.25 for each processor count that we consider. This means that in this case the comparison between a standard and a bisected partitioning is unfair. In fact, it is a benefit of our method that we always end up with well-balanced partitionings.

As already hinted at before, when considering higher processor counts, the structures visible in the partitionings become far richer. We give two examples of partitionings for $p = 256$ processors in figure 3.8 which illustrates this.

An alternative baseline to compare against would be a partitioning in cubes, by splitting the volume into $p = p_0 \times p_1 \times p_2$ equal parts. Because it is unclear in general how to choose (p_0, p_1, p_2), we only consider the special case of $p = 64$ where we can naturally split into $4 \times 4 \times 4$ parts. The resulting communication volumes are shown in table 3.3. For some acquisition geometries, this cube partitioning is an improvement over the standard slab partitioning.

3.4.2 Effects on runtime

To evaluate the effect of the partitioning on the runtime of tomographic reconstruction, we have developed a software package for performing distributed tomographic reconstruction. This *Tomos toolbox* can be found in an online, open-source repository[1]. We have run experiments using Tomos on the Lisa Cluster maintained by SURFsara in Amsterdam. Our communication is implemented using the Bulk library[2], and carried out on top of MPI. The experiments were executed on up to 16 nodes with Intel E5-2650 v2 processors running at 2.60 GHz that have 16 cores each and 64GB of RAM. The nodes were connected using Mellanox FDR InfiniBand.

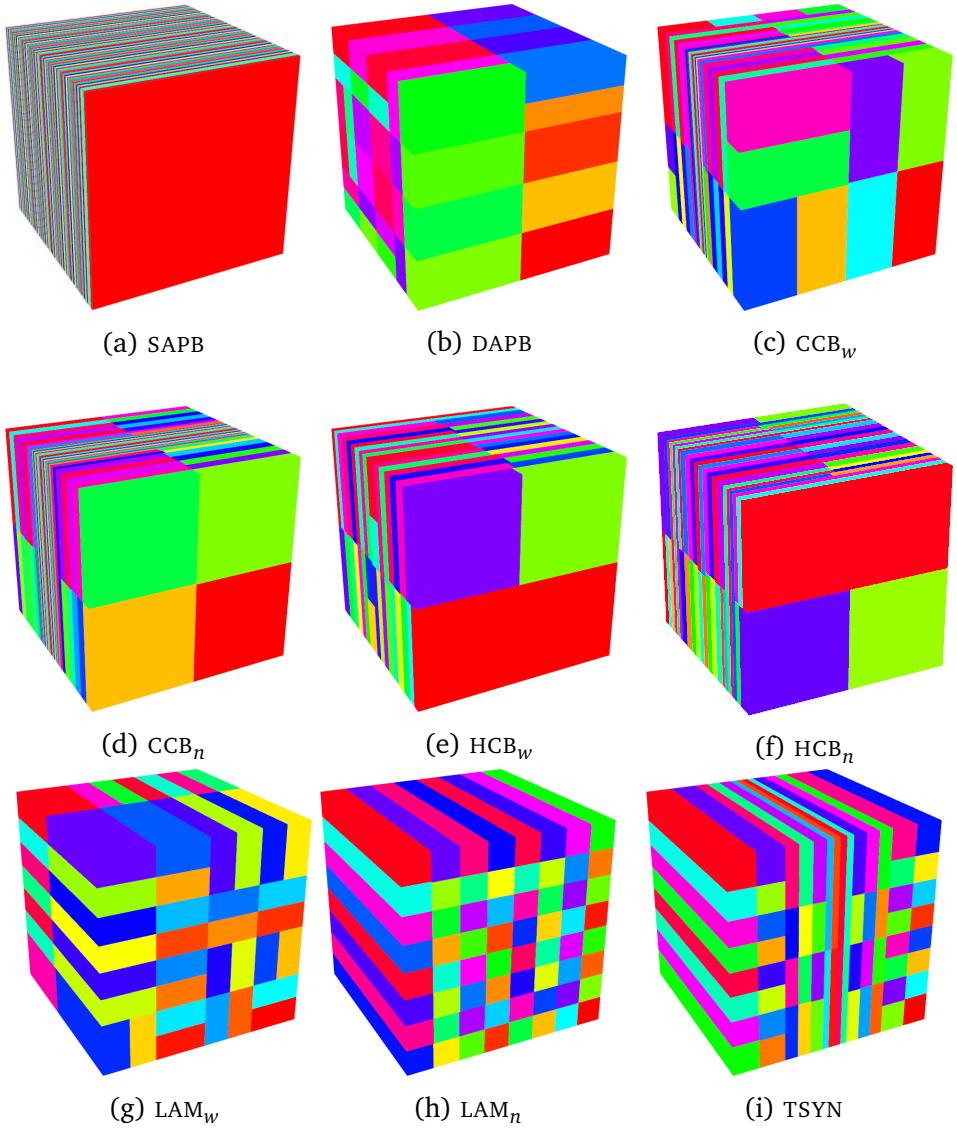
In figure 3.9, we show the effect of the partitioning method on the runtime of a distributed reconstruction algorithm for a varying number of processors. For our results, we use the SIRT reconstruction algorithm. Our evaluation focuses on cone-beam geometries, in particular the CCB_n, HCB_w, LAM_w and TSYN acquisition geometries. The GRCB partitioned object volumes lead to a significant speedup for the reconstruction relative to the

[1] https://www.github.com/jwbuurlage/Tomos/

[2] https://www.github.com/jwbuurlage/Bulk/

$\mathscr{G}$		$p = 16$ ($\times 10^5$)	$p = 32$ ($\times 10^6$)	$p = 64$ ($\times 10^7$)	$p = 128$ ($\times 10^8$)	$p = 256$ ($\times 10^9$)
SAPB	V_{GRCB}	0	0	0	0	0
	V_{STD}	0	0	0	0	0
	g	0.0%	0.0%	0.0%	0.0%	0.0%
	ϵ	0.00	0.00	0.00	0.00	0.00
DAPB	V_{GRCB}	4.9	5.2	6.1	6.5	0.8
	V_{STD}	11.8	19.5	31.6	51.0	10.2
	g	58.7%	73.2%	80.7%	87.2%	92.0%
	ϵ	0.03	0.04	0.05	0.03	0.05
CCB_n	V_{GRCB}	1.1	1.6	1.9	2.3	0.3
	V_{STD}	1.1	1.9	3.2	5.2	1.0
	g	0.1%	16.8%	39.6%	55.8%	69.0%
	ϵ	0.04	0.04	0.05	0.03	0.05
CCB_w	V_{GRCB}	1.9	2.4	2.9	3.2	0.4
	V_{STD}	2.5	4.3	7.1	11.6	2.3
	g	21.5%	44.8%	59.8%	72.0%	81.5%
	ϵ	0.04	0.04	0.05	0.03	0.05
HCB_w	V_{GRCB}	2.3	2.5	2.8	3.3	0.4
	V_{STD}	1.8	2.9	4.7	7.7	1.5
	g	-29.6%	14.3%	40.7%	57.3%	71.0%
	ϵ	0.05	0.04	0.05	0.03	0.05
HCB_n	V_{GRCB}	2.3	2.1	2.3	2.6	0.4
	V_{STD}	1.1	1.8	3.0	4.9	1.0
	g	-104.4%	-12.4%	24.2%	45.7%	62.0%
	ϵ	0.04	0.05	0.05	0.04	0.05
LAM_n	V_{GRCB}	1.4	1.9	2.2	2.7	0.4
	V_{STD}	3.7	6.3	10.2	16.6	3.3
	g	62.0%	69.5%	78.1%	83.9%	89.0%
	ϵ	0.00	0.01	0.05	0.03	0.05
LAM_w	V_{GRCB}	2.5	3.3	3.7	3.9	0.6
	V_{STD}	6.2	10.3	16.9	27.3	5.5
	g	60.2%	68.2%	77.9%	85.8%	90.0%
	ϵ	0.00	0.04	0.04	0.03	0.05
TSYN	V_{GRCB}	1.1	1.5	1.8	2.1	0.3
	V_{STD}	2.3	4.0	6.6	10.8	2.2
	g	51.0%	62.5%	72.8%	80.4%	86.6%
	ϵ	0.03	0.02	0.05	0.03	0.05

Table 3.2: Communication volumes for the acquisition geometries under consideration, for a varying number of processors p. The communication volume under the GRCB partitioning is given by V_{GRCB}, while the communication volume under a standard 1D slab partitioning is given by V_{STD}. The *gain* g is defined as $g = (1 - V_{\text{GRCB}}/V_{\text{STD}}) \times 100\%$. The load imbalance of the GRCB partitioned volume is kept under $\epsilon_{\max} = 0.05$, and is given as ϵ. The *closest-voxel* DIM was used.



(a) SAPB (b) DAPB (c) CCB_w

(d) CCB_n (e) HCB_w (f) HCB_n

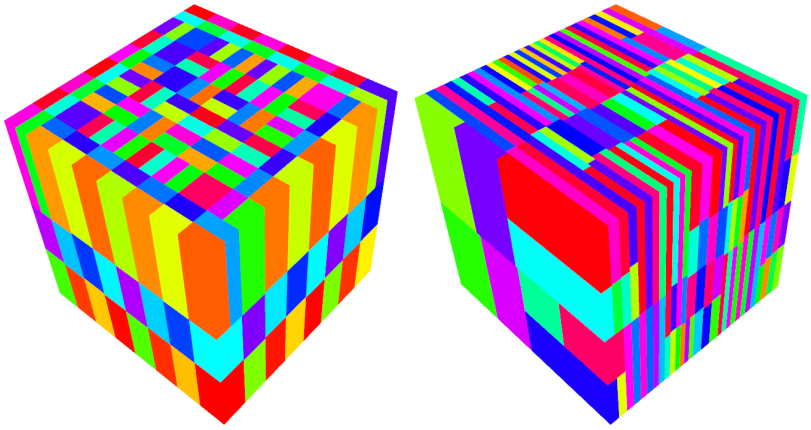
(g) LAM_w (h) LAM_n (i) TSYN

Figure 3.7: Resulting GRCB partitionings for $p = 64$ processors. The axes are as in 

. If there is a main rotation axis, it corresponds to z. For TSYN, the stationary detector is placed perpendicular to the z-axis.

	CCB_n	CCB_w	DAPB	HCB_w	HCB_n	LAM_n	LAM_w	SAPB	TSYN
V_{GRCB}	1.9	2.9	6.1	2.8	2.3	2.3	3.7	0.0	1.8
V_{CUBE}	4.4	4.5	6.2	4.9	4.8	4.1	4.6	6.2	3.8
V_{STD}	3.2	7.1	31.6	4.7	3.0	10.2	16.9	0.0	6.6

Table 3.3: Additional partitioning results, cf. table 3.2. Here, we additionally give the communication volume V_{CUBE} for a partitioning into $p = 64 = 4\times4\times4$ equal parts. Communication volume is given in multiples of 10^7.



(a) LAM_w (b) HCB_w

Figure 3.8: Resulting GRCB partitionings for $p = 256$ processors.

standard slab partitioned object volumes. When isolating the communication times, the effect is even more noticeable, as illustrated in figure 3.10.

In the previous section, we noted the high load imbalance and the relatively low communication volume of the standard partitioning for the HCB_w geometry in case of small p. In the results presented here, we see that indeed the communication time for low processor counts for the GRCB partitioning is higher for HCB_w; however, the *total runtime* of a SIRT iteration is always in favour of the GRCB partitioning since it assures that the computational load is balanced.

When comparing the communication times with the communication volumes shown in table 3.2, one has to take into consideration that the times are not expected to be linearly dependent on the total communic-

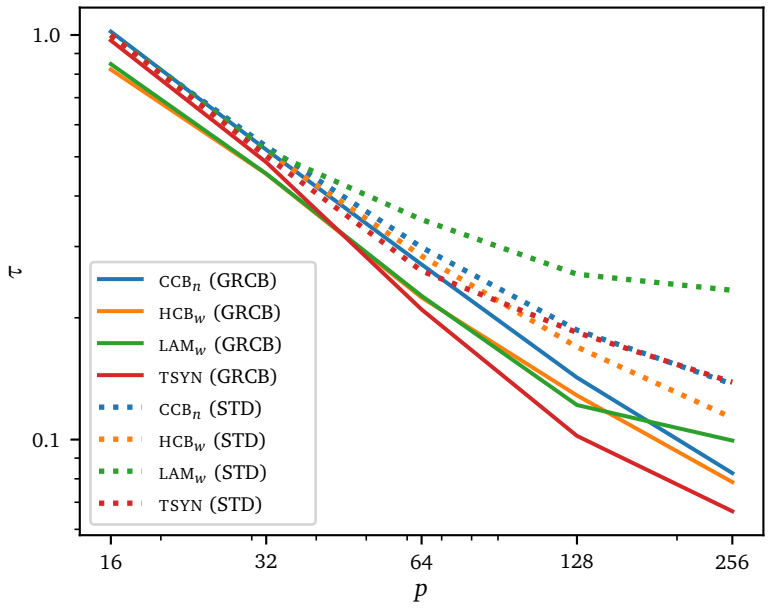


Figure 3.9: The runtime of one SIRT iteration plotted against the number of processors. Vertically, the relative runtime τ is shown on a logarithmic scale, defined for each geometry as the time compared to the runtime of reconstructing using a standard partitioning with $p = 16$ processors. The reconstruction times for the GRCB partitionings are shown using solid lines, and for the standard partitionings using dotted lines. Horizontally, the number of processors is shown on a logarithmic scale. The runtimes for GRCB partitionings with $p = 256$ processors are 18.28, 10.52, 13.57 and 19.58 seconds for CCB$_n$, HCB$_w$, LAM$_w$ and TSYN, respectively.

ation volume. Other important factors are the *maximum communication volume per part*, and the *number of messages* that are sent.

The main assumption we make is that by reducing the total communication volume, and keeping the parts balanced, we also indirectly reduce the communication volume per part and ultimately the total communication time. Based on the results we present, we may conclude that our partitioning method leads to a large decrease in communication time and better scalability, as well as a better load balancing.

The number of messages μ is shown in table 3.4, and is defined as

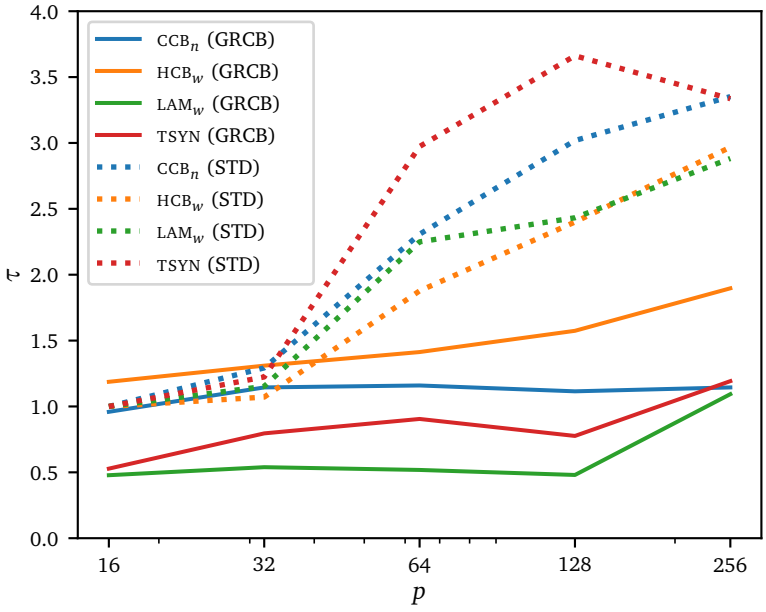


Figure 3.10: The communication time of one SIRT iteration plotted against the number of processors. Vertically, the relative communication time τ is shown, defined for each geometry as the time compared to the communication time for a standard partitioning with $p = 16$ processors. The communication times for the GRCB partitionings are shown using solid lines, and for the standard partitionings using dotted lines. Horizontally, the number of processors is shown on a logarithmic scale. The communication times for GRCB partitionings with $p = 256$ processors are 3.09, 2.94, 5.74 and 4.90 seconds for CCB$_n$, HCB$_w$, LAM$_w$ and TSYN, respectively.

the number of sender–receiver pairs of processors that are communicating with one another during the reconstruction. Our method does not try to reduce the total number of messages, and we observe that the number of messages is of the same order of magnitude for both partitioning methods. In fact, in many cases the number of messages approaches the maximum possible number of messages which is $2p(p-1)$. This seems hard to avoid, since interactions in tomography are global; the rays in the acquisition geometry cross the entire object volume, coupling all the voxels they intersect.

When using the partitionings for distributed reconstruction, only a rep-

		CCB_n	HCB_w	LAM_w	TSYN	$\mu_{\max}$
$p = 16$	μ_{STD}	84	276	360	116	480
	μ_{GRCB}	92	316	360	166	
$p = 32$	μ_{STD}	300	1032	1454	412	1984
	μ_{GRCB}	388	1092	1108	582	
$p = 64$	μ_{STD}	1084	4064	5836	1538	8064
	μ_{GRCB}	1356	4040	4324	2022	
$p = 128$	μ_{STD}	4156	16228	23339	6020	32512
	μ_{GRCB}	4688	14854	14554	6400	
$p = 256$	μ_{STD}	16228	64648	93644	23916	130560
	μ_{GRCB}	17324	53972	47052	18170	

Table 3.4: The message counts for a number of geometries and a varying number of processors. The message count for the standard partitioning is denoted by μ_{STD}, while for GRCB partitioned volumes they are denoted by μ_{GRCB}. The maximum possible number of messages (all-to-all) is given as $\mu_{\max}$.

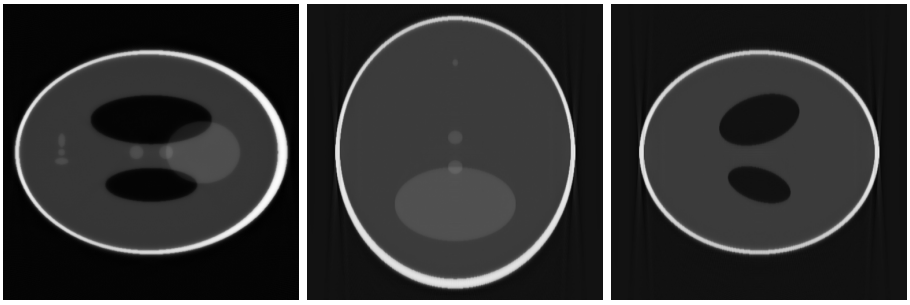


Figure 3.11: Reconstructed slices for an object volume of $512 \times 512 \times 512$ voxels with the CCB_n acquisition geometry using 64 processors. For the reconstruction, 100 iterations of SIRT were applied with a slice-interpolated DIM. Here we used a modified 3D Shepp–Logan phantom. The left, middle, and right reconstructed slices are taken in the middle along the **z**, **x**, and **y** axes respectively.

resentation of the bisectionings has to be stored and loaded. A suitable DIM for the acquisition geometry is chosen independently. To demonstrate that our implementation actually works in practice, we show a reconstruction for CCB_n in figure 3.11.

3.5 Discussion

For our evaluation we used straightforward custom implementations of the projection operations. In a heavily optimized implementation, we expect that the communication times will play an even more important role. In the future, we plan on employing the partitionings found with the GRCB method to improve the reconstruction times for real-world tomographic experiments. This involves combining the partitionings presented in this chapter, with state-of-the-art software for tomographic reconstruction. So far, we have used CPUs for our evaluation, but we plan to use GPUs instead, making computations faster but also making communication relatively even more important.

The load balancing constraint we employ models only the number of nonzeros assigned to each processor, where a nonzero indicates a line–voxel intersection. The actual time spent by a processor in the local forward projection and backprojection steps depends on a number of additional factors. For example: (i) there is an overhead relating to the number of local rows, because the nonzeros are generated instead of stored, (ii) memory access patterns are known to have an important influence, (iii) depending on the chosen DIM the actual nonzero pattern can differ from the one used in our model, (iv) there are effects relating to the system, such as variability between cores and the scheduling of processes. To check the relation between the modelled computational load and the actual runtime, we have measured the time $\tilde{T}^{(s)}$ spent by processor s in the local forward projection step (not including any communication) for the CCB_w geometry. The *runtime imbalance* $\tilde{\epsilon} = \max_{0 \leq s < p} \tilde{T}^{(s)} / \tilde{T}_{\text{avg}} - 1$, was found to be between 0.07 and 0.15, while the load balance $\epsilon_{\max}$ was set to 0.05. A more sophisticated model for the computational load beyond counting the number of local nonzeros may improve the actual achieved runtime balance, but is outside the scope of this work.

With variational reconstruction methods, prior information on the object can be incorporated. A common approach is to include the norm of the image gradient as an additional penalty term. In distributed-memory implementations, evaluating the gradient in every voxel requires the communication of all interfaces between subvolumes. We have not modeled this additional communication in the derivation of our algorithm. For the partitionings presented here, the communication volume due to gradient com-

putations is an order of magnitude lower than the communication volume due to the total line cut for all acquisition geometries except single-axis parallel beam. Therefore, we think it is warranted to ignore this cost in our expression for the communication volume.

In this work, we have assumed a simple network topology, where communication performance is identical between any pair of nodes. However, many modern HPC systems are hierarchical. For example, there could be p_1 nodes, where each node has p_2 processing elements such as CPU cores or GPUs. If we use our unmodified method to partition the object volume into $p = p_1 p_2$ parts, we would not take into account that communication between processing elements residing on the same node is more efficient.

We will sketch how, by a straightforward modification of the load balance constraints used in the algorithm, a suitable partitioning can be found for hierarchical systems. The idea is to allow a relatively large load imbalance between the nodes, resulting in low inter-node communication volume, and to pay for this by imposing a smaller load imbalance within a node, at the cost of a potentially higher intra-node communication volume. In the first stage, the partitioning algorithm is used to split the volume into p_1 parts using a load imbalance $\epsilon_1 = \gamma\epsilon$. Here, $0 < \gamma < 1$ relates to the ratio between the inter-node and intra-node communication cost. After this first stage, each of the p_1 parts are partitioned independently into p_2 parts by the same algorithm. For the second partitioning stage, a part-dependent load imbalance $\epsilon_2(s)$ will ensure that the resulting load imbalance is at most ϵ. How to choose γ to optimally exploit a two-level memory hierarchy requires further study that is beyond the scope of the present work.

3.6 Conclusion

We consider distributed-memory tomographic reconstruction and introduce a tomographic partitioning problem (TOMPP). We present GRCB, a partitioning method to solve this problem, that considers the underlying geometry of the tomographic reconstruction. This is in contrast to combinatorial partitioning methods that are based solely on the nonzero pattern of the corresponding sparse matrix. Our method can be applied to arbitrary acquisition geometries. We show that with our new method, we can

reduce the necessary communication in distributed-memory parallel tomographic reconstruction and improve the scalability of an important class of reconstruction algorithms, including SIRT, CGLS and other Krylov methods, ML-EM, FISTA and Chambolle–Pock.

Proofs

proof of lemma 2. It suffices to show that a line intersects both subvolumes if and only if it has a non-empty intersection with (or *crosses*) the interface between them. If a line is contained in the interface, then the statement holds since it crosses the interface, and it intersects both subvolumes. Assume the line is not contained in the interface. Say that a line intersects both $\mathcal{V}_0$ and $\mathcal{V}_1$, then there exist points $a \in \mathcal{V}_0$ and $b \in \mathcal{V}_1$ that are both on the line. Because cuboids are convex, the line segment from a to b (which is contained in the original line) is entirely in $\mathcal{V}$, and starts in $\mathcal{V}_0$ while it ends in $\mathcal{V}_1$. Therefore, it has to cross the interface. Conversely, if a line crosses the interface at a point c, then we immediately have $c \in \mathcal{V}_0$ and $c \in \mathcal{V}_1$ so that the line intersects both subvolumes. □

proof of theorem 3. Since we can no longer assume that each line intersects the full volume in each term, we define

$$\lambda'_\ell(\pi) = \max(\lambda_\ell(\pi) - 1, 0),$$

so that

$$V = \sum_{\ell \in \mathcal{G}} \lambda'_\ell(\pi).$$

In other words, if ℓ crosses the volume to be split, $\lambda'_\ell(\pi)$ is the number of subvolumes crossed by ℓ minus one, otherwise it is zero. It is enough to consider each term, corresponding to individual lines, separately. We have to show:

$$\lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-1}) = \lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-2} \cup \mathcal{V}_{p-1}) + \lambda'_\ell(\mathcal{V}_{p-2}, \mathcal{V}_{p-1}).$$

We will split the proof into two cases. If a line does not intersect $\mathcal{V}_{p-2} \cup \mathcal{V}_{p-1}$, then both sides equal $\lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-1})$.

If it does intersect $\mathcal{V}_{p-2} \cup \mathcal{V}_{p-1}$, then we have two subcases corresponding to the line intersecting either both $\mathcal{V}_{p-2}$ and $\mathcal{V}_{p-1}$, or one of the two. For the former, we have:

$$\begin{aligned}\lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-2} \cup \mathcal{V}_{p-1}) + \lambda'_\ell(\mathcal{V}_{p-2}, \mathcal{V}_{p-1}) &= \lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-3}) + 1 + 1 \\ &= \lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-3}, \mathcal{V}_{p-2}, \mathcal{V}_{p-1})\end{aligned}$$

as required. For the latter, we assume without loss of generality that it intersects $\mathcal{V}_{p-2}$ and compute

$$\begin{aligned}\lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-2} \cup \mathcal{V}_{p-1}) + \lambda'_\ell(\mathcal{V}_{p-2}, \mathcal{V}_{p-1}) &= \lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-3}) + 1 + 0 \\ &= \lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-3}, \mathcal{V}_{p-2}) \\ &= \lambda'_\ell(\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-3}, \mathcal{V}_{p-2}, \mathcal{V}_{p-1})\end{aligned}$$

which finishes the proof. □

Parameters of the acquisition geometries

$\mathcal{G}$	k	$\mathbf{s}$	$\mathbf{d}$	D	φ	r_s	r_d	ϑ
SAPB	512			$(1.0, 1.0)$				
DAPB	512			$(1.0, 1.0)$				
CCB_n	768	$(-5.0, 0.5, 0.5)$	$(4.0, 0.5, 0.5)$	$(2.0, 2.0)$				
CCB_w	768	$(-2.0, 0.5, 0.5)$	$(2.0, 0.5, 0.5)$	$(2.0, 2.0)$				
HCB_w	512	$(-3.0, 0.5, 0.5)$	$(4.0, 0.5, 0.5)$	$(2.0, 2.0)$	4π			
HCB_n	512	$(-5.0, 0.5, 0.5)$	$(6.0, 0.5, 0.5)$	$(2.0, 2.0)$	4π			
LAM_n	512	$(0.5, 0.5, 3.0)$	$(0.5, 0.5, -2.0)$	$(2.5, 2.5)$		0.5	0.5	
LAM_w	512	$(0.5, 0.5, 3.0)$	$(0.5, 0.5, -2.0)$	$(2.5, 2.5)$		1.0	1.0	
TSYN	768	$(0.5, 0.5, 3.0)$	$(0.5, 0.5, -1.0)$	$(2.0, 2.0)$				0.7

Table 3.5: Parameters of the acquisition geometries used for partitioning the volume. In all cases, the physical extent of the object volume is $[0, 1]^3$ and the number of voxels is 512^3. The number of projections is always 512. Positions are given in (x, y, z) coordinates. An empty field means that the parameter is not applicable for that geometry. The primary rotation axis is always the z-axis, except for TSYN where it is the x-axis. For DAPB the second rotation axis is the x-axis. Angles are given in radians. With k we denote the number of rows and columns on the detector. The source and detector are positioned at $\mathbf{s}$ and $\mathbf{d}$ respectively. The size of the detector is denoted by D. With φ we denote the total rotation angle, i.e., $\varphi = 4\pi$ means two full revolutions are made in the helical geometries. With r_s and r_d we respectively denote the radius of the source circle and detector circle for laminography. With ϑ we denote the total arclength of the source movement in tomosynthesis.

Chapter 4

A projection-based partitioning method

With tomographic techniques, the interior of an object can be imaged without destroying the object, which makes tomography an important tool in medicine, industry, and science. Using a beam of penetrating radiation, consisting of, e.g., photons or electrons, two-dimensional projections of an object are acquired. These projections can be related to integrals of some volumetric property of the object, such as its density. *Computed Tomography* (CT) is a technique to retrieve a 3D profile of this property from the measured projection images [Buz08; Avi01].

A tomographic experiment is performed using a *source* that emits the penetrating radiation, and a two-dimensional *detector* that captures the projection images. A finite number of projections are taken of the object. In this chapter, we will consider *point sources*, and rectangular *flat panel detectors*. This means that each projection corresponds to a cone, with at its base the detector, and at its apex the source.

Two important operations in CT algorithms are the *forward projection*

This chapter is based on:

A projection-based partitioning method for distributed tomographic reconstruction. *JW Buurlage, WJ Palenstijn, RH Bisseling, KJ Batenburg*. Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing, 58–68, 2020

and the *backprojection*. A forward projection operation is a linear transformation that models the physical experiment. It takes a discretized representation of the object, and outputs the two-dimensional projections of the object. The *backprojection* operator is the adjoint of the forward projection operator. Various models can be used for this linear transformation [LFB10; MB04; XM06].

There are a broad variety of reconstruction algorithms for CT. An important subset of these algorithms uses forward projection and backprojection operations, and these operations typically dominate their runtime costs. Our focus in this chapter is on reconstruction methods that alternate between forward projection and backprojection operations, with optionally some in-between operations in the image or measurement domain. These include SIRT [Gil72], Krylov methods such as CGLS [HS52], ML-EM [LC+84], and methods originating from convex optimization such as FISTA [BT09] and Chambolle–Pock [CP10].

The computational cost of these reconstruction methods grows superlinearly with respect to the input data. The size of typical tomographic data sets is rapidly increasing, due to advances in hardware and increased interest in multi-modal imaging, imaging of dynamic systems, and adaptive acquisition. Large data sets of many GBs in size are increasingly common, and for these data sets even optimized GPU implementations do not always suffice to keep the computational costs manageable. This motivates the move to large distributed-memory compute clusters, to keep reconstruction times reasonable.

When performing projection operations on a distributed-memory system, communication is the main bottleneck for algorithms that make use of alternating forward projection and backprojection operations. The data partitioning method presented in this chapter concerns itself with minimizing the required communication, without changing the overall structure of the underlying algorithms, for an arbitrary acquisition geometry, i.e., a set of source and detector positions. It is a refinement of the previously published GRCB partitioning algorithm [BBB19].

While the GRCB method has a good time complexity compared to, e.g., the projection operations, it is still too slow to apply in real time. This limits its applicability in various situations, such as adaptive acquisition where the user may want to zoom in on a region-of-interest after initial inspection, or in cases where the acquisition geometry simply changes from

scan to scan, because the user changes, e.g., the source-to-object distance, or the source-to-detector distance.

This chapter is structured as follows. In Section 4.1, we discuss how to model tomographic reconstruction as a linear inverse problem, discuss an associated partitioning problem, and summarize the original GRCB partitioning method. In Section 4.2, we introduce a geometric characterization of the partitioning problem, and use this to develop a more efficient partitioning algorithm. In Section 4.3, we introduce a memory-efficient data structure that stores communication metadata. In Section 4.4, we give the results of our numerical experiments. Finally, in Section 4.5, we present our conclusions.

4.1 Background

Tomographic reconstruction.

Tomographic reconstruction can be modeled as a linear system of equations. The physical model is discretized in order to obtain a matrix $W \in \mathbb{R}^{m \times n}$, that maps a discretized representation $\mathbf{x} \in \mathbb{R}^n$ of the object (the *image*), to a vector of measurements $\mathbf{b} \in \mathbb{R}^m$. A component x_j corresponds to the jth voxel in the volume. A component b_i corresponds to a measurement for the ith ray, between a source point and a pixel on the detector. The reconstruction problem in tomography is a linear inverse problem of the form: given W and $\mathbf{b}$, find $\mathbf{x}$ such that:

$$W\mathbf{x} \approx \mathbf{b}.$$

In order to construct the *system matrix* W we introduce two concepts: the *acquisition geometry*, and the *object volume*. The acquisition geometry is a set of line segments in three-dimensional space. For each projection, where the radiation source and detector positions are fixed, each *detector element* on the detector (corresponding to a pixel in the projection image), is the end point of a line segment that starts at the position of the source. The imaged object is represented as a discretized volume of *voxels*. Each voxel corresponds to a small cube, and the associated value x_j corresponds to some volumetric property of the object, such as its density, at the location of the voxel.

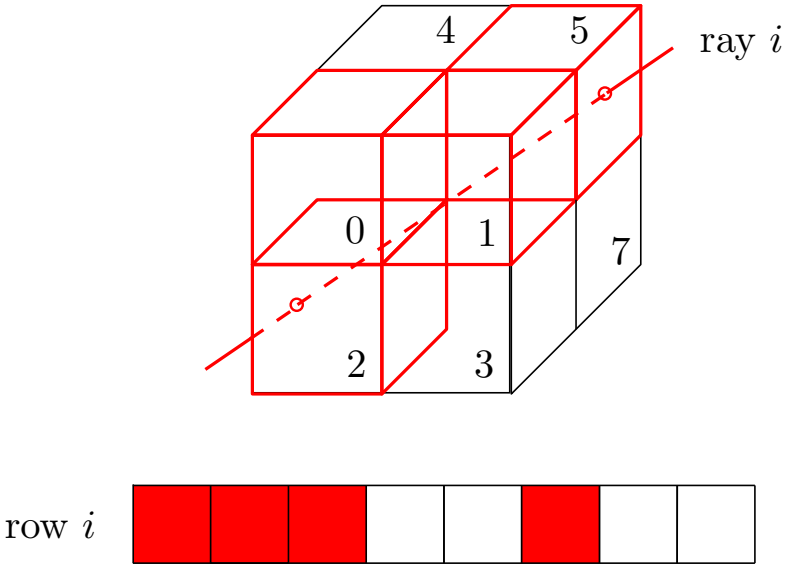


Figure 4.1: Constructing a row of the system matrix W. The object volume is discretized into $2 \times 2 \times 2$ voxels, and a ray from the acquisition geometry intersects this volume. Here, it passes through four of the numbered voxels, the ones marked red, leading to four nonzeros in the corresponding matrix row.

We do not consider parallel-beam geometries, where conceptually the source is infinitely far away, as they are usually easier to partition. However, the method we present should generalize to those geometries as well.

Each row of the matrix W corresponds to a line segment in the acquisition geometry. Each column of W corresponds to a voxel of the object volume. We assume that the matrix elements W_{ij} are given by the length of the intersection of the ith line with the jth voxel. Note that W is sparse, as each line will only intersect a relatively small collection of voxels. This construction is illustrated in figure 4.1.

The forward projection and backprojection operations that are crucial for many reconstruction algorithms, correspond to sparse matrix–vector (SpMV) products with W and W^T, respectively.

Parallel execution of projection operations.

When the sparse matrix–vector products $\mathbf{y} = W\mathbf{x}$ and $\mathbf{x} = W^T\mathbf{y}$ are executed on a distributed-memory system that consists of p nodes (or *processing elements*, or simply *processors*), communication between the nodes is the single most important consideration for the computational efficiency. The relevant data are the nonzeros of the matrix W, the components of the image $\mathbf{x}$, and the components of the measurements $\mathbf{y}$. For each of these three types of data, a suitable p-way partitioning has to be chosen.

The sth part of the data, is assigned to the sth processor. The three types of data: the image, measurements, and nonzeros, correspond to three ways of partitioning the underlying sparse matrix. An image partitioning implies a *column partitioning* of the matrix, a measurement partitioning implies a *row partitioning* of the matrix, and finally a nonzero-based partitioning gives a *2D partitioning* of the matrix.

Communication occurs because different processors depend on the same data. Each nonzero W_{ij} corresponds to two floating-point operations (flops), as it has to be multiplied with image component x_j and the result of this multiplication occurs in the sum for the measurement component $y_i = \sum_{W_{ij}\neq 0} W_{ij}x_j$. In other words, a nonzero element W_{ij} couples the jth component of $\mathbf{x}$ and the ith component of $\mathbf{y}$. Communication is usually unavoidable if one requires a balanced partitioning where each part is of roughly equal size, but by choosing a suitable partitioning the total *communication volume*, i.e., the number of data words sent, can be reduced significantly. The components of the vectors $\mathbf{x}$ and $\mathbf{y}$ must also be assigned to a processor, without any restriction. In that case, the parallel algorithm will have four phases: (i) a *scatter* phase where each component x_j is communicated to the processors that need it; (ii) a local computation of products $W_{ij}x_j$ followed by an addition of products for the same row i; (iii) a *gather* phase where the contributions to each component y_i are communicated to the owner of the component; (iv) a local addition of the received contributions for each component y_i.

Partitioning for SpMVs is a well-studied problem in combinatorial scientific computing. The underlying structure is modeled as a hypergraph, where common models include *row-net* and *column-net* [CA99], *medium-grain* [PB14], and *fine-grain* [CA01]. Partitioning methods aim to find a balanced partitioning of the vertices of the model hypergraph, that minim-

izes the total communication volume and in certain cases also the number of messages sent.

The system matrix for a tomographic reconstruction problem is sparse and consists of $\mathcal{O}\left(mn^{1/3}\right)$ nonzeros, and common values for m and n are 10^9 or even higher. This corresponds to many terabytes of data, which means that the matrix cannot be stored explicitly for the desired high resolutions, even when employing a sparse data structure, and that the forward and backprojection must be implemented in a matrix-free manner. This also means that it is not at all clear how SpMV partitioning approaches can be applied. Instead, we consider the underlying geometry of the problem.

Tomographic partitioning problem.

In tomographic reconstruction, a cuboid region $V \subset \mathbb{R}^3$ called the *object volume* is defined. The sample being scanned is completely contained in V, and after discretizing the object volume into $n = n_x \times n_y \times n_z$ voxels, the sample can be represented using an image $\mathbf{x}$ with one component for each voxel. For distributed-memory tomographic reconstruction, we choose to find a suitable partitioning of the object volume V, which after discretizing gives a partitioning of the image $\mathbf{x}$, corresponding to a column partitioning of the matrix. The relevant part of W can be generated locally on each processor. Only contributing partial sums for the projection data have to be communicated during the projection operations.

The quality of a partitioning is judged on two grounds: the amount of communication it induces, and whether or not the parts are roughly equal in terms of computational cost.

Instead of considering the nonzeros, we can look at the problem geometrically. A tomographic measurement consists of a number of projections, and for each projection we consider the line segments from the source position to each pixel on the detector. This defines a set of line segments $\mathcal{G}$ that we call the *acquisition geometry*. Communication is required for each line in the acquisition geometry that travels through multiple parts of the image volume. The number of parts a line ℓ crosses for a partitioning π is denoted by $\lambda_\ell(\pi)$. Since we can designate one of the parts as the

owner of the line, we have for the communication volume:

$$\Lambda(\pi) = \sum_{\ell \in \mathcal{G}} (\lambda_\ell(\pi) - 1).$$

For a good partitioning π, this value will be manageably small.

The computational cost of a part is modeled as the number of flops it has to perform in a projection operation. Each voxel is involved in twice as many flops as there are lines $\ell \in \mathcal{G}$ crossing the voxel. For the jth voxel, we write $\omega(j)$ for the number of lines crossing the voxel. The total computational weight of the sth part is then given by:

$$T^{(s)} = \sum_{j\,:\,x_j \in V_s} \omega(j).$$

Here, the notation $x_j \in V_s$ indicates that the voxel x_j is assigned to the sth part after discretizing. For a good partitioning, the following *load imbalance* ϵ should be kept small:

$$\epsilon(\pi) = \max_{0 \le s < p} \frac{T^{(s)}}{T_{\text{avg}}} - 1,$$

where $T_{\text{avg}} = \sum_s T^{(s)}/p$. We can summarize the tomographic partitioning problem as follows.

Definition 4. Let $\mathcal{G}$ be an acquisition geometry, $\mathcal{V}$ the object volume, $\epsilon_{\max}$ the maximum allowed load imbalance, and p the number of processors. Let Π denote the set of all p-way volume partitionings. The tomographic partitioning problem is the following optimization problem:

$$\begin{aligned} \textit{minimize}_{\pi \in \Pi} \quad & \Lambda(\pi) \\ \textit{subject to} \quad & \epsilon(\pi) \le \epsilon_{\max}. \end{aligned}$$

Geometric recursive coordinate bisectioning.

The GRCB algorithm only looks at partitionings π that are obtained by recursive coordinate bisectioning. That is to say, the volume is recursively split $p-1$ times, each time along one of the axes. Axis-aligned cuboid partitionings such as the ones obtained by GRCB are convenient in practice,

and can be expected to give reasonably good results. Because the communication volume is additive (see theorem 2 in [BBB19]), bisectioning can be done independently for each part, which is why we can obtain a good partitioning for any number of processors by recursively splitting in two.

The splitting subroutine of GRCB that performs the bisectioning, is based on a plane sweep. We are able to identify which splitting plane, among all the possible axis-aligned ones, is able to best limit the communication volume by directly considering all the lines in the acquisition geometry $\mathscr{G}$.

4.2 A new projection-based partitioning method

The GRCB algorithm uses a discrete model for the acquisition geometry, explicitly considering a set of rays. While this leads to an exact representation of computation load (in flops) and communication volume (in data words), it does mean that the input data sizes for the partitioning method are large.

Here, we take a different approach and use a continuous model for the acquisition geometry, communication volume, and computational load. For fine enough resolutions, we expect the discretization error incurred by this model to be small. Instead of minimizing the communication volume subject to a load balance constraint, we now aim to minimize the communication volume and the load imbalance simultaneously by generating a candidate split for each of the three coordinate axes on the basis of load balance, and among these candidate splits choose the one that realizes the lowest communication volume.

As before, the object volume is a cuboid $\mathscr{V} = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \subset \mathbb{R}^3$ that we want to partition into p parts. We limit ourselves to partitionings obtained by recursive bisectioning. In this chapter, the faces of a cuboid are considered part of the cuboid.

The acquisition geometry is modeled as a set P of cone-shaped projections p_k. Each projection p_k can be described by a source–detector pair. The point-source is at position $\mathbf{s}_k \in \mathbb{R}^3$. The detector is a rectangular region $D_k \subset \mathbb{R}^3$. The cone with base D_k and apex $\mathbf{s}_k$ defines the projection p_k.

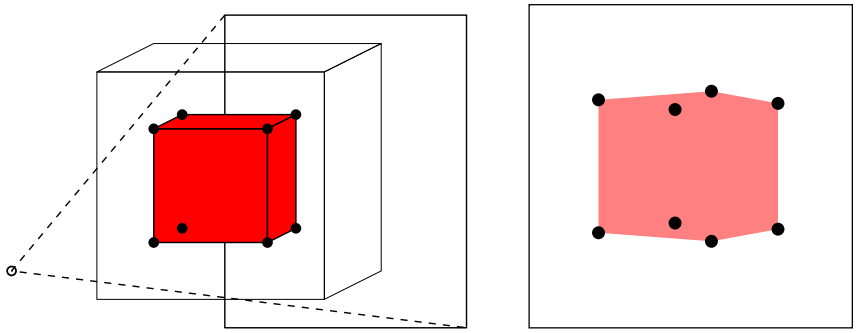


Figure 4.2: The shadow of a part with respect to the point source defines the region on the detector for which line segments cross the part. Here, the part and its shadow are shown in red. The shadow can be computed by projecting the eight vertices of the cuboid on the detector, and then taking their convex hull.

4.2.1 Communication volume.

We consider the effect that one of the projections $p_k = (\mathbf{s}_k, D_k)$ has on the communication volume. In the discrete model, the volume depends on the resolution on the detector, i.e., the *shape* in pixels of the detector D_k, e.g., 2000×2000. For each detector pixel with center $\mathbf{d}_k^{(i)}$, we consider the line segment ℓ from $\mathbf{s}_k$ to $\mathbf{d}_k^{(i)}$. The number of cuts in ℓ, which is the number of additional parts of the object volume that it intersects, is the contribution in number of data words to the communication volume. Note that we determine a single p-way partitioning of the object volume for the set of all rays from all projections.

We describe here a new approach that works directly on the cones defined by the projections, rather than the individual pixels. It is therefore independent of the detector discretization, and this greatly reduces the size of the input data to the partitioning algorithm.

We exploit the fact that line segments corresponding to neighbouring pixels often cross the same parts. We want to group rays by identifying pixels in a region of the detector for which the corresponding line segments all cross exactly the same parts. The key observation that makes this possible is that a region of the detector for which the line segments cross a given part of the object volume, corresponds to the shadow of that part onto the detector. This is illustrated in figure 4.2.

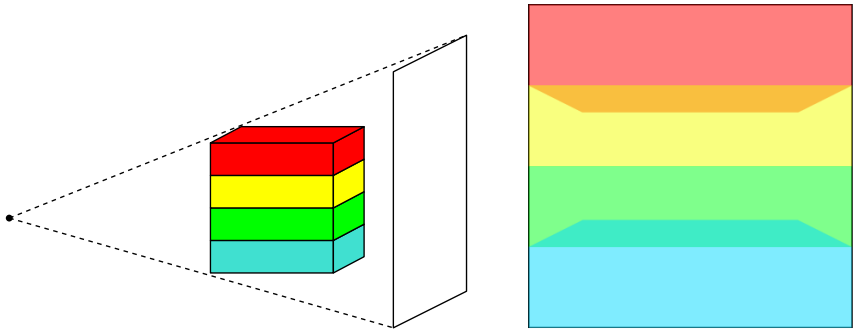


Figure 4.3: Where shadows of a part overlap, line segments in that region cross multiple parts.

The communication volume in our continuous model is estimated in the following way. We consider a candidate split into two parts. Strategies to generate these candidate splits are discussed later. This split happens along one of the axes of the object volume, at a given location. The split induces two subvolumes, one to the *left* of the splitting plane, and one to the *right*. To identify the region on the detector for which the line segments cross both parts, we forward project the vertices of these subvolumes onto the detector. The shadow of each subvolume can be found by taking the convex hull of its projected vertices. The *area* of the intersection of the two shadows is proportional to the number of line segments crossing both parts for any fine enough discretization of the detector. We compute this for each projection in P in order to find the total communication volume.

A subroutine for computing the communication volume for any candidate split of the volume V into a left part V_L and a right part V_R is given in algorithm 5, optionally taking into account volume for a gradient-based regularizer as discussed in section 4.2.3.

Because the communication volume is additive, we can split the volume recursively. After $p-1$ splits, we have obtained a partitioning into p parts. The interplay between shadow intersections and communication volume for a fixed projection is illustrated in figure 4.3.

Algorithm 5 Computing the communication volume for a given split. Here, M is a magnification value, relating the detector size to the object volume size, and V_L and V_R are cuboids corresponding to the volumes to the left and to the right of the candidate splitting plane.

Subroutine: COMMUNICATIONVOLUME
Input: V_L, V_R, P
Output: Λ

$\Lambda \leftarrow 0$
for all $p_k \in P$ **do**
  $S_L \leftarrow$ CONVEXHULL(PROJECT(p_k, VERTICES(V_L)))
  $S_R \leftarrow$ CONVEXHULL(PROJECT(p_k, VERTICES(V_R)))
  $\Lambda \leftarrow \Lambda +$ AREA($S_L \cap S_R$)

if consider gradient **then**
  $\Lambda \leftarrow \Lambda + M \times$ AREA($V_L \cap V_R$)

4.2.2 Load balance.

We next discuss generating a set of candidate splits that we want to evaluate. These candidate splits should divide the object volume into parts with roughly equal computational weight, and among that set we choose the one that induces the least amount of communication.

Modeling the computational weight in our continuous setting does not appear to be as straightforward as for the communication volume. Recall that the computational weight of a voxel is defined as the number of lines intersecting it. We no longer have an explicit set of lines nor of voxels, but regardless of the discretization we have that the line density in the volume for a given projection decreases as $1/r^2$ where r is the distance to the source. The computational weight of a part V_s should therefore be proportional to the integral:

$$\sum_{k=1}^{|P|} \int_{V_s} \frac{1}{||\mathbf{x} - \mathbf{s}_k||_2^2} d\mathbf{x}. \tag{4.1}$$

If we want to split along, say, the x axis, into two parts with equal compu-

tational weight, then we want to find $c \in [x_1, x_2]$ so that

$$\int_{x_1}^{c}\int_{y_1}^{y_2}\int_{z_1}^{z_2}\sum_{k=1}^{|P|}\frac{1}{||\mathbf{x}-\mathbf{s}_k||_2^2}\,dz\,dy\,dx = \int_{c}^{x_2}\int_{y_1}^{y_2}\int_{z_1}^{z_2}\sum_{k=1}^{|P|}\frac{1}{||\mathbf{x}-\mathbf{s}_k||_2^2}\,dz\,dy\,dx.$$

The volume integral for a rectangular volume $V = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$ can be written as the following 2D integral:

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2}\sum_{k=1}^{|P|}\Big(\frac{1}{a_k(x,y)}\Big(\arctan\Big(\frac{z_2-s_{k,z}}{a_k(x,y)}\Big) - \arctan\Big(\frac{z_1-s_{k,z}}{a_k(x,y)}\Big)\Big)\Big)\,dy\,dx, \tag{4.2}$$

where

$$a_k(x,y) = \sqrt{(x-s_{k,x})^2 + (y-s_{k,y})^2}.$$

This is, of course, more efficient to solve numerically compared to the original three-dimensional problem.

For finding c, we use the following strategy. We take N samples in the volume V. Next, we choose c such that

$$(c-x_1)\bar{f}_L = (x_2-c)\bar{f}_R,$$

where $\bar{f}_L$ is the average of the integrand in (4.1), or the more efficient variant in (4.2), for samples with an x-coordinate smaller than c, and $\bar{f}_R$ for the remaining samples. We find the optimal c by sorting the N samples by their x-coordinate, and performing a linear scan while updating the averages to the left and right of c. It is possible to decide on the number of samples N dynamically, by updating c for each new sample, and taking samples until the optimal value for c converges.

A difficulty is introduced for acquisition geometries where, because of a limited detector size, or a source that is close to the object, the object volume is not contained in the cones defined by the projections. In these cases, we want to integrate over the intersection of the cone and the volume. This can be easily realized by rejecting samples for a projection

p_k if the sample projects to a point outside of the detector. For these acquisition geometries, we cannot employ the analytical reduction from 3D to 2D shown in (4.2).

As an alternative to approximating the above integrals numerically, we can employ a simpler strategy to identify valid candidate splits. We still consider each axis in turn. If we split *in the middle* along a given axis, we end up with two parts that are equal in volume and should thus have the same number of voxels (up to discretization errors). If the number of lines intersecting a voxel is more or less constant throughout the volume, the number of voxels is one way to achieve a reasonable load balance.

Solving the numerical integraton problem, or using the splitting in the middle strategy (which we will refer to as MIDWAY in our experiments), both result in three candidate splits, one for each axis. Out of these three candidate splits, the best one is chosen each time, based on communication volume.

4.2.3 Image gradient computations.

Image gradient computations form an optional component of a number of reconstruction methods. Prior information on the object, such as the object being piecewise constant, or being smooth, can be incorporated as a penalty term involving the norm of the image gradient. In these cases, tomographic reconstruction is performed by solving a regularized least-squares problem. For example, for TV regularization we have:

$$\arg\min_{\mathbf{x}\in\mathbb{R}^n} ||\mathbf{b} - W\mathbf{x}||_2^2 + \lambda|| \, |\nabla\mathbf{x}| \, ||_1. \tag{4.3}$$

To perform (discrete) gradient computations, each processor requires the value of the neighbouring voxels to each of its voxels. This means that values for voxels that lie next to the splitting plane have to be obtained from a remote processor. In previous work, this communication cost was ignored in the partitioning algorithm. However, it is straightforward to include this as a term in the communication volume, by considering the area of the splitting plane in addition to the area of the shadow intersections.

Both the area of the splitting plane, and the area of the shadow intersections on the detector are proportional to their respective communication weights, but by a different coefficient. Therefore, the areas should be normalized, so that they can be compared to each other. The discretization

on the detector should take into account the total area of the detector, and the discretization of the object volume should in turn take into account its total volume.

In particular, discretization is commonly chosen so that if a voxel in the volume has a cross-section of area X, then the area of its shadow Y corresponds roughly to the size of a detector pixel. We will use the magnification value $M = Y/X$ to relate the communication volumes due to gradient computations and due to an SpMV.

In our new algorithm, this communication volume for gradient computations is optionally taken into account. When splitting a part that is elongated in some direction, the cross-section (area of the splitting plane) will depend on the axis chosen, and this can influence the resulting partitioning.

4.3 Communication data structures

In this subsection, we will discuss how to use the partitionings efficiently in practice. The partitionings aim to minimize the communication volume, while evenly sharing the work among the processors. However, performing the communication requires storing information on what gets sent where during the execution.

The iterative algorithms that are the focus of this work, perform alternating forward projection and backprojection operations. During a forward projection, that is the calculation of $\mathbf{y} = W\mathbf{x}$, *contributions* to the components of $\mathbf{y}$ are computed by the processors whose part of the object volume is crossed by the line segment corresponding to that component. Therefore, each component of $\mathbf{y}$ has one or more *contributing processors*. One of these contributing processors is designated as the owner of the component. The owner computes the sum of the contributions. Before a backprojection $\mathbf{x} = W^T\mathbf{y}$, this sum is distributed to the group of contributing processors. With this *gather–scatter* setup the modeled communication volume is realized in practice.

The *communication data structure* contains information on the sets of contributing processors for each component. This information has to be stored so that the gather and scatter operations can be executed efficiently in every iteration.

A straightforward way to build the communication data structures, is to compute and store for each individual component its set of contributing processors, and to designate one of them as an owner (e.g., at random or through a round-robin scheme). However, this will severely increase the memory use, since the size of the communication data structures for a realistic number of processors will be bigger than the projection data itself. This is because the metadata, that identifies what is being communicated, is associated with every individual component.

To remedy this problem, we again exploit the fact that line segments corresponding to neighbouring pixels on the detector often cross the same parts. In particular, we would like to find the regions of pixels of projection images that have the same set of contributing processors. This can be realized by looking at arrangements induced by the shadows of each part of the partitioning.

An *arrangement* is a subdivision of the plane into a collection of labeled regions, or *faces*. In our case, we are interested in subdivisions of the detector plane, and the labels (or *tags*) are the sets of contributing processors for the face.

We consider each projection separately. Every processor shadow defines an arrangement of the rectangle of the detector containing two faces: the shadow of the part, and its complement. The p arrangements can be merged efficiently, as described in section 2.3 of the textbook by de Berg et al. [Ber+08]. The resulting overlay arrangement has faces defined by the intersections of the faces in the original arrangements, and the tags can be combined arbitrarily. In our case, the faces in the original arrangements have a single contributing processor corresponding to a tag that is a list with one element. We start with an empty arrangement, and iteratively merge in the arrangements for each processor. When new faces are constructed during the MERGE subroutine, the lists of contributors of the original faces are concatenated. After merging together the p arrangements, the resulting overlay structure defines a number of faces, and each of these faces has an associated set of contributing processors as defined by its tag. We summarize this method in algorithm 6.

Our novel communication data structure is thus a subdivision of the detector into a set of faces, with an associated tag for each face listing the contributing processors for that region. For a visual example, see figure 4.4. We then proceed to rasterize these faces, leading for each face to a

Algorithm 6 Finding the overlay for the communication structure for a given projection p_k. Here, $[s]$ is a list with a single element s.

Subroutine: FINDFACES
Input: $\pi = \{V_s\}, p_k$
Output: OVERLAY

OVERLAY ← EMPTYARRANGEMENT
for $0 \leq s < p$ **do**
  $\text{CORNERS}_s \leftarrow \text{PROJECT}(p_k, \text{VERTICES}(V_s))$
  $\text{SHADOW}_s \leftarrow \text{CONVEXHULL}(\text{CORNERS}_s)$
  $\text{ARRANGEMENT}_s \leftarrow \text{FROMFACETAG}(\text{SHADOW}_s, [s])$
  $\text{MERGE}(\text{OVERLAY}, \text{ARRANGEMENT}_s, \text{CONCATENATE})$

collection of scanlines. A *scanline* is a consecutive set of pixels of a row on the detector. We use this collection of scanlines in the final algorithm for performing the communication during an SpMV operation. This novel approach not only drastically reduces the size of the communication data structures, but also allows to perform aggregate reads from GPU memory.

4.4 Numerical experiments

We consider four categories of acquisition geometries for our numerical experiments.

- CCB. *Circular cone-beam*. The source and detector move in a circular trajectory around the object. This is the typical geometry for laboratory CT machines. We distinguish between CCB_n where the cone has a narrow angle, and the source is relatively far away, and CCB_w with a wide angle, and the source is close to the volume.
- HCB. *Helical cone-beam*. The source and detector move in a helical trajectory around the object. This is similar to CCB, but in addition to the circular movement, the source and detector also move along the orthogonal direction. This is a common acquisition geometry in medical CT.
- LAM. *Laminography*. The source and detector both move along their own circular trajectory, but these trajectories are on opposite sides of

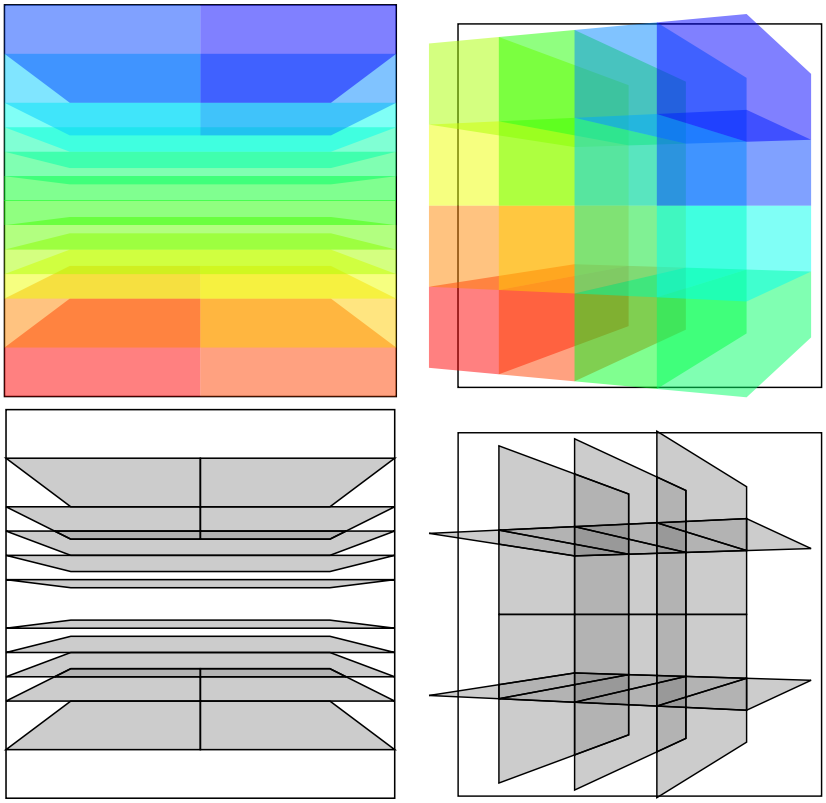


Figure 4.4: Example of the overlay structure for a single projection of the CCB_w (left), and LAM_w (right) geometries (see section 4.4). Note that the shadows of a part might partially fall outside of the detector. On the top row, the shadows of the coloured parts are given. On the bottom row, the overlay structure is shown. In the overlay, a darker gray indicates a larger set of contributing processors.

the volumes, typically perpendicular to one of the axes of the object volume. This geometry can be used to image flat objects. We distinguish between LAM_w with circular trajectories with a large radius, and LAM_n with a small radius.

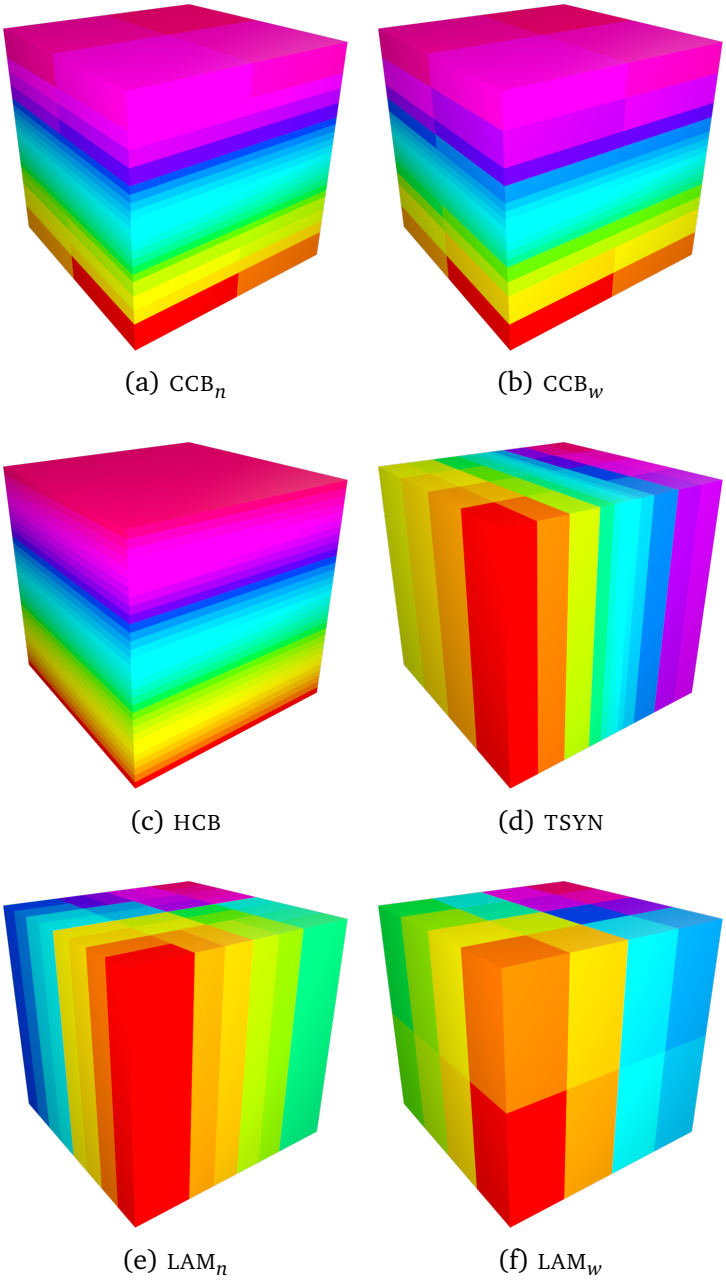
- TSYN. *Tomosynthesis*. A static detector is placed under the object, while the source moves along a limited-angle arc above the object. This geometry is used, e.g., for breast cancer screening and airport security.

Partitioning results.

Here, we compare two methods for load balancing that were discussed in section 4.2.2, MIDWAY where we split the volume into two parts of equal volume, and SAMPLING where we take a fixed number $N = 100\,000$ of sample points for which we evaluate the integrand in (4.1), and then perform a linear scan to find the optimal splitting point. We compare the results for these methods with the original GRCB partitioning method. In practice, these partitionings are of interest for multi-GPU clusters consisting of up to $p = 64$ GPUs. We therefore consider three processor counts, 16, 32, and 64. The partitioning statistics such as communication volume and load imbalance are evaluated on volumes consisting of 256^3 voxels, which is fine enough to obtain accurate statistics. We show some of the partitionings visually in figure 4.5.

In table 4.1, we show the communication volume Λ, load imbalance ϵ, number of messages μ (i.e., the number of processor pairs that perform the communication), and partitioning time T. Note that we do not optimize for the number of messages explicitly. First, we observe that there are no large discrepancies in the communication volume between the three different methods. For MIDWAY, the partitioning time is low (between 100 ms–600 ms), but the load imbalance can be up to 0.34 for the geometries considered. The number of messages is somewhat lower compared to the other partitioning algorithms, since the parts are automatically aligned because of the fixed split points, which is beneficial for the number of messages. The maximum number of messages is $\mu_{\max} = p(p-1)$, and we note that the number μ achieved is often a significant fraction of $\mu_{\max}$. This attests to the difficulty of avoiding communication in tomography, caused by rays crossing the object in many directions. We see that the SAMPLING method based on our continuous formulation of the load balance is able to achieve a reasonable load balance. Only in two cases it is slightly higher than the maximum load imbalance (0.05) that was imposed for GRCB. The runtime of the partitioning algorithm is up to 100× less than the runtime of GRCB, while the resulting partitionings have similar quality.

In table 4.2, we consider the communication volume for regularized reconstruction methods that solve (4.3). We do this by explicitly considering communication because of image gradient computations during the partitioning method, or ignoring this cost, as explained in section 4.2.3. The



(a) CCB$_n$ (b) CCB$_w$

(c) HCB (d) TSYN

(e) LAM$_n$ (f) LAM$_w$

Figure 4.5: Resulting partitionings for the circular cone-beam (CCB), helical cone-beam (HCB), tomosynthesis (TSYN), and laminography (LAM) acquisition geometries. The results shown are for $p = 32$ processors using the MIDWAY load balancing strategy.

		MIDWAY				SAMPLING				GRCB			
p	$\mathscr{G}$	Λ	ϵ	μ	T	Λ	ϵ	μ	T	Λ	ϵ	μ	T
16	CCB_n	0.72	0.00	44	0.15	0.72	0.00	40	6.72	0.72	0.00	44	242.54
	CCB_w	1.26	0.01	64	0.09	1.26	0.04	64	4.95	1.26	0.03	64	274.98
	HCB	1.14	0.28	84	0.16	1.18	0.04	96	4.76	1.18	0.05	96	203.48
	LAM_n	0.92	0.00	84	0.15	0.92	0.04	140	7.28	0.91	0.05	120	240.11
	LAM_w	1.61	0.00	180	0.14	1.61	0.03	180	7.04	1.61	0.05	180	296.20
	TSYN	0.72	0.10	76	0.15	0.73	0.03	76	2.22	0.72	0.05	76	210.32
32	CCB_n	1.28	0.00	180	0.32	1.28	0.00	172	4.75	1.28	0.04	180	273.62
	CCB_w	1.90	0.01	272	0.31	1.90	0.04	272	4.74	1.90	0.04	272	350.76
	HCB	1.91	0.33	328	0.33	1.96	0.04	350	4.92	1.98	0.05	368	296.13
	LAM_n	1.53	0.01	340	0.24	1.53	0.05	446	7.23	1.53	0.05	394	330.22
	LAM_w	2.61	0.17	552	0.19	2.66	0.03	552	7.23	2.65	0.05	552	347.40
	TSYN	1.19	0.10	284	0.19	1.19	0.05	272	2.43	1.19	0.05	284	253.28
64	CCB_n	1.92	0.04	640	0.38	1.92	0.04	648	5.43	1.92	0.05	640	324.07
	CCB_w	2.86	0.01	1040	0.37	2.85	0.04	1040	5.14	2.85	0.05	1040	449.93
	HCB	2.74	0.34	1052	0.40	2.80	0.05	1170	5.25	2.79	0.05	1150	400.16
	LAM_n	2.21	0.01	1268	0.37	2.20	0.06	1412	10.17	2.31	0.04	1346	480.77
	LAM_w	3.68	0.17	1978	0.60	3.74	0.06	2048	10.29	3.73	0.05	2020	536.73
	TSYN	1.79	0.12	936	0.40	1.80	0.05	928	4.09	1.80	0.05	948	246.44

Table 4.1: Partitioning statistics. We compare the communication volume Λ, given in multiples of 10^7, the load imbalance ϵ, the number of messages μ and the partitioning time T in seconds for various combinations of processor count p and acquisition geometry $\mathscr{G}$.

p	$\mathscr{G}$	$\frac{\Lambda^G}{\Lambda}$	$\frac{\Lambda^G_{\text{reg}}}{\Lambda_{\text{reg}}}$	$\frac{\Lambda^G_{\text{total}}}{\Lambda_{\text{total}}}$
16	CCB_n	1.00	0.88	0.97
	CCB_w	1.00	1.00	1.00
	HCB	1.00	1.00	1.00
	LAM_n	1.00	1.00	1.00
	LAM_w	1.00	1.00	1.00
	TSYN	1.00	1.00	1.00
32	CCB_n	1.00	0.92	0.99
	CCB_w	1.00	1.00	1.00
	HCB	1.00	0.94	0.99
	LAM_n	1.00	1.00	1.00
	LAM_w	1.00	1.00	1.00
	TSYN	1.00	0.92	0.99
64	CCB_n	1.00	1.00	1.00
	CCB_w	1.00	0.92	0.99
	HCB	1.00	1.00	1.00
	LAM_n	1.00	1.00	1.00
	LAM_w	1.00	1.00	1.00
	TSYN	1.00	0.95	0.99

Table 4.2: The relative performance when considering the gradient-based regularization in the communication volume. We compare the communication Λ due to SpMV, the communication Λ_{reg} due to an image gradient computation, as well as the total communication $\Lambda_{\text{total}} = \Lambda + \Lambda_{\text{reg}}$. The fractions given are the communications when explicitly taking into account the gradient communication during the partitioning (marked with a superscript G), divided by the communication when this cost is ignored, both for the SAMPLING method.

effect is limited because the communication due to image gradient computations is relatively small, as especially for larger processor counts the total communication volume is dominated by that of the SpMV step. However, it improves the overall communication in some cases, up to 3% for CCB_n, which is an acquisition geometry with relatively low communication volume Λ for the SpMVs.

Performance measurements.

We have implemented an extension to the open-source ASTRA tomography toolbox that allows tomographic reconstruction algorithms to run on distributed memory GPU clusters. This extension is called *Pleiades*, after the famous open star cluster. The ASTRA toolbox [Aar+16] has highly optimized GPU implementations of projection operators, which we use for the local forward projection and backprojection operations. Our extension uses Bulk [BBB18] to realize the communication between nodes. Bulk is a modern C++ library for bulk-synchronous parallel programs. It simplifies the implementation of communication logic significantly compared to, e.g., BSPlib or MPI.

Our extension is an improvement over a previously published extension to the ASTRA toolbox based on MPI [Pal+17], which we will call ASTRA-MPI. This previous extension uses slab partitionings, where the volume is split up into blocks of consecutive slices along one of the axes. This makes it suitable only for circular cone-beam geometries.

In contrast, our distributed-memory extension to the ASTRA tomography toolbox is flexible with respect to the acquisition geometry and the used data partitioning, which we achieved by implementing the techniques outlined in this chapter. We compare the performance of Pleiades to that of ASTRA-MPI for the only acquisition geometries in our set that ASTRA-MPI supports, which are CCB_n and CCB_w. In addition, we test the scalability of Pleiades for HCB. Our test consists of three Landweber iterations defined by the update:

$$\mathbf{x} \leftarrow \mathbf{x} + W^T(\mathbf{b} - W\mathbf{x}),$$

which follows the typical structure of an iterative method by alternating forward projection and backprojection operations. In all cases, we take a volume of 2048^3 voxels, and 1024 projections of 2048×2048 pixels.

The performance tests were run on a compute cluster of 8 nodes with a 40 Gbit Mellanox Infiniband connection. Each node has four NVIDIA GeForce GTX TITAN X GPUs, two Intel Xeon E5-2630 v3 CPUs running at 2.40GHz, and 128GB RAM. We use the MIDWAY strategy for Pleiades, partitioning over the 32 GPUs. Figure 4.6 shows the results of these measurements. For some acquisition geometries, the amount of available memory made it impossible to run with a low number of GPUs. Our initial implementation of Pleiades supports only $p = 2^q$ processors. We observe that

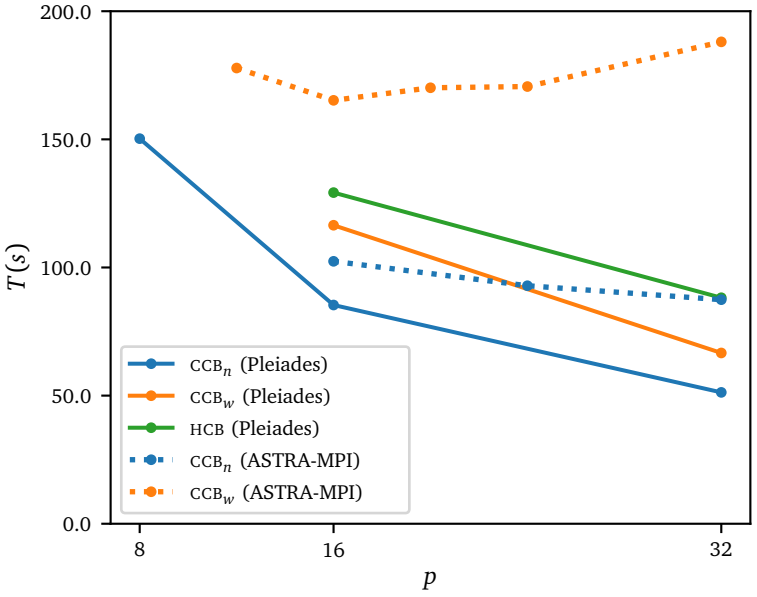


Figure 4.6: Scaling results of Pleiades versus ASTRA-MPI. Vertically, the runtime in seconds of three consecutive Landweber iterations is shown. Horizontally, we show the number of GPUs that were used.

Pleiades is significantly faster than ASTRA-MPI, and Pleiades continues to scale even when using all the available GPUs, unlike ASTRA-MPI which reached a communication bottleneck for CCB$_w$ at around 16 GPUs.

4.5 Conclusion

We presented a new partitioning method for tomographic reconstruction that can handle arbitrary acquisition geometries. Furthermore, we introduced an efficient data structure for the communication metadata that needs to be stored to use these partitionings in practice. We demonstrated that the method is able to produce partitionings of similar quality to those produced by the previously published GRCB method, but is much faster. Finally, we showed scalability results for using these partitionings in practice for a typical reconstruction task. For CCB$_w$ with 32 GPUs we achieved

a speedup of 2.8× compared to ASTRA-MPI.

Chapter 5

Real-time quasi-3D tomographic reconstruction

Tomography is an important non-destructive technique for studying the three-dimensional structure of samples in various scientific fields such as biology, material science, and medicine, as well as being broadly applied in industry. Increasingly, tomography is used to understand dynamic processes in detail, e.g., by imaging biological samples that vary with time [Moo+13], or by studying material properties in a changing environment [Pat+15; Gib+15].

The change from static to time-resolved tomography is accompanied by a steep increase in computational requirements for the tomographic reconstruction. Moreover, many experiments have controlled parameters that rely, e.g., on specific events happening in the sample, which can be hard to identify from projection images alone. This means not only that the reconstruction is computationally expensive, but also that the typical offline reconstruction does not fulfill current needs due to long computation times.

In addition to the need for real-time tomography, i.e., having access

---

This chapter is based on:

Real-time quasi-3D tomographic reconstruction. *JW Buurlage, H Kohr, WJ Palenstijn, KJ Batenburg*. Measurement Science and Technology 29 (6), 2018

to reconstructions while scanning, developments in acquisition hardware also contribute to the computational challenge. For instance, the number of pixels on detectors is growing, and the detectors are operating at increasing frame rates. Furthermore, real-time tomography scanners are being developed for, e.g., airport security setups [Tho+15; War+16], which are able to perform full scans in short time windows. This highlights the importance of efficient reconstruction techniques.

Current approaches to tackle the computational challenges in real-time tomographic reconstruction can be roughly subdivided into two groups. First, reconstruction algorithms that are computationally more efficient are being adopted. Two examples of this are the *gridrec* method [Dow+99; MS12] and methods based on the log-polar Radon transform [Nik+17]. Second, reconstruction algorithms can be run in parallel, either on distributed compute clusters or specialized hardware such as GPUs [PBS11; Pal+17; Xu+10]. However, while these approaches can lead to a dramatic reduction in reconstruction times, the computational demands for reconstructing the full 3D volume remain a bottleneck for truly real-time tomographic reconstruction. By realizing that while currently often full 3D reconstructions are made, the reconstructed volume is primarily viewed slice by slice, we observe that more computational work is done than necessary.

Instead, one can create a processing workflow where slices are only reconstructed on demand. In this way, the computational requirements can be reduced by orders of magnitude, and in many cases the required amount of data communication can also be significantly reduced. Filtered backprojection (FBP) type methods allow these slices to have an arbitrary orientation. From a user's point of view these slices can easily be shifted and rotated and effectively it is as though 3D data is available, while only a small number of slices are actually reconstructed at any time, as illustrated by Figure 5.1. With this shift in perspective, we make *quasi*-3D real-time tomographic reconstruction feasible, in the sense that the results are visually identical to an architecture where the full 3D volume is reconstructed and then viewed slice-by-slice, yet at a fraction of the computational cost.

In this chapter we present a new methodology for real-time reconstructions, together with a software stack implementing these ideas. In Section 5.1 we revisit the mathematical properties of FBP type methods, that enable us to reconstruct arbitrarily oriented slices without forming the full 3D volume. While these properties follow directly from the basic formu-

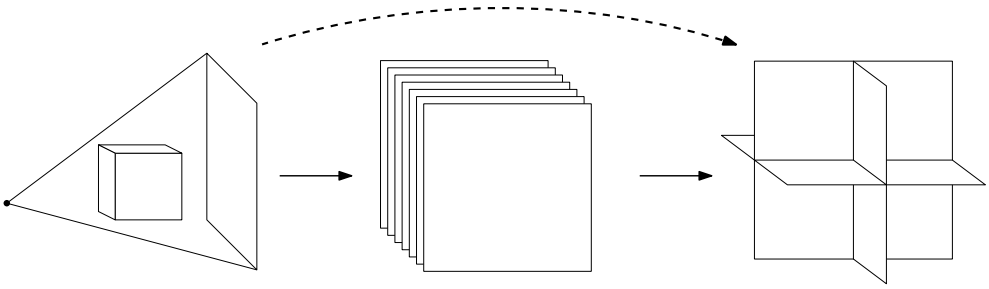


Figure 5.1: The solid arrows give a high-level overview of the data flow in a typical tomographic reconstruction setup. On the left, the projection data is acquired. In the middle, a reconstruction stack is created with an image for each slice along the rotation axis. From these slices, arbitrary slices of other orientations can be obtained through interpolation. In our new approach, represented with a dotted line, the generation of the reconstruction stack is skipped, and arbitrary slices are reconstructed directly from the projection data.

las, current approaches usually reconstruct the full 3D data at once. In Section 5.2, we present the interface and usage of the RECAST3D (REConstruction of Arbitrary Slices in Tomography) visualization software. It is a vital component of the proposed real-time reconstruction pipeline, as it allows the user to choose the slice(s) of interest in a dynamic way. In Section 5.3 we introduce the different components that are necessary to perform quasi-3D reconstructions. We highlight the unique distributed architecture of our novel reconstruction pipeline. Finally, in Section 5.4, we show that this new software greatly reduces reconstruction times, ultimately enabling almost instant slice reconstructions.

5.1 Reconstruction of arbitrary slices

Filtered backprojection (FBP) type methods for tomography are known to be very efficient in terms of numerical complexity and data usage. Whenever there are sufficiently many projections over the entire range of view angles, and the noise level is not too high, FBP typically performs very well also in terms of reconstruction quality. Here we understand as FBP any method that adheres to the “convolve, then backproject” workflow as shown in Fig-

ure 5.2. Examples of such methods are standard parallel beam FBP, the FDK algorithm for circular cone beam reconstruction [FDK84], and Katsevitch's algorithm for helical cone-beam reconstruction [Kat02] or general source trajectories [Kat03].

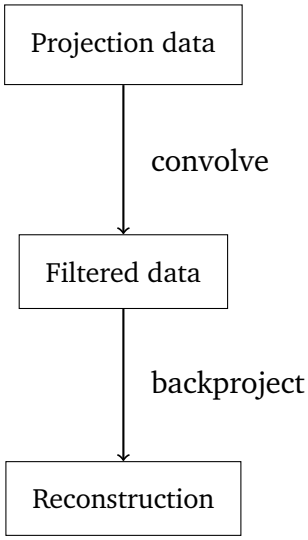


Figure 5.2: Workflow of filtered backprojection methods.

It is well-known that in 3D parallel beam geometry, horizontal slices can be reconstructed independently and from a single detector row. However, as we will demonstrate, FBP values in any subset of the reconstruction volume are mutually independent in any geometry. We start by recapitulating the well-known horizontal slice-by-slice reconstruction method in parallel beam geometry and then generalize to arbitrary slices and arbitrary geometries.

5.1.1 Parallel beam geometry

We consider the 3D parallel beam geometry with the z-axis as the only rotation axis (single-axis tilting). If f denotes a 3D volume, the corresponding projection data is given as the line integrals

$$g(\varphi, s, z) = \int_{-\infty}^{\infty} f\big(-t\sin\varphi + s\cos\varphi, t\cos\varphi + s\sin\varphi, z\big)\,\mathrm{d}t.$$

In an idealized setting, these values are available for $\varphi \in [0, \pi)$ and $(s,z) \in \mathbb{R}^2$. Filtered backprojection now consists of a one-dimensional filtering operation with a filter $k : \mathbb{R} \to \mathbb{R}$ in the s variable for each z, followed by backprojection:

$$g_{\text{filtered}}(\varphi,s,z) = \int_{-\infty}^{\infty} g(\varphi,s-u,z)\,k(u)\,\mathrm{d}u, \tag{5.1}$$

$$f_{\text{FBP}}(x,y,z) = \int_0^{\pi} g_{\text{filtered}}\big(\varphi, x\cos\varphi + y\sin\varphi, z\big)\,\mathrm{d}\varphi. \tag{5.2}$$

From (5.1) and (5.2) it is immediately clear that horizontal slices

$$f_{z_0}(x,y) = f(x,y,z_0),$$

with fixed $z = z_0$ can be reconstructed from a single data row $g_{z_0}(\varphi,s) = g(\varphi,s,z_0)$, i.e.,

$$f_{z_0,\,\text{FBP}}(x,y) = \int_0^{\pi} g_{z_0,\text{filtered}}\big(\varphi, x\cos\varphi + y\sin\varphi\big)\,\mathrm{d}\varphi.$$

In fact, if one is interested only in $f_{z_0}(x,y)$ for a single value z_0, then one has to perform also the filtering in (5.1) only for this fixed value of z_0. This reduces the whole task of reconstructing a horizontal slice to a two-dimensional problem.

Remark: It is important to notice that for all variants of single-slice reconstruction, the computed values in the slice are *identical* to the values in the full 3D reconstruction, restricted to the same slice.

The right-hand side of (5.2) refers only to the *current* reconstruction point (x,y,z), which implies that these points can be placed arbitrarily in 3D space. In particular, this mutual independence is not a special property of the parallel beam geometry but rather of the structure of the FBP algorithm itself. Hence it generalizes immediately to arbitrary slices and arbitrary geometries.

For instance, the remaining ortho-slices can be reconstructed as follows:

$$f_{x_0,\,\text{FBP}}(y,z) = \int_0^{\pi} g_{\text{filtered}}\big(\varphi, x_0\cos\varphi + y\sin\varphi, z\big)\,\mathrm{d}\varphi,$$

$$f_{y_0,\,\text{FBP}}(x,z) = \int_0^{\pi} g_{\text{filtered}}\big(\varphi, x\cos\varphi + y_0\sin\varphi, z\big)\,\mathrm{d}\varphi,$$

with the evident definitions

$$f_{x_0}(y,z) = f(x_0,y,z), \quad f_{y_0}(x,z) = f(x,y_0,z).$$

Again, the orthoslices contain the exact same values as a full volumetric reconstruction after restriction to these slices. Note, though, that both ortho-slices require the whole dataset since the z variable appears on both sides.

5.1.2 Cone beam geometry

We define the widely used *circular cone beam geometry* which is characterized by a point source moving on a circle of radius $r > 0$ in the x-y-plane and a flat detector on the opposite side of the same circle.

We parametrize the unit circle in the x-y-plane by

$$\boldsymbol{\theta}(\varphi) = (-\sin\varphi, \cos\varphi, 0), \quad \varphi \in [0, 2\pi).$$

Now we define the *source position* and the *detector piercing point* as two opposite points on a circle with radius $r > 0$ in the same plane:

$$\boldsymbol{a}(\varphi) = -r\boldsymbol{\theta}(\varphi), \quad \boldsymbol{p}(\varphi) = r\boldsymbol{\theta}(\varphi).$$

Finally we place a flat rectangular detector such that the ray from the source through the origin "pierces" the detector midpoint exactly at the piercing point $\boldsymbol{p}(\varphi)$, and orient the detector perpendicular to the piercing ray:

$$D(\varphi) = \left\{ \boldsymbol{p}(\varphi) + u\,\boldsymbol{\theta}^{\perp}(\varphi) + z\,\boldsymbol{e}_z \;\middle|\; -w/2 \le u \le w/2,\ -h/2 \le z \le h/2 \right\}.$$

Here, w and h stand for the width and the height of the detector, respectively, $\boldsymbol{\theta}^{\perp}(\varphi) = (\cos\varphi, \sin\varphi, 0) = -\boldsymbol{\theta}(\varphi + \pi/2)$ the unit vector tangent to the circle at angle φ, and $\boldsymbol{e}_z = (0,0,1)$. See Figure 5.3 for an illustration of the geometry.

This definition is straightforward to extend to arbitrary rotation axes and different radii for source and detector circles.

With these geometric conventions we define the projection data in circular cone beam geometry as

$$g(\varphi, \boldsymbol{y}) = \int_0^\infty f\big(\boldsymbol{a}(\varphi) + t\big(\boldsymbol{y} - \boldsymbol{a}(\varphi)\big)\big)\,\mathrm{d}t, \quad \varphi \in [0, 2\pi),\ \boldsymbol{y} \in D(\varphi). \tag{5.3}$$

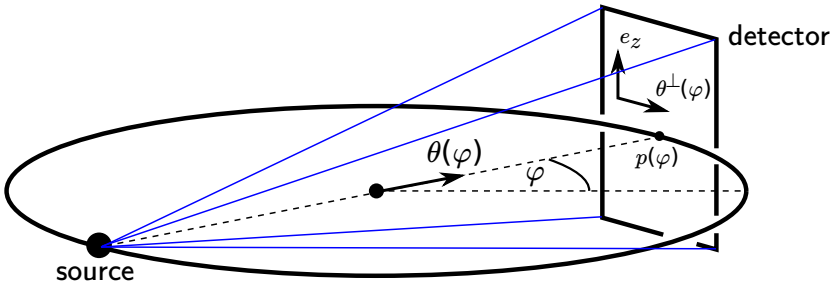


Figure 5.3: Sketch of a circular cone beam acquisition geometry as used by the backprojection (5.4).

It can be shown (see, e.g., [NW01]) that the backprojection for this geometry in a point $\boldsymbol{x} = (x, y, z)$ is

$$\mathrm{BP}[g](\boldsymbol{x}) = \int_0^{2\pi} \frac{1}{2r\,t(\boldsymbol{x},\varphi)^2}\, g\left(\varphi, \frac{\boldsymbol{x}\cdot\boldsymbol{\theta}^\perp(\varphi)}{t(\boldsymbol{x},\varphi)}, \frac{\boldsymbol{x}\cdot\boldsymbol{e}_z}{t(\boldsymbol{x},\varphi)}\right) \mathrm{d}\varphi, \tag{5.4}$$

where "$\cdot$" is the dot product in 3 dimensions and

$$t(\boldsymbol{x},\varphi) = \frac{\big(\boldsymbol{a}(\varphi) - \boldsymbol{x}\big)\cdot\boldsymbol{a}(\varphi)}{2r^2}$$

is the relative position of a reconstruction point $\boldsymbol{x} \in \mathbb{R}^3$ along the ray from the source point $\boldsymbol{a}(\varphi)$ to the detector through $\boldsymbol{x}$. Although the backprojection (5.4) is more involved to evaluate numerically, it still computes the value at a given volume point $\boldsymbol{x}$ independently from any other such point.

A very popular reconstruction method in circular cone beam geometry is the FDK algorithm [FDK84]. It consists of applying a one-dimensional filter k_{FDK} along the column coordinate u to preweighted measurements $\tilde{g}$, followed by the backprojection given in (5.4):

$$\begin{aligned}
\tilde{g}(\varphi, \boldsymbol{y}) &= \tfrac{\|\boldsymbol{p}(\varphi)-\boldsymbol{a}(\varphi)\|}{\|\boldsymbol{y}-\boldsymbol{a}(\varphi)\|}\, g(\varphi, \boldsymbol{y}),\\
g_{\mathrm{filtered}}(\varphi, u, z) &= \textstyle\int_{\mathbb{R}} \tilde{g}(\varphi, u-v, z)\, k_{\mathrm{FDK}}(v)\,\mathrm{d}v,\\
f_{\mathrm{FDK}}(\boldsymbol{x}) &= \mathrm{BP}[g_{\mathrm{filtered}}](\boldsymbol{x}).
\end{aligned}$$

Typically, k_{FDK} is chosen to be the *ramp filter*. To reconstruct an arbitrary slice $S = \boldsymbol{r} + \boldsymbol{n}^\perp$, $\boldsymbol{r}, \boldsymbol{n} \in \mathbb{R}^3$, $\boldsymbol{n} \neq (0,0,0)$, we can simply evaluate this

formula for all $\boldsymbol{x} \in S$. Just as for parallel beam reconstruction, the values computed in the slice are the same as if a full 3D FDK reconstruction was restricted to the same slice. In fact, the single-slice reconstruction avoids the interpolation step that would otherwise be incurred when restricting a full 3D reconstruction to a slice.

The FDK algorithm approximates the exact solution only in the central horizontal slice $z = 0$, while for other points in the volume, the data provided by circular cone beam acquisition is insufficient, leading to cone-beam artifacts. In [JKM11] the performance of FDK for experimental data is discussed. Certain extensions and modifications such as those that choose a specific filter, see, e.g., [Hah+13], also fit into our proposed framework.

To acquire complete data, one can additionally move both $\boldsymbol{a}$ and $\boldsymbol{p}$ with constant velocity $l/(2\pi)$ along the rotation axis $\boldsymbol{e}_z$ relative to the object, resulting in a helix instead of a circle:

$$\boldsymbol{a}(\varphi) = -r\boldsymbol{\theta}(\varphi) + \frac{l\varphi}{2\pi}\boldsymbol{e}_z, \quad \boldsymbol{p}(\varphi) = r\boldsymbol{\theta}(\varphi) + \frac{l\varphi}{2\pi}\boldsymbol{e}_z.$$

For this helical geometry, the reconstruction formula of Katsevich [Kat02] provides exact inversion. It is also of filtered backprojection type, even though both filtering and backprojection have more complex expressions. The formula induces a family of FBP methods by replacing the filter for exact inversion with a regularizing filter. In fact, for any piecewise smooth source trajectory satisfying certain geometric conditions, an exact FBP type reconstruction formula can be given [Kat03].

In conclusion, a method for the fast computation of a single-slice FBP reconstruction is useful for applications with either parallel beam or cone beam acquisition.

5.2 Software

Using the mathematical properties of FBP methods discussed in the previous section, we can introduce an optimized workflow for real-time visualization of tomographic reconstructions. In this section we present *RECAST3D*, visualization software that controls an on-demand reconstruction pipeline. In particular, it can be used for on-the-fly reconstruction of arbitrarily oriented slices. Our novel approach is to only compute a limited

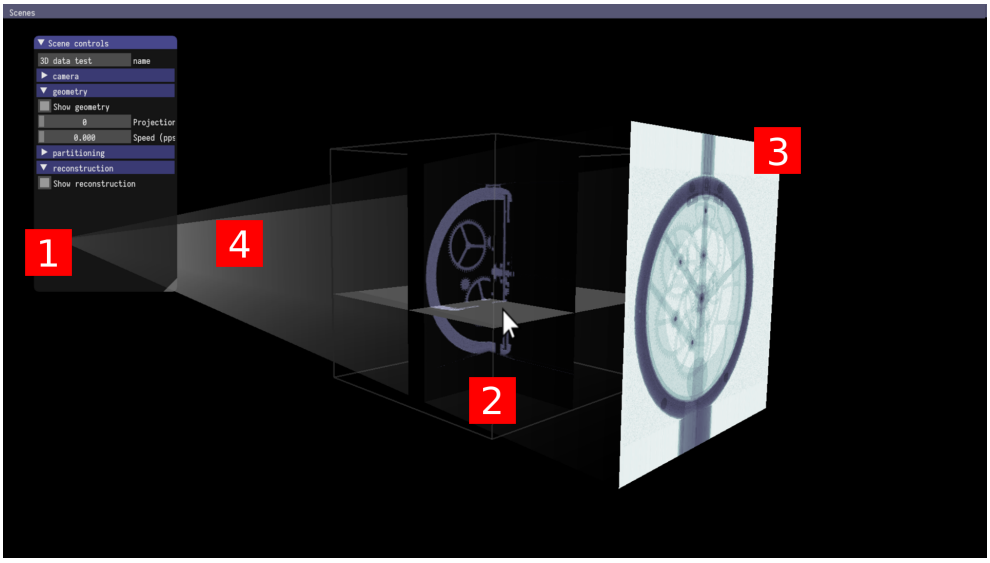


Figure 5.4: Screenshot of RECAST3D. Some simple analysis tools are provided in a GUI (1). In this example setup, three orthogonal slices are being shown in the middle (2) with the mouse currently hovering over one of them. A user can translate and rotate the planes by dragging them with the mouse. When the mouse button is released, the visualizer requests a reconstruction of the new slice. During the change of slice orientation and position, a low-resolution preview is shown. The interface is highly extensible. As an example we show the projection images (3) and the beam direction (4) in the same scene as the reconstruction, providing the user with additional information about the experimental setup.

number of slices, for example a set of three orthogonal slices, lowering the computational costs of the reconstruction tremendously. The slices that are being reconstructed can be changed with an intuitive interface. An exemplary screenshot of the visualization software is shown in Figure 5.4.

From a user's perspective, a typical workflow with RECAST3D is as follows. The tool is started on a workstation and connects to a reconstruction server that receives the relevant projection images. For small enough problems, this server can be the workstation itself. The software asks for specific slice reconstructions from the reconstruction server, initially presenting three orthogonal slices to the user. Assuming RECAST3D is used in a real-time setting, these are being reconstructed on-the-fly. The user can

hover the mouse over the slices and rotate and translate them in an intuitive manner. As new projection images arrive, the slices can be updated continuously.

We envision a modular system which we can extend gradually over time, as common needs and requirements become more clear. In the initial version of RECAST3D, next to the high resolution slices a low-resolution 3D preview is available when changing the orientation of a slice which allows the user to identify slices that are of particular interest. In addition, we show the projection images and visualize the acquisition geometry in the same 3D scene as the reconstruction. This presents the user with even more insight on the data that is coming in in real-time. It is possible to, e.g., change the color scheme that is used, or to rescale the data.

5.3 Implementation

The implementation of RECAST3D required a complete redesign of the typical tomographic reconstruction pipeline. In our discussion here we distinguish between three different stages of the reconstruction pipeline: acquisition, reconstruction and visualization. Note that in an actual experimental setting, we will need additional operations such as flatfielding and ring artefact correction. In the realization of our new quasi-3D reconstruction pipeline, all these stages work together with the common goal of giving the user a real-time quasi-3D reconstruction. To ensure the flexibility and scalability of our pipeline the system is completely distributed, in the sense that communication between the software components for the different stages happens through well-defined packets using a message passing protocol. The software stack consists of three main components:

1. *Reconstruction software* that is capable of performing the reconstruction of an arbitrarily oriented slice.

2. Definitions of the various *packets* supported by our communication protocol, together with a software library for constructing, sending, receiving, and parsing these packets.

3. The *software for real-time visualization*, RECAST3D, which is also the *control center* for the distributed software stack.

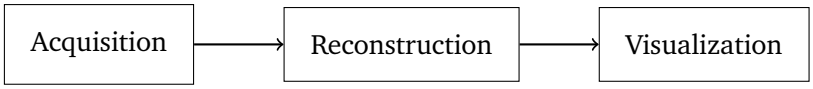


Figure 5.5: A simplified, but typical tomography pipeline, where the common pre- and post-processing steps are ignored. We emphasize here its linearity, i.e., data proceeds in its entirety from one stage to the next. Furthermore, in most cases these phases happen completely in a sequential manner.

Together, these components form an implementation of an extended reconstruction pipeline. Typically, the data in a tomography setup flows as in the linear pipeline shown in Figure 5.5. The software stack we introduce puts all the components in direct and real-time contact, enabling finer control over the dataflow, as shown in Figure 5.6. This has a number of advantages. We list some of them, in no particular order:

- Only subsets of the data have to be sent (or are requested) between the different stages.
- The computational requirements are significantly reduced, since only the slices that are shown are reconstructed.
- Since the entire system is integrated, the rich feedback allows the user to perform experiments faster and more efficiently

Distributed architecture

As mentioned in the previous section, our distributed architecture is based on a message passing protocol. Here, we describe in detail the different concepts and parts used in the distributed pipeline.

An experiment, or reconstruction, is captured in the system as a *scene*. These scenes consist of a number of data objects, such as reconstructed slices, projection data, and information on the acquisition geometry.

The central concept in the distributed pipeline is that of a *packet*. There are various packets that are used for communication, some examples are given in Listing 5.1. Every packet contains metadata used to identify an object in question (e.g., an identifier for a scene, and a slice), and perhaps

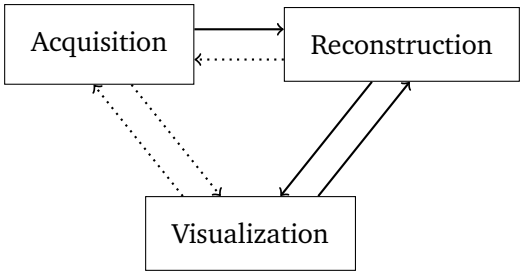


Figure 5.6: An extended complete pipeline, cf. Figure 5.5. All different stages are in direct contact, and no longer happen sequentially but in parallel. The implementations of the stages of the tomographic pipeline now communicate and coordinate with each other, reducing the dataflow and computational requirements. Although our distributed pipeline supports all communication paths, only the solid arrows are currently used.

some payload (i.e., a projection image, or a reconstructed slice) together with fields describing the payload such as the number of pixels or the position of the detector and source.

The packets that are described are independent of the specific technology used for sending them. In our reference implementation, ZeroMQ streams are used for communication. The core of the software stack is written in the C++ programming language.

Because the architecture is completely distributed, all components can be used independently and they are easily extensible. This modular approach allows users of our software to easily use or replace parts of the pipeline to suit their own purposes. Bindings to the Python programming language are provided, giving an accessible customization point. See also Listing 5.2 for an example of a custom script in our framework, which is able to completely replace the reconstruction component.

More generally, an important internal guideline for the development of this new pipeline is that it should be able to leverage existing and future software that is developed for image reconstruction. The library and specification take care of the necessary communication and coordination. The extended pipeline is implemented on a high level, rather than modifying existing software. Instead, existing software is used wherever possible. This gives our new system the great advantage of supporting custom software, from acquisition to reconstruction to visualization. Our current

```
struct GeometrySpecification {
    int32_t scene_id;
    bool parallel;
    int32_t projections;
    std::array<float, 3> volume_min_point;
    std::array<float, 3> volume_max_point;
};

struct SliceData {
    int32_t scene_id;
    int32_t slice_id;
    std::array<int32_t, 2> slice_size;
    std::vector<uint32_t> data;
};
```

Listing 5.1: Example packets, represented as a record data structure in the C++ programming language. The first packet defines some global information on the acquisition geometry: the number of projections, whether it describes a parallel or cone beam setup, together with the object volume which describes a bounding box for the sample being imaged. The second packet defines the data for a specific slice, with fields for the number of pixels together with the raw reconstructed data.

reconstruction server is built on top of ODL (the Operator Discretization Library [AKÖ17]) for describing the required geometric transformations at a high level, and the ASTRA Toolbox [Aar+16] for GPU-accelerated back-projection, customized for single slice processing.

Our software is available in open-source repositories, and can be found at `https://github.com/cicwi/`.

5.4 Results

In this section we compare the computational performance (i.e., the speed of reconstruction) of quasi-3D reconstructions to full 3D reconstructions. For the results presented here, the reconstructions are performed on a single node. This node has two Intel Xeon E5-2623v3 processors, 128 GB

```
import tomop

def reconstruction_callback(slice_geometry):
    data = custom_slice_data_function(slice_geometry)
    return data

server = tomop.server("Scene title", "tcp://localhost:5555")
server.set_callback(reconstruction_callback)
server.serve()
```

Listing 5.2: Example script for custom on-demand slice reconstruction. When the user rotates, translates, or creates a slice in the visualization interface, the system will request the new data for this slice using the user-supplied callback function. In the first line, the tomopackets library is imported. Next, a callback function is defined that takes an orientation, reconstructs the corresponding slice, and returns that reconstructed data. Below that, it is shown how to setup and connect a server.

RAM, and two dual-GPU NVIDIA GTX TITAN Z cards for a total of 4 GPUs with 6GB RAM each. The projection data has been prerecorded and pre-filtered, and is directly available to the reconstruction software. During a scan, the filtering can be done at the detector while taking images, without impacting the reconstruction time.

We use simulated data in our experiments. The test geometry is a circular cone beam geometry with rotation axis $\mathbf{z}$. The object has size $N \times N \times M$. The virtual detector is of size $N \times M$ and is positioned at the origin. The source is at distance $10 \times N$ from the center of the object. We take a total of N projections. Here, N and M are varied throughout our experiments.

The number of detector pixels that are required for the reconstruction of a single slice depends on the orientation of the slice (see also Section 5.1). We consider three slices: 1. an *axial slice* is a slice orthogonal to the rotation axis, 2. a *vertical slice* is parallel to the rotation axis, 3. a slice in between these extremes is a *tilted slice*.

We compare the timings of a full 3D reconstruction, with the timings of slice-based reconstructions for various orientations in Table 5.1. Some examples of the reconstructed slices are shown in Figure 5.7. Note that,

voxels	GPUs	full 3D	axial	vertical	tilted
$256 \times 256 \times 256$	1×	0.84 s	26.5 ms	22.6 ms	23.8 ms
	4×	0.31 s	35.9 ms	26.6 ms	22.9 ms
$512 \times 512 \times 512$	1×	1.07 s	33.4 ms	22.6 ms	31.8 ms
	4×	0.60 s	40.4 ms	27.2 ms	23.5 ms
$1024 \times 1024 \times 1024$	1×	17.3 s	61.6 ms	64.8 ms	63.1 ms
	4×	6.69 s	38.5 ms	39.1 ms	37.2 ms
$2048 \times 2048 \times 1024$	1×	274 s	286 ms	5.22 s	5.48 s
	4×	65.0 s	100 ms	106 ms	105 ms

Table 5.1: Reconstruction times for full 3D data, compared to reconstruction times for 2D slices of various orientations. See the text for a description of the hardware and test geometry. Here, the axial and vertical slices are taken at the center of the volume. The tilted slice is an axial slice, rotated 45° around the $\mathbf{x}$ axis. We consider a varying number of reconstructed voxels, corresponding to the $N \times N \times M$ volumes in the text. The performance when using a single GPU or multiple GPUs is also compared. For the relatively low numbers presented here, the standard deviation can be as high as 20% of the measurement, while for higher resolutions the numbers get relatively more stable with standard deviations of about 10% of the measurement.

as explained in Section 5.1, the single slice reconstructions are identical to reconstructions that would be obtained from a full 3D reconstruction. In particular, there is no loss of accuracy. The results show that individual slices can be computed quickly, even at high resolutions. The distributed system induces some overhead, which is included in the numbers presented. These can be a significant part of the total reconstruction times, particularly at lower resolutions. Using multiple GPUs can significantly decrease the reconstruction times, especially at high resolutions. For the highest resolution considered, the required data for reconstructing non-axial slices no longer fits on a single GPU which means that using multiple GPUs is a necessity for obtaining low reconstruction times.

When reconstructing vertical slices, already the complete data has to be filtered. In addition, the majority of the data is required for a backprojection. If all three orthoslices are required, then the complete data set is needed for the backprojection. However, the computational cost of the reconstruction always remains low. Because we visualize only individual

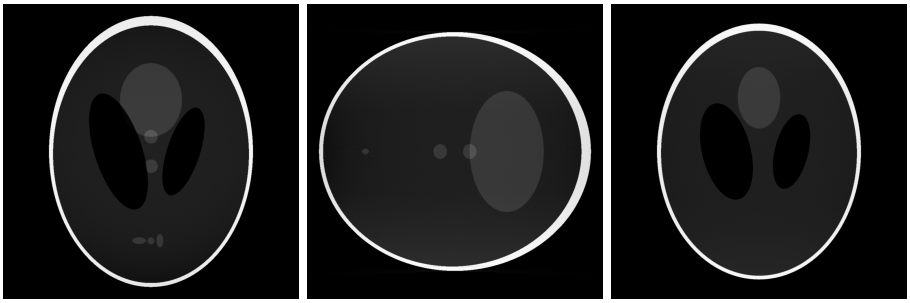


Figure 5.7: Reconstructed slices for a volume of $1024 \times 1024 \times 1024$ voxels. Here we used a modified 3D Shepp-Logan phantom. The left, middle and right reconstructed slices correspond to the axial, vertical and tilted slices as defined in Table 5.1.

slices, the amount of data required for visualization is always limited.

In our experiments we considered a circular cone beam geometry because in general it is a harder geometry to reconstruct than a parallel geometry. However, for quasi-3D reconstructions many properties that usually make reconstructing parallel geometries much simpler are lost, because slices of arbitrary orientation have to be reconstructed. In our experiments, we have observed similar performance for parallel geometries as for cone beam geometries.

5.5 Use cases

The ability to observe the internal state of the object in quasi-3D through the RECAST3D software is mainly valuable if real-time actions can be taken as a result of the observations, which would not be possible if one has to wait for a full 3D volume to be reconstructed. The RECAST3D software has several use cases, all related to various dynamic aspects of the image acquisition:

- **Dynamic processes within the object** of interest itself can be followed in real-time in a quasi-3D setting. For example, a bubble that moves through a liquid can be tracked by using three slices positioned in the center of the bubble and adjusting the slices to the observed direction.

- **Dynamic external parameters** related to the object state (temperature control, pressure control) can be adjusted to the observed state of the object. For instance, using a temperature controlled stage, the temperature of the object can be lowered until certain phase transitions occur inside the object (observed in the slices), after which the object is scanned at constant temperature.

- **Dynamic acquisition parameters** (source and detector positioning, rotation of the object) can be adjusted to the observed features of the object. For instance, the scanning geometry can be adjusted for the presence of metal (leading to artefacts) that has been observed at certain locations in the object and the object can be positioned closer to the source, zooming into a region-of-interest.

Moreover, the ability to quickly visualize several slices through the interior of the object while the object is in the scanner provides immediate feedback about the quality of the data, showing for example if the scan is good enough to resolve features of interest that are oriented in a particular direction chosen by the user.

5.6 Experiments

In this section we give two concrete examples of applications for the RECAST3D methodology.

The two datasets are acquired using the custom built and highly flexible FleX-Ray CT scanner, developed by XRE NV and located at CWI. The apparatus consists of a cone-beam microfocus X-ray point source that projects polychromatic X-rays onto a 1943×1535 pixels, 14-bit, flat detector panel. The acquired data is binned on the fly by 2-by-2 pixel windows, i.e., each raw projection is of size 972×768. The data is collected over 360 degrees in circular and continuous motion with 1200 projections distributed evenly over the full circle. For dataset A, the exposure time was 160 ms, the X-ray tube settings were 50kV, 50W, and we consider a limited detector window of size 1943×1135. For dataset B, exposure time was 100 ms, the X-ray tube settings were 40kV, 20W. The data is openly available online [CBB18].

As a first application, we give an example of a dynamic imaging situation where slice-based reconstruction can be sufficient. Consider a biomedical application where a needle is inserted into a subject or sample along a straight line, until some target is reached. First, the needle has to be located which can always be done by looking at, e.g., the standard three ortho-slices. After this, a slice containing the needle can be reconstructed dynamically. If necessary, this slice can be adjusted if the needle moves. To create a simplified test case for this use case, a needle-shaped structure was made out of Play-Doh and inserted in a box filled with poppy seeds (dataset A). As illustrated in Figure 5.8, a single projection is not sufficient to locate the needle, although the needle is visible. However, using the quasi-3D reconstruction a slice containing the needle can easily be identified.

As a second application, we consider an adaptive experiment where some finer structure is first located, after which a more detailed scan of this structure is made. An example would be to image growth rings in wood structures. This can be used, e.g., for non-destructive dendrochronology in archeological samples [Bil+12]. In the overview scan, the plane in which the growth rings lie can be found using our proposed methodology. After identifying this region, a high-resolution scan of this region can be made. As a test case we consider a piece of wood shaped as an egg (dataset B). In Figure 5.8, we show a single projection of the wooden egg, a quasi-3D visualization, and a slice containing the growth rings. Observe that in general it is hard to identify the growth-ring orientation from projection images alone.

5.7 Outlook and conclusions

In this chapter, we have introduced a new methodology for real-time quasi-3D tomographic reconstruction, and software implementing these ideas called RECAST3D. We show that reconstructing a limited number of arbitrarily oriented slices can be done at a fraction of the computational cost of a full 3D reconstruction, yet yielding similar information and insights for certain use cases.

In this work we focused on FBP and related reconstruction methods. In comparison, algebraic reconstruction methods lack the important proper-

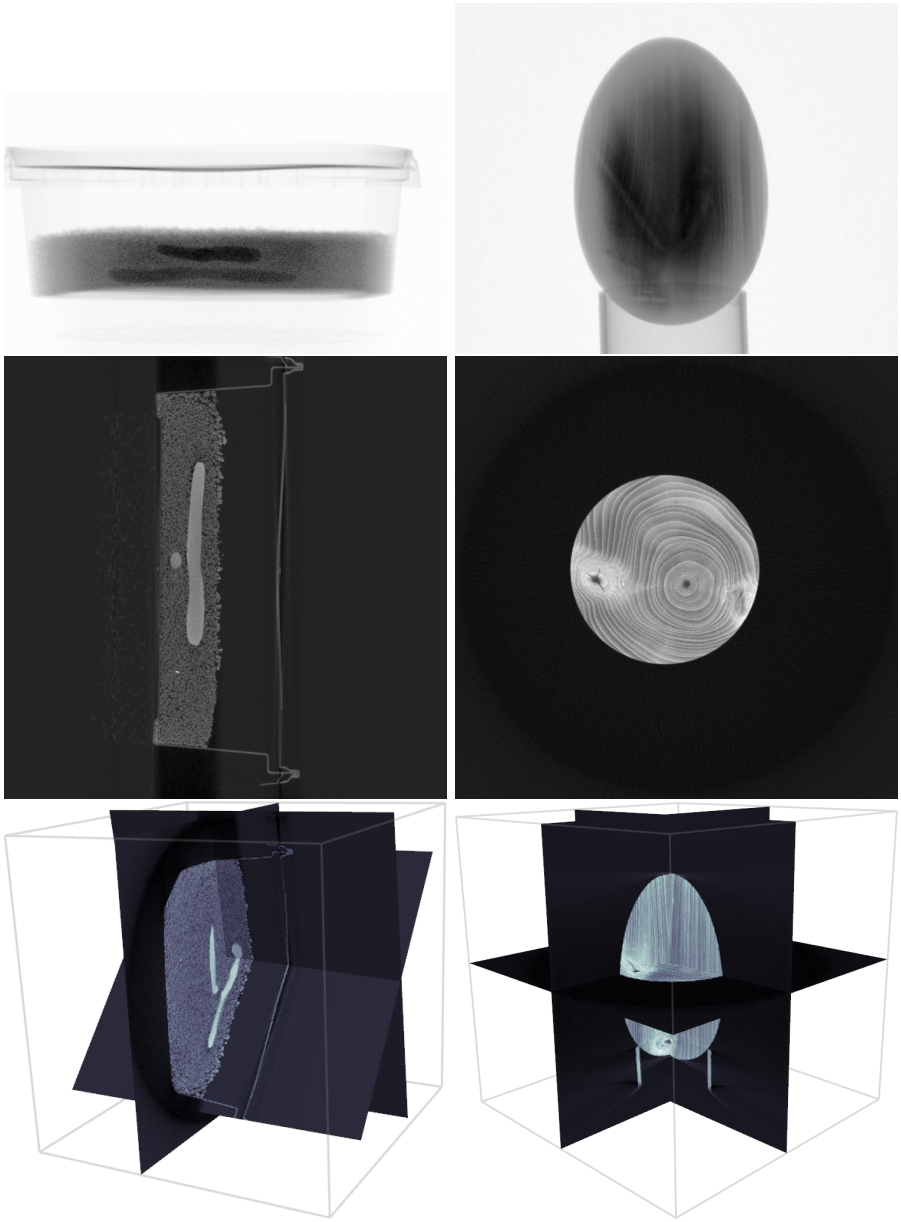


Figure 5.8: We show projections (top row), reconstructed slices (middle row) and quasi-3D reconstructions (bottom row). The contrast of the projections has been tuned by hand. On the left, dataset A is shown. On the right, dataset B is shown.

ties that we exploit. However, hybrid methods are conceivable which are tightly related to techniques for region-of-interest tomography. We expect that these more advanced reconstruction techniques can also fit into the framework presented here.

In addition to time-resolved experiments becoming more common, an interesting challenge will be to develop adaptive techniques. With these techniques, the scanning process itself can be steered based on the real-time reconstructions. Our distributed pipeline was developed specifically with this use-case in mind. Indeed, the cross-links between the different stages give rise to many interesting new possibilities. For example, the reconstruction cluster is able to control the scanner. This allows for algorithmically controlled experiments, that are driven dynamically by the reconstructions.

Chapter 6

Application of quasi-3D reconstruction to synchrotron tomography

Synchrotron tomography beamlines are powerful tools for obtaining high-resolution interior visualisations of a wide variety of opaque specimens with applications in life sciences, energy research, new materials, and many other fields. Thanks to advances in CMOS detector technology during the last decade and to the high photon flux available at state-of-the-art tomographic microscopy endstations, it is now possible to acquire the raw data required for computing a full 3D snapshot in well under one second at micron resolution, promoting the use of tomographic microscopy for time-resolved 3D imaging of interior dynamics [Mai+16; Gar+18; San+14]. For example, the GigaFRoST detector [Mok+17] in use at the fast tomography endstation of the TOMCAT beamline at the Swiss Light Source (PSI) can acquire up to 1255 full frame projection images of size 2016×2016 pixels, each second, and directly stream them to a data

This chapter is based on:

> Real-time reconstruction and visualisation towards dynamic feedback control during time-resolved tomography experiments at TOMCAT. *JW Buurlage, F Marone, DM Pelt, WJ Palenstijn, M Stampanoni, KJ Batenburg, CM Schlepütz*. Scientific Reports 9 (1), 1-11, 2019

backend that is capable of receiving and storing this 7.7 GB per second in a ring buffer. Efficient handling of these large data rates associated with time resolved tomographic experiments is a major challenge: large bandwidths for data transfer and data storage are required as well as sufficient computational resources for performing tomographic reconstruction and subsequent analysis. Even with modern efficient software packages and high-performance computational resources, the rate at which the data can be processed and analysed is often several orders of magnitude slower than these high rates of data acquisition. At most beamlines, typically the tomographic reconstruction of a high resolution volume takes at least a few minutes, with differences related to the used algorithm and available computational resources (e.g., [Gür+14; Atw+15; Mar+17; Pan+18]).

Direct visual feedback during a time-resolved experiment is of key importance for streamlining the efficiency of the physical imaging setup and the computational pipeline, which jointly determine the overall utilisation of the synchrotron beamline. At TOMCAT the beamline operator currently makes use of two types of direct visual feedback: (i) by observing the raw projection images it is possible to locate regions of interest in the sample, as long as these regions can be clearly identified in the projections, which is not always the case; (ii) by reconstructing a single axial slice on-the-fly and observing it during the experiment it is possible to get an initial grasp of the internal structure of the sample [Mar+17]. This limited form of real-time feedback during the experiment does not provide detailed insights in the 3D structure of the sample, particularly important for strongly anisotropic objects (e.g., fibres), where virtual tomographic slices with different orientations can look very different and provide valuable complementary information.

The lack of real-time 3D feedback represents a major obstacle to the efficiency of in particular dynamic imaging experiments. Rapid access to tomographic volumes could increase the success chances of the measurement campaign as it permits fast reaction towards the optimisation of the beamline parameters and data collection protocols to guarantee sufficient image quality to subsequently extract the relevant physical information. Acquisition problems that result in imaging artefacts, such as detector misalignment, could be resolved on-the-fly, thereby making much more effective use of expensive and scarcely available synchrotron beamtime. *In situ* experiments often require event-driven imaging, where the timing of the

operations performed on the sample (e.g., heating, wetting) and the timing of the image acquisition are tightly connected. Examples include stress loading of construction materials, water uptake of textiles, and migration processes inside batteries. By observing the interior dynamics in real-time during the experiment, the control parameters could be adjusted on-the-fly in response to the observed phenomena. Real-time feedback on the 3D structure of the sample would provide the ability to match the number of acquired tomographic volumes to the observed dynamics leading to a potentially substantial reduction of the total amount of produced data, not irrelevant during time-resolved experiments with kHz frame rate detectors, and to a maximisation of the information content in the stored datasets.

Because of the importance of direct 3D feedback during the experiment, previous research has focused on reducing the required computation time for obtaining a 3D snapshot of the scanned object, often through computational advances. One approach is to use supercomputing facilities to massively parallelise the various computations [Bic+15; Bic+17], significantly reducing the required computation time. For example, by using 32K supercomputing nodes, it is possible to compute full iterative 3D reconstructions in minutes [Bic+15]. However, supercomputing facilities typically have to be shared with other users, and computing time may not be available at the time it is needed during the experiment. A different approach is to use smaller clusters of GPU-equipped machines in combination with advanced software packages that can efficiently stream data to and from the GPUs [Vog+12]. As an example, with this approach it is possible to compute 3D snapshots of moderately sized problems in several seconds using six GPUs [Vog+12]. Despite these advances, real-time (i.e., sub-second) reconstruction and 3D visualisation during time-resolved tomography experiments is still out of reach.

In this chapter we present a data processing pipeline for real-time reconstruction and visualisation during the imaging experiment. Our main contribution is that we combine recent improvements in ultra-fast detector technology, networking, and tomographic reconstruction. This is a complex engineering effort, which requires combining expertise from multiple disciplines. Although several groups have shown the potential of real-time reconstruction at synchrotron light sources, we demonstrate for the first time a fully implemented pipeline for real-time reconstruction and visualization of time-resolved tomographic experiments. Instead of comput-

ing an entire 3D snapshot of the scanned object, our approach computes multiple arbitrarily oriented slices. The pipeline is based on combining the GigaFRoST detector system[Mok+17], which provides direct access to newly acquired projections, with the recently published RECAST3D software [Buu+18], which enables real-time visualisation of arbitrarily oriented slices by directly reconstructing the slices from the measured projections. The image reconstruction part of our pipeline runs on a single GPU-equipped workstation, thereby providing an imaging solution that can be implemented at the beamline in a straightforward manner without need for on-demand access to compute and network resources at a supercomputing facility. By setting up three orthogonal slices across the three main axes of the imaging system, a quasi-3D visualisation of the interior structure of the sample is obtained. During the experiment, the visualisations are automatically updated in real-time, ensuring that the most recent state of the scanned object is always shown. Since the visualised slices can be re-positioned and tilted in arbitrary directions at any time, the visualisation can be dynamically aligned with features of interest of the scanned object providing key information to the scientists in real time, unlocking the possibility to take further action towards the optimisation and control of the imaging and experimental parameters.

6.1 Method

To achieve real-time visualisation of tomographic experiments, our pipeline includes two main parts: a detector component that provides direct access to acquired projections in real time, and a software component that can process the acquired data and visualise results in real time. In the realisation we present here, the detector component is implemented using the GigaFRoST detection and readout system [Mok+17], while the software component consists of the RECAST3D real-time reconstruction and visualisation software and streaming architecture [Buu+18]. We will first discuss in more detail the elements of both components relevant to the presented pipeline, and then explain how the two components were integrated.

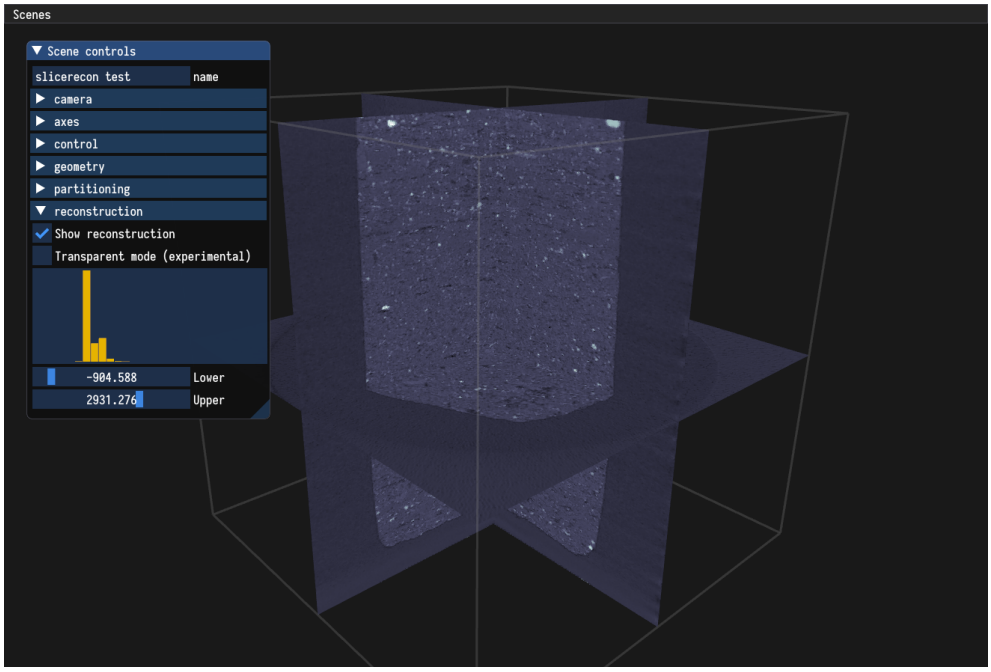


Figure 6.1: Overview of the RECAST3D interface. A number of arbitrarily oriented slices are chosen by a user using a simple, intuitive interface. Reconstructions are continuously updated as new data comes in, giving real-time visual feedback during time-resolved tomography experiments. The slices can be reoriented as necessary without any noticeable impact on the reconstruction time. Various controls for adjusting visualisation and reconstruction parameters are shown on the left.

6.1.1 GigaFRoST

The GigaFRoST [Mok+17] is a detection and readout system that can acquire and stream data continuously at 7.7 GB/s to a dedicated backend server. Coupled to a scintillator screen and efficient optics, this hardware unlocks unprecedented time-resolved tomographic microscopy capabilities, including simultaneously an elevated time resolution and the ability to follow dynamic phenomena for a long time. Built on top of a commercial CMOS sensor, it does not have an on-board RAM as is typically the case for high frame rate cameras on the market optimised for burst operation, but it directly streams the acquired data through eight fibre-optics connec-

tions to a backend server. In this way the number of images that can be acquired in one sequence is not limited by the internal detector memory and sustained fast data acquisition is possible. The backend server collects the data blocks dispatched by the detector and reassembles them into projection images in a ring buffer. These frames can then be sent to any downstream process (e.g., reconstruction pipeline and file writer). For this purpose, a publishing process posts the data using a distributed message passing protocol based on ZeroMQ streams. In this way, simple direct access to the acquired images is guaranteed: any downstream process can subscribe to the ZeroMQ data stream published by the backend.

6.1.2 RECAST3D

The RECAST3D framework [Buu+18] provides a quasi 3D reconstruction of the scanned object by simultaneously reconstructing and visualising a set of arbitrarily oriented tomographic slices, which can be dynamically chosen by the user and are constantly updated in real time (Figure 6.1). To derive a computationally efficient technique for reconstructing such arbitrarily oriented slices, we first note that the reconstruction problem in tomography can be modelled as a linear system $A\mathbf{x} = \mathbf{b}$. Here, $\mathbf{x}$ has a component for each of the $N_x \times N_y \times N_z$ voxels in the discretised representation of the object being imaged, $\mathbf{b}$ is the collection of (preprocessed) intensity measurements obtained on the detector, and A is the forward-projection operator, with a_{ij} the contribution of voxel j on intensity measurement i. A is sparse with only $O(N_\phi)$ nonzero entries in each column, where N_ϕ is the number of projection angles. This sparsity can be used to efficiently compute reconstructed slices using the filtered backprojection (FBP) technique. FBP is a popular reconstruction technique, because it is computationally efficient, straightforward to implement[PSV09], and provides high-quality reconstructions if a sufficient number of projections is available and the noise level is limited. An FBP reconstruction consists of two steps: first, the data is filtered, and afterwards, the filtered data is backprojected into the image array to produce the final reconstruction. Using the notation above, FBP can be written as

$$\mathbf{x} = A^T C\mathbf{b}, \tag{6.1}$$

where C is a filtering operation that performs 1D convolutions on each individual row of the projection images.

The key to the RECAST3D approach is that only a limited number of components of $\mathbf{x}$ needs to be computed for arbitrarily oriented slices, namely those corresponding to voxels of the slices. Without loss of accuracy, this can be done efficiently using an FBP algorithm. First, filtered projections $\mathbf{y} = C\mathbf{b}$ can be computed relatively easily in real-time, because the computation is trivially parallel (each row of each projection can be filtered independently) and because the 1D convolution operations can be efficiently calculated as element-wise multiplications in the Fourier domain. Second, since each column of A only contains N_ϕ nonzeros, the reconstructed value for a single voxel at any arbitrary position in the volume is given by a weighted sum of N_ϕ (filtered) data elements (see equation (6.1)). As a result, the reconstruction of an arbitrarily oriented slice with n^2 voxels requires only $O(n^2 N_\phi)$ operations, which is significantly less computationally demanding than the reconstruction of the full 3D n^3 voxels volume ($O(n^3 N_\phi)$ operations), since n is typically as high as a few thousand. In addition, for the reconstruction of an arbitrarily oriented slice, the system in equation (6.1) can be reduced to include only the information relevant to the voxels of interest:

$$\begin{bmatrix} \mathbf{x}_{\text{slice}} \\ \mathbf{x}_{\text{other}} \end{bmatrix} = \begin{bmatrix} A_{\text{slice}} & A_{\text{other}} \end{bmatrix}^T \mathbf{y} \qquad \Rightarrow \qquad \mathbf{x}_{\text{slice}} = A_{\text{slice}}^T \mathbf{y}. \tag{6.2}$$

Because this reduced system still represents a backprojection operation, existing efficient and highly flexible GPU based backprojection routines (e.g., those found in the ASTRA toolbox[Aar+16]) can be readily used to compute arbitrarily oriented slices without modification. The local properties exploited in the presented approach are specific to the FBP algorithm. Other reconstruction techniques, such as *gridrec* [Dow+99], which revolve around regridding of the data in Fourier space, can be up to 20 times faster for full 3D data sets than FBP [MS12], but cannot be restricted to reconstruct arbitrarily oriented slices, since they rely on a Fourier inversion of the entire volume.

The quasi-3D reconstruction pipeline of RECAST3D [Buu+18] is built upon a message-passing protocol between a visualisation tool for reconstructed slices and a reconstruction server. The reconstruction server holds (preprocessed) tomographic projections in memory, and is able to recon-

struct arbitrarily oriented slices from this data on demand, e.g., by dynamic selection by the user in the visualisation interface (Figure 6.1). A low-resolution 3D preview is provided by the reconstruction server as well, to aid the user while selecting slice positions and orientations. The active set of projections is continuously being updated during the scan, ensuring that the current state of the scanned object is always visualised.

6.1.3 Integration

The GigaFRoST system and the RECAST3D reconstruction pipeline are linked through a distributed message passing protocol based on ZeroMQ streams, which abstract away much of the network communication. The reconstruction server subscribes to the backend server stream to obtain, in real time, the projections from the tomographic measurement. Currently, a single workstation is used for reconstruction and visualisation. This workstation consists of an NVIDIA Quadro K6000 GPU with 12GB on-card memory, and two Intel Xeon CPU E5-2680 v2 CPUs. Projections are received from the backend server over a 10 Gbit network connection.

The tomographic measurement consists of multiple scans. In each scan, a data frame of N_ϕ projections is recorded. These projections are pre-processed and filtered as they come in by the combined 40 independent hardware threads of the CPUs, and then uploaded to GPU memory. The implementation also supports optional phase retrieval using the Paganin method [Pag+02]. The GPU holds two buffers, each large enough to store a data frame. The active buffer is always the latest complete data frame that has been fully processed and uploaded. New slice reconstructions are triggered in two ways: (i) when the user interactively chooses a new slice to be visualised, typically by translating or rotating one of the active slices in the visualisation tool, and (ii) when a new data frame has been fully processed and uploaded. A reconstruction is realised by a single backprojection operation onto a slice from the data in the active buffer, cf. Equation (6.2). Additionally, a low-resolution 3D volume is reconstructed when a new data frame has been fully processed and uploaded. A separate process handles the connection to a visualisation server, sending new reconstruction data when it becomes available. Optionally, remote observers can connect through an internet connection to the reconstruction server, and can request slice reconstructions independently from the on-site user. The

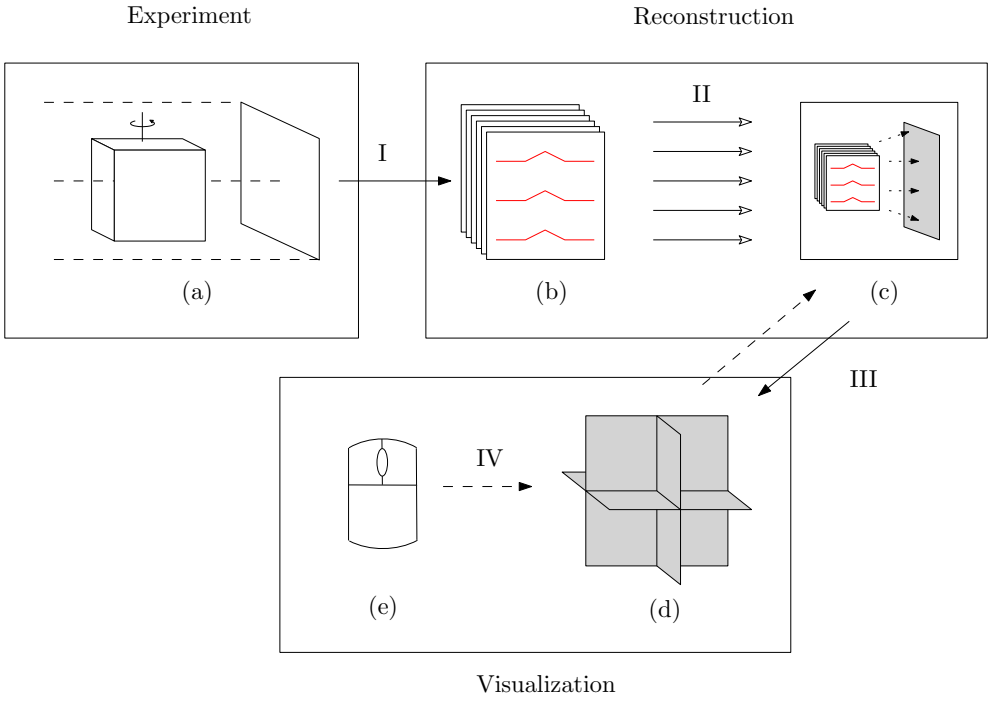


Figure 6.2: A tomographic measurement (a) leads (I) to a stack of projection images (b). The rows of these images have to be filtered (in red), which can be done in parallel (II). The filtered projection images can be used to reconstruct individual slices (in grey), by local backprojection operations (c). These slices can be shown together (d). The visualisation software can request reconstructions (III), in particular upon interactive slice rotation and translation (IV) by the user (e).

overall setup is illustrated in Figure 6.2. The IT infrastructure is illustrated in Figure 6.3.

Benchmarking results of the current implementation are presented in Table 6.1. There are two main performance aspects to consider. The first aspect is the time it takes to process and upload a data frame to the GPU. From the results, we see that the setup is capable of processing a set of 400 projections with 768×520 pixels and uploading it to the GPU well within a single second. One reason we do not use the full detector resolution, is to ensure that the projection data fits in the memory of the used GPU. The GPU memory usage is dominated by the active and passive projection

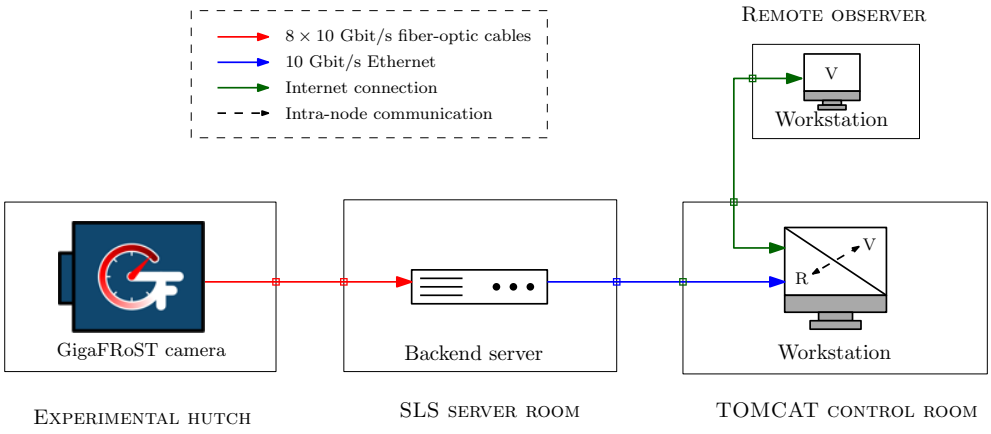


Figure 6.3: The IT infrastructure used in the real-time reconstruction pipeline. The data from the detector is received by a ring buffer on the backend server. This data is then streamed to the reconstruction software (R) currently running on a single workstation in the control room. The communication between the visualisation software (V) and the reconstruction software now happens within this single workstation.

buffers. Using the full GigaFRoST detector resolution of 2016×2016 for 400 projections would lead to a memory usage for the projection buffers of roughly $2016^2 \times 400 \times 2 \times 4\,\mathrm{B} \approx 13\,\mathrm{GB}$ when the values are stored in single precision. The other buffers, one to store a single reconstructed slice and one for the low-resolution 3D volume, take up a negligible amount of memory. Besides affecting memory usage, limiting the size of the projections also reduces the computational load in the preprocessing step, as well as the bandwidth required to upload the data to the GPU in time. The restriction on the size of the projection data can be lifted by using a GPU with more memory, or, as we discuss later, by moving to an implementation that uses multiple GPUs. We are currently able to realise a raw data bandwidth of roughly 4 Gbit/s. If required, only a part of the data is selected for use in the real-time reconstruction, to ensure that the incoming data can be processed and uploaded in time. In practice, this means that the reconstruction shown in the visualiser always comes from data that has been recorded less than one second earlier. The second performance aspect to consider is the time it takes to reconstruct an arbitrary slice from the active data buffer. Because of the way slices are reconstructed, it is con-

venient to choose a fixed size for the slices regardless of orientation. This is a parameter that can be set by the user. We use voxel-driven backprojection, and sampling is done by interpolating values of the projection images. For this benchmark, we choose to reconstruct slices with a relatively high resolution of 1024×1024 pixels to obtain a conservative estimate for the maximum reconstruction time. The total response time between the reconstruction and visualisation server, i.e., the time between requesting a slice reconstruction and receiving it, is less than 100 ms, realising the goal of being able to examine the imaged sample in real time. In summary, we see that with this implementation the time elapsing between the selection of a new slice by the user and its visualisation is negligible, so that it is as though fully reconstructed 3D data is available.

The connection between the backend server and the RECAST3D service is realised using a publish/subscribe pattern. The subscriber listens to messages sent by the publisher. In our case, a single message corresponds to one projection image. Using the ZeroMQ implementation of this pattern, message order is maintained between the publisher and subscriber, and messages are received at most once. However, there are no other strict guarantees on the messages. For example, it is not guaranteed that all messages are received by the subscriber. Our setup is mostly robust to missing messages, as the corresponding part of the buffer will be filled with zeros. When a backprojection operation using this buffer is executed, the missed projection images are then effectively ignored. In the worst case, this can result in missing angle artefacts when the number of dropped images is large, and it can reduce the overall intensity of the reconstructed image.

The employed scheme gives a lot of flexibility to the system. Listeners can subscribe and unsubscribe on demand, without requiring any additional logic to be implemented on the backend server. In the current implementation, messages are queued when the subscriber is overworked, and once this queue is full messages will start to drop. This can happen when we deal with particularly high-throughput data. One possibility to resolve this issue is to only send every Nth data frame to RECAST3D, for some appropriate value of N, while all get saved to disk, which would require inserting a ZeroMQ stream splitter into the stream.

In order to support higher resolution data sets, or to increase the number of data frames the pipeline can process, we have to move beyond using a single GPU. While currently only a single workstation is used for recon-

struction and visualisation, the framework is scalable. Multiple compute nodes can be used for processing and reconstructing in parallel. One way to achieve this is to split a data frame into groups of projections and distribute them over a number of GPUs. Each group of projections can be filtered independently. When a slice reconstruction is requested, each GPU performs a backprojection with its local group of projections leading to a contribution to the reconstructed slice. Next, we perform a single distributed step over all GPUs, where the contributions are summed to obtain the slice reconstruction for the full projection set. We note that this summation is performed only on 2D data, limiting the required communication between GPUs as well as the computational cost. This parallelization method is possible because the backprojection operator is linear. In the current implementation, the CPU-based pre-processing could form a bottleneck to the scalability. However, expensive steps such as filtering the projections could be offloaded to a GPU. Based on the results obtained with a modest workstation, we expect that when a small-size cluster of about 8 GPU nodes is used, full-resolution tomographic reconstructions with a finer temporal resolution should be achievable.

It is also possible to further optimise the GigaFRoST system for the specific application of real-time visualisation and feedback. The service running on the GigaFRoST backend server is currently implemented as a ring buffer, and it does not guarantee streaming out the frame data in consecutive order. This is in order to optimise performance when it is under heavy load. Although ZeroMQ streams guarantee maintaining message order, this means we cannot rely on this in practice, because images from successive data frames can intermix and throw off the processing and updating of the active buffer. To circumvent this issue for the present experiments, we chose a sufficiently long wait period between individual scans. After a planned modification to the service running on the backend server, this should no longer be necessary in the future.

Our primary aim of the proposed pipeline is not to outperform the already established pipeline running on a large CPU cluster in the reconstruction of complete 3D data sets. Instead, the main advantages of the proposed pipeline over the existing pipeline are: (i) Slices with arbitrary orientations through the volume can be reconstructed. This would not be possible on the existing production cluster, which relies on the gridrec algorithm instead of FBP and would thus first need to reconstruct the full

PROCESS	UPLOAD	PREVIEW	SLICE	TOTAL FOR THREE SLICES
386.3 ms	197.9 ms	49.1 ms	31.5 ms	727.9 ms

Table 6.1: Benchmark results for the reconstruction pipeline. Each data frame contains 400 projections with 768 × 520 pixels. The reconstructed slices consist of 1024 × 1024 pixels. The reconstructed 3D preview consists of 128 × 128 × 128 voxels. Here, PROCESS is the processing time for a single data frame, e.g., flat fielding and filtering. The total time to upload a data frame to the GPU is shown as UPLOAD. The reconstruction time for processed data stored on the GPU for a 2D slice and a 3D preview is given as SLICE and PREVIEW respectively. Although many of the steps happen in parallel, a worst-case estimate for the processing of a single data frame and reconstruction of three arbitrary slices can be found by the sequential time computed as PROCESS + UPLOAD + PREVIEW + 3 × SLICE. This estimate is shown as TOTAL FOR THREE SLICES.

volume before being able to compute and visualise an arbitrarily oriented data slice through the volume. Due to this, the performance gain for visualising arbitrarily oriented slices is over a factor of 10 compared to the production pipeline. (ii) The current production environment lacks the interactive visualisation environment provided by RECAST3D, and thus also the capability to choose and adjust the requested slice positions dynamically during the running measurement. (iii) The proposed system is designed to run on a very simple and modest compute infrastructure compared to the relatively large CPU cluster required by the existing pipeline.

6.2 Scientific applications

In this section, the new features, current benefits and future potential of the presented real-time reconstruction and visualisation tools are illustrated on a selected case study which is, however, representative of a wide range of dynamic phenomena.

Fluid uptake characteristics and transport mechanisms in fibrous materials are widely and intensively studied on a very fundamental level, both experimentally [Zha+17; Par+19] and through models and simulations [Kis16; LCL08], for a variety of technical applications, ranging from the impregnation of carbon fibre composite materials with a fluid polymer mat-

rix, via the wettability and absorption of ink in paper and cloth-based carrier materials during ink jet printing, to the functionalisation of wearable textiles to control their water-repellent or moisture absorbing and transporting properties.

The wicking behaviour of a single yarn thread is investigated in a simple dynamic model experiment. A yarn is essentially a spun bundle of individual fibres. Depending on the fibre material, size distribution and homogeneity, and the tension and twist applied during spinning, the volume, distribution, shape, and inter-connectivity of pore spaces within the yarn differ significantly and, in turn, crucially affect the water transport and distribution within the yarn.

Figure 6.4 shows a sketch and photo of the setup used in the experiment. A yarn has been fabricated from 96 polyethylene terephthalate (PET) fibres of 22 μm diameter. It is mounted inside a vertically positioned kapton tube of ca. 6 mm diameter and 50 mm length and is subject to a slight amount of twist and tension. The bottom part of the kapton tube features an aperture to allow liquid to enter in order to get the lower end of the yarn in contact with water. The tube is placed into a larger reservoir holder into which one can inject the liquid from a remotely controlled syringe pump. The whole assembly is mounted on the rotation stage in the beamline hutch and can be positioned such that the yarn is centred along the rotation axis.

Edge-enhanced X-ray absorption images are produced using the filtered (20 mm pyrolitic graphite + 75 um W) white beam of a 2.9 T superbending magnet, converted to visible light with a 150 μm thick LuAG:Ce scintillator (Crytur, Czech Republic), and recorded using the GigaFRoST camera coupled to a high numerical aperture microscope [Büh+19] (Optique Peter, France) featuring an optical magnification of 4x. This results in an effective pixel size of 2.75 μm. The scintillator was placed 320 mm downstream from the sample to obtain some degree of edge-enhancement from the weakly absorbing PET fibres. Projection images were cropped to a size of 384 pixels horizontally by 800 pixels vertically to capture the full extent of the yarn illuminated by the approximately 2.2 mm high X-ray beam.

The experimental challenge for this system is twofold: Firstly, more than one transport mechanism governs the evolution of the water content in the yarn. These processes inherently proceed at different speeds and can result in abrupt changes of the uptake velocity over time. To capture

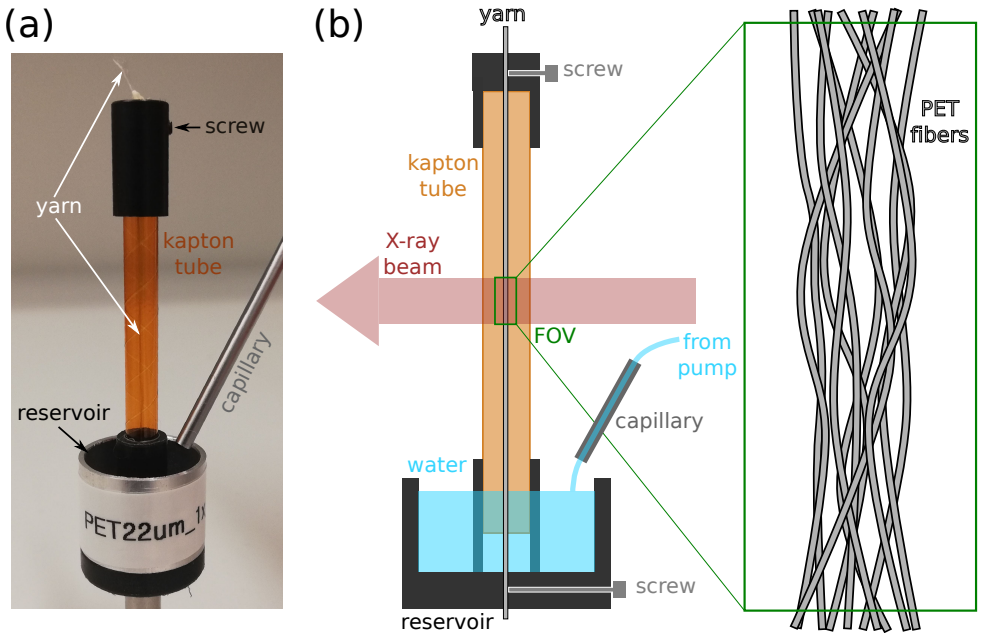


Figure 6.4: Experimental setup. (a) Photo of the yarn sample holder. (b) Schematic drawing of the sample holder and measurement geometry.

the fast dynamics, scan times for individual volume reconstructions need to be kept as short as possible, ideally of the order of 0.1 - 0.5 seconds. Secondly, the arrival time point of the liquid front at the measurement position, which lies 10-20 mm above the water surface level in the reservoir, is very unpredictable and varies considerably from specimen to specimen. Hence, the data acquisition needs to be sustained at high speeds over a long period of time, thus putting stringent demands on the data streaming and storage infrastructure. In the end, the interesting dynamics will be restricted to only a short period during this extended time series, rendering most of the data unimportant.

The experiment then proceeds as follows: First, the rotation of the dry sample is started and the acquisition of projection images with the GigaFRoST camera [Mok+17] is initiated. To ensure an identical sample orientation for successive volume scans and to throttle the scan rate, we employ the so-called sequence mode for data acquisition [Lov+16], where the collection of a series of 400 images over a 180 degree range is triggered by

the position-sensitive output signal from the rotation stage every 720 degrees during continuous rotation. With the chosen exposure time of 1 ms for each projection, this results in a scan time of 0.4 seconds per scan and a scan period of 1.6 seconds.

We will now discuss three specific examples of capabilities that our approach enables in practice.

Capability I: real-time alignment of the setup

One of the first steps in any tomographic X-ray imaging experiment is the assessment and optimisation of the reconstruction image quality. Usually this is incrementally adjusted through a series of alignment procedures and test scans which have to be reconstructed and examined individually after each alignment step. Parameters to be aligned and optimised may include the tilt and position of the rotation axis with respect to the camera or the propagation distance between the sample and the scintillator to achieve the right amount of edge-enhancement. Performing these alignment and optimisation steps is greatly simplified and accelerated by the availability of a live view of reconstructed slices. These steps are demonstrated in movie S1 in the supplementary materials[1]. The rotation axis is initially offset with respect to the centre of the camera by a few tens of pixels, resulting in the characteristic C-shaped artefacts of the individual fibres comprising the sample structure in the axial slices of the live reconstruction. By simply tweaking the rotation axis' or camera's position transverse to the beam direction with the corresponding translation stage, one can progressively improve the quality of the reconstructed sample structure until an adequate alignment has been achieved. Note that when measuring radiation-sensitive samples, the alignment step should naturally be performed with a dedicated alignment tool before mounting the real samples. The precise centring of the rotation axis is, however, not strictly necessary to conduct an experiment, as the actual location can be determined in the reconstruction process and a slight misalignment can be easily corrected *a posteriori* to improve the reconstructed image quality. Similarly, in cases where a precise alignment during the experiment may not be possible due

[1]The supplementary materials referenced are for the publication on which this chapter is based.

to mechanical constraints, a non-centred axis position could be specified as an input for the real-time reconstruction via RECAST3D.

Capability II: real-time sample positioning

While the above mentioned optimisation and alignment steps usually only need to be performed at the beginning of an experiment series, each sample to be measured has to be positioned correctly with respect to the rotation axis to ensure that the proper region of interest (ROI) is imaged. For many samples, like the yarns in this experiment, this is easily achieved simply by looking at the projection images. However, particularly when looking at smaller regions inside an extended sample, navigating to the correct ROI simply based on the radiographic projections is often not straightforward. Again, a live view of a small number of reconstructed slices through the volume can easily guide the navigation and ensure that the proper region is imaged in the real experiment. An example of this live navigation inside a sample is seen in movie S2 in the supplementary materials[1], where the region of interest to be measured is the interface between two different mineral phases in a piece of volcanic rock. While the sample is continuously rotating, one can easily search for the desired location and accurately position it within the reconstructed field of view shown by the live preview of RECAST3D.

Capability III: real-time observation of water uptake

Much as the setup alignment and sample positioning are facilitated by the nearly real-time visualisation of reconstructed slices, the main purpose of the presented tool is to allow for the live observation of a dynamical process as it is happening during an experiment. In the case of our yarn sample, this means the observation of the waterfront arrival in the imaged sample region and the subsequent filling of the full yarn's pore structure with liquid. Once the sample is completely wetted, the measurement can be stopped to avoid the acquisition of unnecessary data. The screencast movie S3 in the supplementary materials[1] shows the whole temporal evolution of the observed sample structure according to the experimental procedure outlined above.

Figure 6.5 shows some representative time points of a data set acquired on an identically prepared sample with 32 fibres instead of 96, imaged under the same experimental conditions. The only difference with respect to the measurement with the live preview was that instead of acquiring one 180 degree scan every two full rotations, a full data set was recorded once per turn, resulting in a scan period of 0.8 seconds instead of the 1.6 seconds used for the scan series visualised on-the-fly with RECAST3D. Panel (a) of figure 6.5 shows flat-field-corrected projection images, or so-called radiographies, of the full extent of the imaged yarn section at the beginning and the end of the water uptake process. Magnified views of small sections at the top and bottom of this imaged region are shown in panel (b) for different time points during the scan series. This is the direct visual feedback tool used so far at most tomographic microscopy beamlines to follow the dynamics of the investigated process. It is essentially impossible to determine when the water front arrives in the two different regions from these projection images as the change in contrast is very small. However, the situation changes dramatically when looking at tomographic slices of the phase-contrast reconstruction. Vertical slices through the centre of the full reconstructed volume are shown for the beginning and the end of the scan series in panel (c). Using axial cuts at the top and bottom of the sample volume, as shown in panel (d), we can readily detect the arrival time point of the leading edge of the water front in the bottom of the imaged region between around 11.2 seconds (still dry) and 12.0 seconds (some pore spaces are filled with water). The same effect is visible in the top slice about 2.5–3 seconds later. The entire time series of flat- and dark field corrected radiographic projection images as well as for the top and bottom reconstructed slices are shown in the supplementary materials movies S4, S5, and S6, respectively[1]. A visual rendering of the full 3-dimensional structure (which would not be available in real-time with RECAST3D) for one time point is shown in panel (e) of figure 6.5 with a red semi-transparent isosurface of the water and fibre structure, three axial slices at the bottom, middle, and top of the sample and a vertical slice through the centre of the fibre bundle.

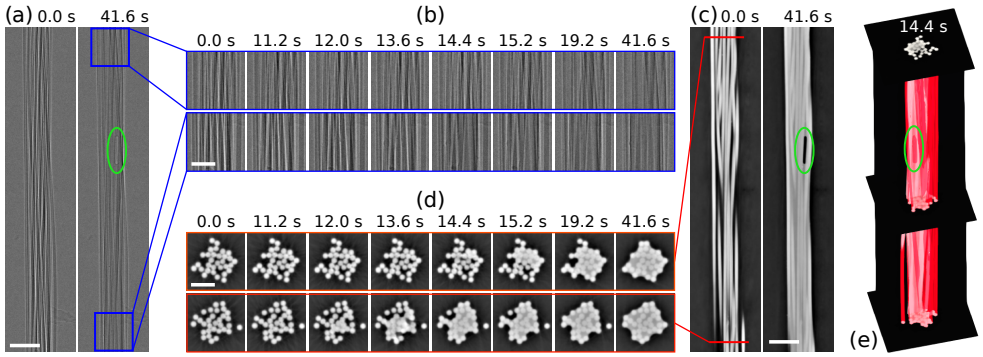


Figure 6.5: Time series of scans showing the water uptake dynamics of the yarn. (a) Radiographic projection images show the whole imaged yarn section at the beginning (0.0 seconds, dry) and the end (41.6 seconds, nearly completely wetted) of the uptake process. Scale bar: 200 μm (b) Magnified sections of the radiographic images, indicated by the blue boxes in (a), at several time points during the scan at the top (upper line) and bottom (lower line) of the sample. Scale bar: 100 μm (c) Vertical centre slice through the phase contrast reconstruction of the yarn sample at the beginning and the end of the scan series. Scale bar: 200 μm (d) Horizontal cuts, indicated by the red line in (c), through the reconstructed volume at the top and bottom at the same time points shown in (b). Scale bar: 100 μm (e) Rendering of the reconstructed volume at an intermediate time point during the uptake process, showing one vertical and three axial slices as well as a semi-transparent red isosurface outlining the volume of the combined yarn and water volume. Green outlines: Air bubble emaining in the yarn structure even at the end of the scan series.

6.2.1 Discussion

In many cases, the information that can be gained from strategically chosen arbitrarily oriented reconstructed slices is a good proxy for the dynamic evolution of the entire sample and is sufficient for adaptive experimental control purposes. By positioning reconstructed slices for instance perpendicular to a front evolution direction, liquid breakthrough can easily be detected. Alternatively, reconstructed slices oriented parallel to it could give a real-time indication of the speed of the propagating front enabling on-the-fly adjustment of the experimental parameters. We believe that if

the reconstructed slices are carefully chosen, quasi-3D reconstructions, as the one used in this yarn example, can in many cases provide valuable and representative information for the full 3D structure. The number and type of measurements which could profit from an active automatic feedback will increase with time as new and increasingly optimised tools are being developed by the large and very active image analysis community.

6.3 Outlook: A route towards adaptive experiment control

The unprecedented possibility provided by the tools presented here to directly visually follow live dynamic processes as they happen is very valuable for enabling adaptive control of the experiment, for instance to stop the image acquisition and the experiment when the phenomenon of interest is over, so avoiding the storage of a large amount of useless data. Another important application is the case of systems where sudden high-speed events of interest happen only occasionally at essentially unpredictable time points.

Access to a few nearly real-time reconstructed slices through the volume in specifically controlled locations opens up the possibility to go even further and to perform quantitative online analysis on these data. Here we sketch a possible route towards an online feedback mechanism for the presented example of water wicking in yarns, in particular, where we aim to identify the time points of water arrival at the bottom and the top of the imaged sample region, as well as the saturation of the pore volume with water.

Judging by the phase-reconstructed slices in Figure 6.5(d), identifying the arrival time point of the waterfront should be relatively straightforward. A simple approach relies on the ability to automatically segment the slice data into air and material (in this case, both water and yarn are classified as material). Since the volume of the yarn does not change during the experiment, any change in the amount of detected material can be attributed to water, and a significant rate of change should only be observed starting with the arrival of the water front. Figure 6.6 plots the total number of pixels per slice classified as material as a function of the scan time. The arrival of the water front is clearly identified as the point when

the material fraction suddenly increases. Consistent with the visual inspection, the top slice starts to gain in material about 2 – 3 seconds after the bottom slice. The small insets show the segmented slices from the bottom of the imaged volume at different time points, using a constant threshold which was determined automatically using the Otsu [Ots79] method on the dry fibre bundle corresponding to the first time point.

Active feedback to the experiment control in this case could be to start saving data only once the arrival of the water front in the bottom-most slice has been detected and to stop recording data once the water content in both the top and bottom slices has not changed considerably over a given time period. Another option would be, for example, to automatically deliver a staining agent to the water reservoir once the unstained waterfront has reached the imaging region such that the liquid transport in the already wetted yarn can be observed under identical experimental conditions in the same sample as the initial wetting behaviour.

Combining our proposed approach for real-time reconstruction with application-specific postprocessing and visualization operations, the present example can easily be adapted for a broad range of other use cases where the state of the sample must be probed and analysed in real-time to allow for on-the-fly adaptation of experimental parameters.

6.4 Conclusions

The present study demonstrates the feasibility, utility and further potential of the real-time reconstruction of a small number of arbitrarily oriented slices to visually observe the evolution of a sample and to obtain quantitative feedback of the dynamic phenomena occurring during tomographic imaging. The real-time reconstruction has been realised at the TOMCAT beamline at the Swiss Light Source (PSI), and only requires a single workstation for the computations. The chosen approach carefully balances the relative trade-offs between the achievable reconstruction speed, the complexity and cost of the necessary IT infrastructure, and the completeness of the available subset of data during online processing to deliver a powerful quantification and visualisation tool that can be relatively easily integrated into existing data acquisition pipelines with only modest investments in the necessary computing resources.

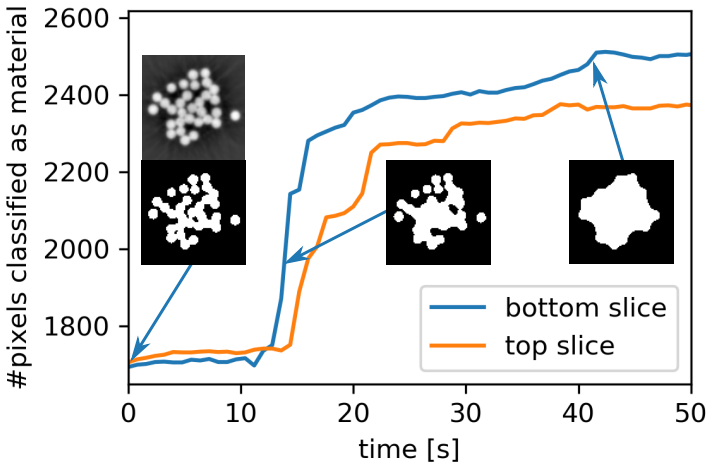


Figure 6.6: Quantification of the water uptake in the bottom and top slice as a function of time. The reconstructed slices are segmented using a constant threshold and the total number of pixels classified as material is plotted. The insets show the segmented bottom slices at different time points (along with the reconstructed grey-level image for the first time point which was used to automatically determine the threshold for the segmentation).

Chapter 7

Conclusion

The goal of the research presented in this dissertation was to develop techniques that enable real-time tomographic reconstruction. In particular, we wanted to reduce the runtime of reconstruction algorithms so that they are in the same ballpark as the time it takes to acquire the projection images.

Throughout this research, my focus has always been on making the developed methods applicable to as many use cases as possible, which resulted in the polyvalent character of the developed methods. Few assumptions are made about the acquisition geometry, reconstructions methods, or application area.

This genericity is exemplified by the Bulk framework in Chapter 2. Although my personal motivation was for Bulk to aid with the implementation of distributed tomographic reconstruction, I tried consciously to not let this use case have too big of an influence on the design decisions that have been made. As a result, the framework should prove useful for many applications in parallel scientific computing.

The partitioning techniques presented in Chapters 3 and 4 can be used for accelerating the forward and backprojection projection operations by employing multiple GPUs simultaneously. Compared to the state-of-the-art, these partitionings enable scaling to many more GPUs than was previously achievable. Together with the newly proposed data structures, I believe these techniques improve greatly upon previously available solutions. An important class of often used reconstruction methods alternates between performing forward and backprojection operations. The data distributions generated by these novel partitioning methods represent a big

step forward in reducing the runtime of these reconstruction methods.

By employing domain-specific information, in our case the acquisition geometry of the tomography experiment, we were able to improve upon existing partitioning techniques. In particular, we could make them scale way beyond what is achievable by solely looking at the nonzero pattern of the corresponding sparse matrix. This makes you wonder: What other application areas could benefit from similarly incorporating such domain-specific information?

The quasi-3D reconstruction technique introduced in Chapter 5 decreases the runtime of reconstruction algorithms such as FBP and FDK by orders of magnitude compared to full 3D reconstructions. For many use cases, quasi-3D reconstructions contain enough qualitative and quantitative information on the imaged object. An important feature is that this technique can be used for many imaging modalities. It has already been successfully applied to μ-CT systems such as the FleX-ray lab at CWI in Amsterdam, synchrotron tomography as discussed in detail in Chapter 6, and electron tomography in a collaboration with EMAT in Antwerp.

Together, the geometric partitioning techniques and quasi-3D reconstruction can accelerate almost every tomographic reconstruction method used in practice. However, reconstruction is only one step of the imaging pipeline. It is usually followed by a post-processing and analysis step. One of the goals of real-time tomographic reconstruction is to enable direct feedback, in order to steer the experiment while it is ongoing. The range of possible applications will increase greatly if post-processing and analysis can also be performed in real time, and this could certainly be an interesting avenue for future research.

The quasi-3D methodology can also aid in realizing this final step of real-time post-processing and analysis. RECAST3D, our software package that implements a full quasi-3D reconstruction pipeline, boasts a powerful plugin system. Plugins that have already been developed include those for real-time segmentation, and the real-time analysis of the curvature of a nanoparticle. Furthermore, more advanced filtered backprojection type algorithms can easily be implemented directly in RECAST3D. We have already seen promising results for improving the image quality of the reconstructions by, e.g., using algebraic filters, computing a combination of filters using neural networks, and by using computer vision techniques to reduce noise and other image artifacts.

By collaborating with experimental imaging groups, I was lucky enough to see my research be useful for imaging experiments. This has been very rewarding, and I hope that in particular Bulk and RECAST3D will aid others in performing their research, and to apply their methods in practice.

Bibliography

[Aar+15] W. van Aarle et al. “The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography”. In: *Ultramicroscopy* 157 (2015), pp. 35–47.

[Aar+16] W. van Aarle et al. “Fast and Flexible X-Ray Tomography Using the ASTRA Toolbox”. In: *Optics Express* 24.22 (2016), p. 25129.

[AKÖ17] J. Adler, H. Kohr and O. Öktem. *Operator Discretization Library*. `https://github.com/odlgroup/odl`. 2017.

[And84] A. Andersen. “Simultaneous Algebraic Reconstruction Technique (SART): A Superior Implementation of the ART Algorithm”. In: *Ultrasonic Imaging* 6.1 (1984), pp. 81–94.

[Atw+15] R. C. Atwood et al. “A high-throughput system for high-quality tomographic reconstruction of large datasets at Diamond Light Source”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 373.2043 (2015), p. 20140398.

[Avi01] M. S. Avinash C. Kak. *Principles of Computerized Tomographic Imaging*. Society for Industrial and Applied Mathematics, 2001.

[BB02] S. Basu and Y. Bresler. “$O(N^3 \log N)$ backprojection algorithm for the 3-D Radon transform”. In: *IEEE Transactions on Medical Imaging* 21.2 (2002), pp. 76–88.

[BB87] M. J. Berger and S. H. Bokhari. “A partitioning strategy for nonuniform problems on multiprocessors”. In: *IEEE Transactions on Computers* C-36.5 (1987), pp. 570–580.

[BBB18] J. W. Buurlage, T. Bannink and R. H. Bisseling. “Bulk: a Modern C++ Interface for Bulk-Synchronous Parallel Programs”. In: *Euro-Par 2018: Parallel Processing*. Vol. 11014. Lecture Notes in Computer Science. Springer, 2018, pp. 519–532.

[BBB19] J. W. Buurlage, R. H. Bisseling and K. J. Batenburg. “A Geometric Partitioning Method for Distributed Tomographic Reconstruction”. In: *Parallel Computing* 81 (2019), pp. 104–121.

[Ber+08] M. de Berg et al. *Computational Geometry: Algorithms and Applications*. 3rd. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[BG05] T. M. Benson and J. Gregor. “Framework for iterative cone-beam micro-CT reconstruction”. In: *IEEE Transactions on Nuclear Science* 52.5 I (2005), pp. 1335–1340.

[Bic+15] T. Bicer et al. “Rapid tomographic image reconstruction via large-scale parallelization”. In: *European Conference on Parallel Processing*. Springer. 2015, pp. 289–302.

[Bic+17] T. Bicer et al. “Real-Time Data Analysis and Autonomous Steering of Synchrotron Light Source Experiments”. In: *2017 IEEE 13th International Conference on e-Science (e-Science)*. Oct. 2017.

[Bil+12] J. Bill et al. “DendroCT – Dendrochronology without damage”. In: *Dendrochronologia* 30.3 (2012), pp. 223–230.

[Bis04] R. H. Bisseling. *Parallel scientific computation: a structured approach using BSP and MPI*. Oxford University Press, Oxford, UK, 2004, p. 325.

[BJ92] T. N. Bui and C. Jones. “Finding good approximate vertex and edge partitions is NP-hard”. In: *Information Processing Letters* 42.3 (1992), pp. 153–159.

[Bon+03] O. Bonorden et al. “The Paderborn University BSP (PUB) library”. In: *Parallel Computing* 29.2 (2003), pp. 187–207.

[BT09] A. Beck and M. Teboulle. “A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems”. In: *SIAM Journal on Imaging Sciences* 2.1 (2009), pp. 183–202.

[Büh+19] M. Bührer et al. “High numerical aperture macroscope optics for time-resolved experiments”. In: *Journal of Synchrotron Radiation* in press (2019).

[Buu+18] J.-W. Buurlage et al. “Real-Time Quasi-3D Tomographic Reconstruction”. In: *Measurement Science and Technology* (2018).

[Buz08] T. M. Buzug. *Introduction to Computed Tomography From Photon Statistics to Modern Cone-beam CT*. Springer, 2008.

[CA01] U. Catalyurek and C. Aykanat. “A fine-grain hypergraph model for 2D decomposition of sparse matrices”. In: *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*. 2001.

[CA99] U. Catalyurek and C. Aykanat. “Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication”. In: *IEEE Transactions on Parallel and Distributed Systems* 10.7 (1999), pp. 673–693.

[CBB18] S. B. Coban, J. Buurlage and K. J. Batenburg. *Two cone beam test dataset for RECAST3D*. `https://doi.org/10.5281/zenodo.1154166`. Jan. 2018.

[Chi+11] S. Chilingaryan et al. “A GPU-based architecture for real-time data assessment at synchrotron experiments”. In: *IEEE Transactions on Nuclear Science*. Vol. 58. 4 PART 1. 2011, pp. 1447–1455.

[Chi+15] A. Ching et al. “One trillion edges: graph processing at Facebook-scale”. In: *VLDB* 8.12 (2015), pp. 1804–1815.

[CP10] A. Chambolle and T. Pock. “A First-Order Primal-Dual Algorithm for Convex Problems With Applications To Imaging”. In: *Journal of Mathematical Imaging and Vision* 40.1 (2010), pp. 120–145.

[Dev+06] K. D. Devine et al. “Parallel hypergraph partitioning for scientific computing”. In: *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*. 2006.

[Dev+16] M. Deveci et al. “Multi-Jagged: a scalable parallel spatial partitioning algorithm”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.3 (2016), pp. 803–817.

[DG04] J. Dean and S. Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Proc. OSDI* (2004), pp. 137–149. arXiv: 10.1.1.163.5292.

[DH11] T. A. Davis and Y. Hu. “The University of Florida sparse matrix collection”. In: *ACM Transactions on Mathematical Software* 38.1 (2011), pp. 1–25.

[Dow+99] B. A. Dowd et al. “Developments in synchrotron x-ray computed microtomography at the National Synchrotron Light Source”. In: *Developments in X-Ray Tomography II*. Sept. 1999, nil.

[FDK84] L. Feldkamp, L. Davis and J. Kress. “Practical cone-beam algorithm”. In: *Journal of the Optical Society of America A* 1.6 (1984), pp. 612–619.

[FJ98] M. Frigo and S. G. Johnson. “FFTW: An Adaptive Software Architecture for the FFT”. In: *Proc. IEEE ICASSP*. 1998, pp. 1381–1384.

[Gar+18] F. García-Moreno et al. “Time-resolved *in situ* tomography for the analysis of evolving metal-foam granulates”. In: *Journal of Synchrotron Radiation* 25.5 (Sept. 2018), pp. 1505–1508.

[GBH70] R. Gordon, R. Bender and G. T. Herman. “Algebraic Reconstruction Techniques (ART) for Three-Dimensional Electron Microscopy and X-Ray Photography”. In: *Journal of Theoretical Biology* 29.3 (1970), pp. 471–481.

[Ger15] A. V. Gerbessiotis. “Extending the BSP model for multi-core and out-of-core computing: MBSP”. In: *Parallel Computing* 41.Supplement C (2015), pp. 90–102.

[Gib+15] J. W. Gibbs et al. “The Three-Dimensional Morphology of Growing Dendrites”. In: *Scientific Reports* 5.1 (2015), p. 11824.

[Gil72] P. Gilbert. “Iterative Methods for the Three-Dimensional Reconstruction of an Object From Projections”. In: *Journal of Theoretical Biology* 36.1 (1972), pp. 105–117.

[Gür+14] D. Gürsoy et al. “TomoPy: a framework for the analysis of synchrotron tomographic data”. In: *Journal of Synchrotron Radiation* 21.5 (Sept. 2014), pp. 1188–1193.

[Hah+13] B. N. Hahn et al. “Combined reconstruction and edge detection in dimensioning”. In: *Measurement Science and Technology* 24.12 (2013), p. 125601.

[HDS97] J. M. D. Hill, S. R. Donaldson and D. B. Skillicorn. “Portability of Performance with the BSPLib Communications Library”. In: *Proc. MPPM*. 1997, p. 33.

[Hel+17] T. Heller et al. “HPX–An open source C++ standard library for parallelism and concurrency”. In: *Proc. OpenSuCo*. 2017, p. 5.

[Hel10] M. A. Helvie. “Digital Mammography Imaging: Breast Tomosynthesis and Advanced Applications”. In: *Radiologic Clinics of North America* 48.5 (2010), pp. 917–929.

[Her09] G. T. Herman. *Fundamentals of computerized tomography*. 2nd ed. Springer-Verlag London, 2009.

[HFE10] K. Hamidouche, J. Falcou and D. Etiemble. “Hybrid bulk synchronous parallelism library for clustered SMP architectures”. In: *Proc. HLPP*. Baltimore, Maryland, USA, 2010, pp. 55–62.

[Hil+98] J. M. D. Hill et al. “BSPlib: The BSP programming library”. In: *Parallel Computing* 24.14 (1998), pp. 1947–1980.

[HS52] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC, 1952.

[IB01] M. A. Inda and R. H. Bisseling. “A simple and efficient parallel FFT algorithm using the BSP model”. In: *Parallel Computing* 27.14 (2001), pp. 1847–1878.

[ISO17] ISO/IEC. *14882:2017(E) – Programming languages – C++*. Geneva, Switzerland, 2017.

[JKM11] N. Jain, M. S. Kalra and P. Munshi. “Characteristic Signature of Specimen Using an Approximate Formula for 3D Circular Cone-Beam Tomography”. In: *Research in Nondestructive Evaluation* 22.3 (2011), pp. 169–195. eprint: `https://doi.org/10.1080/09349847.2011.577270`.

[Kac37] S. Kaczmarz. “Angenäherte Auflösung von Systemen linearer Gleichungen”. In: *Bull. Int. Acad. Sci. Pologne, A* 35 (1937), pp. 355–357.

[Kat02] A. Katsevich. “Theoretically exact filtered backprojection-type inversion algorithm for spiral CT”. In: *SIAM Journal on Applied Mathematics* 62.6 (2002), pp. 2012–2026.

[Kat03] A. Katsevich. “A general scheme for constructing inversion algorithms for cone beam CT”. In: *International Journal of Mathematics and Mathematical Sciences* 21 (2003), pp. 1305–1321.

[Keß00] C. W. Keßler. “NestStep: Nested parallelism and virtual shared memory for the BSP model”. In: *J. Supercomputing* 17.3 (2000), pp. 245–262.

[Kis16] E. Kissa. “Wetting and Wicking”. In: *Textile Research Journal* 66.10 (2016), pp. 660–668.

[KS01] A. C. Kak and M. Slaney. *Principles of computerized tomographic imaging*. SIAM, 2001, p. 323.

[Kun+07] H. Kunze et al. “Filter determination for tomosynthesis aided by iterative reconstruction techniques”. In: *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*. 2007, pp. 309–312.

[LC+84] K. Lange, R. Carson et al. “EM reconstruction algorithms for emission and transmission tomography”. In: *Journal of Computer Assisted Tomography* 8.2 (1984), pp. 306–16.

[LCL08] T. Liu, K.-f. Choi and Y. Li. “Wicking in twisted yarns”. In: *Journal of Colloid and Interface Science* 318.1 (2008), pp. 134–139.

[Len90] T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. John Wiley and Sons, Chichester, UK, 1990.

[LFB10] Y. Long, J. A. Fessler and J. M. Balter. “3D Forward and Back-Projection for X-ray CT Using Separable Footprints”. In: *IEEE Transactions on Medical Imaging* 29.11 (2010), pp. 1839–1850.

[LGB05] F. Loulergue, F. Gava and D. Billiet. “Bulk Synchronous Parallel ML: modular implementation and performance prediction”. In: *Proc. ICCS*. 2005, pp. 1046–1054.

[Lov+16] G. Lovrić et al. “A multi-purpose imaging endstation for high-resolution micrometer-scaled sub-second tomography”. In: *Physica Medica* 32.12 (2016), pp. 1771–1778.

[Mai+16] E. Maire et al. “20 Hz X-ray tomography during an in situ tensile test”. In: *International Journal of Fracture* 200.1 (July 2016), pp. 3–12.

[Mal+10] G. Malewicz et al. “Pregel: a system for large-scale graph processing”. In: *Proc. SIGMOD* (2010), pp. 135–146. arXiv: arXiv:1503.00626v1.

[Mar+17] F. Marone et al. “Towards On-The-Fly Data Post-Processing for Real-Time Tomographic Imaging At TOMCAT”. In: *Advanced Structural and Chemical Imaging* 3.1 (2017), p. 1.

[Mas97] D. N. Mastronarde. “Dual-axis tomography: an approach with alignment methods that preserve resolution”. In: *Journal of Structural Biology* 120.3 (1997), pp. 343–352.

[MB04] B. D. Man and S. Basu. “Distance-driven projection and backprojection in three dimensions”. In: *Physics in Medicine and Biology* 49.11 (2004), pp. 2463–2475.

[MD09] P. A. Midgley and R. E. Dunin-Borkowski. “Electron tomography and holography in materials science”. In: *Nature Materials* 8.4 (2009), pp. 271–280.

[Mok+17] R. Mokso et al. “Gigafrost: the Gigabit Fast Readout System for Tomography”. In: *Journal of Synchrotron Radiation* 24.6 (2017), pp. 1250–1259.

[Moo+13] J. Moosmann et al. “X-ray phase-contrast in vivo microtomography probes new aspects of Xenopus gastrulation”. In: *Nature* 497.7449 (2013), pp. 374–377.

[MPI94] MPI Forum. “MPI: A Message-Passing Interface Standard”. In: *International Journal of Supercomputer Applications and High-Performance Computing* 8 (1994), pp. 165–414.

[MPS10] M. Maisl, F. Porsch and C. Schorr. “Computed laminography for x-ray inspection of lightweight constructions”. In: *2nd International Symposium on NDT in Aerospace* (2010), pp. 2–8.

[MS12] F. Marone and M. Stampanoni. “Regridding Reconstruction Algorithm for Real-Time Tomographic Imaging”. In: *Journal of Synchrotron Radiation* 19.6 (2012), pp. 1029–1037.

[Nik+17] V. V. Nikitin et al. “Fast hyperbolic Radon transform represented as convolutions in log-polar coordinates”. In: *Computers & Geosciences* 105 (2017), pp. 21–33.

[NR98] R. W. Numrich and J. Reid. “Co-array Fortran for parallel programming”. In: *ACM SIGPLAN Fortran Forum* 17.2 (1998), pp. 1–31.

[NW01] F. Natterer and F. Wübbeling. *Mathematical Methods in Image Reconstruction*. Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, Jan. 2001.

[ONU14] A. Olofsson, T. Nordström and Z. Ul-Abdin. “Kickstarting high-performance energy-efficient manycore architectures with Epiphany”. In: *Asilomar Conference on Signals, Systems and Computers*. IEEE. 2014, pp. 1719–1726.

[Ots79] N. Otsu. “A Threshold Selection Method from Gray-Level Histograms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66.

[Pag+02] D. Paganin et al. “Simultaneous Phase and Amplitude Extraction From a Single Defocused Image of a Homogeneous Object”. In: *Journal of Microscopy* 206.1 (2002), pp. 33–40.

[Pal+17] W. J. Palenstijn et al. “A distributed ASTRA toolbox”. In: *Advanced Structural and Chemical Imaging* 2.1 (2017), p. 19.

[Pan+18] R. J. Pandolfi et al. “*Xi-cam*: a versatile interface for data visualization and analysis”. In: *Journal of Synchrotron Radiation* 25.4 (July 2018), pp. 1261–1270.

[Par+19] M. Parada et al. “Two stage wicking of yarns at fiber scale investigated by synchrotron X-ray phase contrast fast tomography”. In: *Textile Research Journal* (2019). Accepted.

[Pat+15] B. M. Patterson et al. “In situ X-ray synchrotron tomographic imaging during the compression of hyper-elastic polymeric materials”. In: *Journal of Materials Science* 51.1 (2015), pp. 171–187.

[PB13] D. M. Pelt and K. J. Batenburg. “Fast tomographic reconstruction from limited data using artificial neural networks.” In: *IEEE Transactions on Image Processing* 22.12 (2013), pp. 5238–5251.

[PB14] D. M. Pelt and R. H. Bisseling. “A Medium-Grain Method for Fast 2D Bipartitioning of Sparse Matrices”. In: *IEEE 28th International Parallel and Distributed Processing Symposium*. 2014, pp. 529–539.

[PBS11] W. J. Palenstijn, K. J. Batenburg and J. Sijbers. “Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)”. In: *Journal of Structural Biology* 176.2 (2011), pp. 250–253.

[Pen+95] P. Penczek et al. “Double-tilt electron tomography”. In: *Ultramicroscopy* 60.3 (1995), pp. 393–410.

[PSV09] X. Pan, E. Y. Sidky and M. Vannier. “Why do commercial CT scanners still employ traditional, filtered back-projection for image reconstruction?” In: *Inverse Problems* 25.12 (2009), p. 123009.

[Rei+11] C. B. Reid et al. “The development of a pseudo-3D imaging system (tomosynthesis) for security screening of passenger baggage”. In: *Nuclear Instruments and Methods in Physics Research, Section A*. Vol. 652. 1. 2011, pp. 108–111.

[Ros+13] J. M. Rosen et al. “Iterative helical CT reconstruction in the cloud for ten dollars in five minutes”. In: *Fully Three-Dimensional - Image Reconstruction in Radiology and Nuclear Medicine* (2013), pp. 241–244.

[San+14] T. dos Santos Rolo et al. “In vivo X-ray cine-tomography for tracking morphological dynamics”. In: *Proceedings of the National Academy of Sciences* 111.11 (2014), pp. 3921–3926.

[SB93] K. Sauer and C. Bouman. “A Local Update Strategy for Iterative Reconstruction From Projections”. In: *IEEE Transactions on Signal Processing* 41.2 (1993), pp. 534–548.

[SH14] H. H. B. Sørensen and P. C. Hansen. “Multicore performance of block algebraic iterative reconstruction methods”. In: *SIAM Journal on Scientific Computing* 36.5 (2014), pp. C524–C546.

[She+14] A. Sheppard et al. “Techniques in helical scanning, dynamic imaging and image segmentation for improved quantitative analysis with X-ray micro-CT”. In: *Nuclear Instruments and Methods in Physics Research, Section B* 324 (2014), pp. 49–56.

[Sid+16] K. Siddique et al. “Apache Hama: An emerging bulk synchronous parallel computing framework for big data applications”. In: *IEEE Access* 4 (2016), pp. 8879–8887.

[SS92] H. Shi and J. Schaeffer. “Parallel Sorting by Regular Sampling”. In: *J. Parallel and Distributed Computing* 14.4 (1992), pp. 361–372.

[Sui] W. Suijlen. *BSPonMPI v0.3*. `https://sourceforge.net/projects/bsponmpi/`.

[Tho+15] W. M. Thompson et al. “High speed imaging of dynamic processes with a switched source X-ray CT system”. In: *Measurement Science and Technology* 26.5 (2015), p. 055401.

[Val11] L. G. Valiant. “A bridging model for multi-core computing”. In: *Journal of Computer and System Sciences* 77.1 (2011), pp. 154–166.

[Val90] L. G. Valiant. “A bridging model for parallel computation”. In: *Communications of the ACM* 33.8 (1990), pp. 103–111.

[VB05] B. Vastenhouw and R. H. Bisseling. “A two-dimensional data distribution method for parallel sparse matrix-vector multiplication”. In: *SIAM Review* 47.1 (2005), pp. 67–95.

[Vog+12] M. Vogelgesang et al. “UFO: A scalable GPU-based image processing framework for on-line monitoring”. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. IEEE. 2012, pp. 824–829.

[Wan+17] X. Wang et al. “Massively parallel 3D image reconstruction”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis SC ’17*. 2017, 3:1–3:12.

[War+16] J. Warnett et al. “Towards in-process X-ray CT for dimensional metrology”. In: *Measurement Science and Technology* 27.3 (2016), p. 035401.

[Wat94] D. W. Watt. “Column-Relaxed Algebraic Reconstruction Algorithm for Tomography With Noisy Data”. In: *Applied Optics* 33.20 (1994), p. 4420.

[Wil+09] S. Williams et al. “Optimization of sparse matrix-vector multiplication on emerging multicore platforms”. In: *Parallel Computing* 35.3 (2009), pp. 178–194.

[XM06] F. Xu and K. Mueller. “A comparative study of popular interpolation and integration methods for use in computed tomography”. In: *3rd IEEE International Symposium on Biomedical Imaging: Macro to Nano* (2006), pp. 1252–1255.

[Xu+10] W. Xu et al. “High-performance iterative electron tomography reconstruction with long-object compensation using graphics processing units (GPUs)”. In: *Journal of Structural Biology* 171.2 (2010), pp. 142–153.

[YB12] A. N. Yzelman and R. H. Bisseling. “An object-oriented bulk synchronous parallel library for multicore programming”. In: *Concurrency and Computation: Practice & Experience* 24.5 (2012), pp. 533–553.

[YR14] A. N. Yzelman and D. Roose. “High-level strategies for parallel shared-memory sparse matrix-vector multiplication”. In: *IEEE Transactions on Parallel and Distributed Systems* 25.1 (2014), pp. 116–125.

[Yze+14] A. N. Yzelman et al. “MulticoreBSP for C: A high-performance library for shared-memory parallel programming”. In: *Int. J. Parallel Programming* 42.4 (2014), pp. 619–642.

[Zen12] G. L. Zeng. “A filtered backprojection algorithm with characteristics of the iterative Landweber algorithm”. In: *Medical Physics* 39.2 (2012), pp. 603–607.

[Zha+17] G. Zhang et al. “X-ray Imaging of Transplanar Liquid Transport Mechanisms in Single Layer Textiles”. In: *Langmuir* 33.43 (2017), pp. 12072–12079.

[Zhe+14] Y. Zheng et al. “UPC++: A PGAS Extension for C++”. In: *Proc. IEEE IPDPS*. 2014, pp. 1105–1114.

List of publications

Publications that are part of this dissertation:

- Real-time quasi-3D tomographic reconstruction. *JW Buurlage, H Kohr, WJ Palenstijn, KJ Batenburg*. Measurement Science and Technology 29 (6), 2018
- Bulk: A Modern C++ Interface for Bulk-Synchronous Parallel Programs. *JW Buurlage, T Bannink, RH Bisseling*. European Conference on Parallel Processing, 519–532, 2018
- A geometric partitioning method for distributed tomographic reconstruction. *JW Buurlage, RH Bisseling, KJ Batenburg*. Parallel Computing 81, 104–121, 2019
- Real-time reconstruction and visualisation towards dynamic feedback control during time-resolved tomography experiments at TOMCAT. *JW Buurlage, F Marone, DM Pelt, WJ Palenstijn, M Stampanoni, KJ Batenburg, CM Schlepütz*. Scientific Reports 9 (1), 1–11, 2019
- A projection-based partitioning method for distributed tomographic reconstruction. *JW Buurlage, WJ Palenstijn, RH Bisseling, KJ Batenburg*. Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing, 58–68, 2020

Publications that are not part of this dissertation:

- Multigrid reconstruction in tomographic imaging. *D Marlevi, H Kohr, JW Buurlage, B Gao, KJ Batenburg, M Colarieti-Tosti*. IEEE Transactions on Radiation and Plasma Medical Sciences 4 (3), 300–310, 2019

- Real-time reconstruction of arbitrary slices for quantitative and in-situ three-dimensional characterization of nanoparticles. *H Vanrompay, JW Buurlage, DM Pelt, V Kumar, X Zhuo, LM Liz-Marzán, S Bals, KJ Batenburg*. Particle and Particle Systems Characterization (accepted). 2020

Samenvatting in het Nederlands

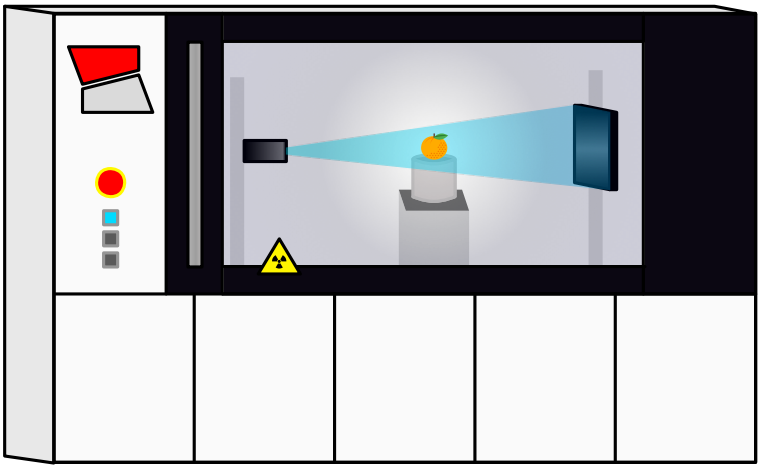
This chapter contains a lay summary of the research presented in this dissertation, and is written in Dutch.

Het in beeld brengen van het binnenste van een object zonder dit open te breken heeft een groot aantal toepassingen. Hier gebruik ik object in de breedste zin van het woord: denk bijvoorbeeld aan een patiënt in een CT scanner, een miniscuul nanodeeltje onder de microscoop, of een brug die geïnspecteerd moet worden op barsten in het beton.

Tomografische reconstructie is de wiskundige methode achter veel 3D beeldvormingstechnieken. Deze technieken werken allemaal op basis van het zelfde principe. Met behulp van straling worden tweedimensionale projectiebeelden van het driedimensionale object gemaakt. Deze projectiebeelden vormen in zekere zin de schaduw van een object onder een bepaalde hoek. Bij een 3D scan worden projectiebeelden gemaakt onder meerdere hoeken, en door deze informatie slim te combineren kan de interne driedimensionale structuur van het object achterhaald worden.

Afhankelijk van de gewenste resolutie van het 3D beeld, kan deze *reconstructiestap* een lange tijd duren. Dat deze algoritmes duur kunnen zijn komt voornamelijk door de grootte van de data. Deze is eenvoudig te demonstreren met een simpele berekening: één projectiebeeld bestaat op het moment typisch uit tot wel 4000 bij 4000 pixels, oftewel 16 megapixels. Van deze projectiebeelden wordt een 3D beeld gemaakt dat bestaat uit 4000 verschillende 16 megapixel plaatjes. Dit komt neer komt op 256 GB aan data voor een enkel 3D beeld.

Zelfs wanneer deze berekeningen op moderne computers worden uitgevoerd, kost het nog altijd minuten tot uren afhankelijk van de precieze methode die gebruikt voor de reconstructiestap. Dit beperkt de mogelijkheid om veranderingen in het binnenste van een object in beeld te brengen



Figuur 1: Een illustratie van het FleX-ray lab bij het CWI. De scanner is ongeveer 2 meter breed, diep, en hoog. Achter het raam wordt een experiment gedaan met een sinaasappel. De röntgenbron, links, bestraalt de sinaasappel, en de detector, rechts, neemt projectiebeelden op terwijl de sinaasappel rond wordt gedraaid.

terwijl ze gebeuren. Met andere woorden, het is tot nu toe niet mogelijk geweest om een actief videobeeld te creëren van het binnenste van een object.

Het doel van het onderzoek gepresenteerd in mijn proefschrift is om de reconstructietijd flink omlaag te brengen, zodat het mogelijk wordt om dynamische veranderingen in het object te volgen terwijl ze gebeuren. Dit is van belang om het experiment te kunnen bijsturen. Een materiaal kan bijvoorbeeld verder verhit worden, een patiënt kan gevraagd worden zijn adem in te houden, of een regio waar ogenschijnlijk iets onverwachts gebeurt kan worden uitvergroot. Figuur 1 toont een illustratie van het FleX-Ray lab van het CWI, waarmee de methoden uit mijn onderzoek getest zijn.

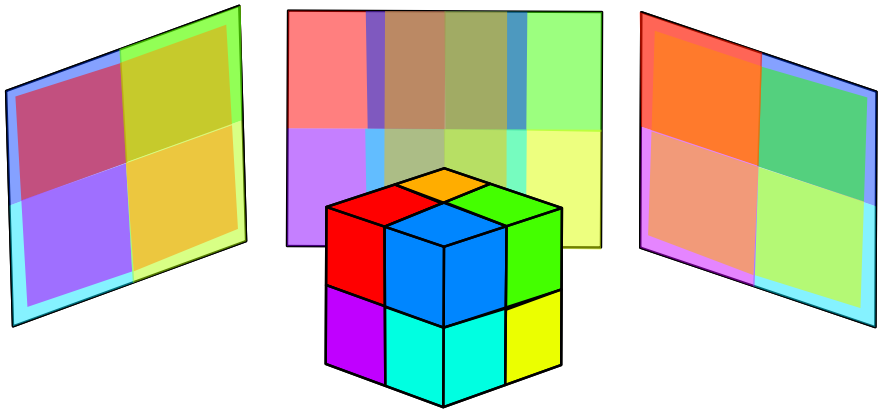
Parallele berekeningen op supercomputers

Een duidelijke trend is dat computers steeds meer parallel worden. Een computer bestaat uit verschillende soorten processoren zoals de conventionele CPU, de processor op een grafische kaart, en eventueel andere acceleratoren. Deze verschillende processoren bestaan zelf vaak uit meerdere cores. Cores kunnen min of meer onafhankelijk van elkaar berekeningen uitvoeren. Een CPU bijvoorbeeld, bestaat typisch uit een paar tot tientallen cores, en dit aantal stijgt sterk. Een moderne grafische kaart bestaat uit duizenden (simpelere) cores.

Deze ontwikkeling naar parallele systemen is niet alleen zichtbaar in consumentenhardware zoals desktops, laptops en mobiele telefoons, maar ook bij supercomputers die gebruikt worden voor grootschalige (wetenschappelijke) berekeningen. Bij een supercomputer kun je denken aan een cluster van computers, die verbonden zijn in de vorm van een netwerk. Elke computer zelf kan bestaan uit meerdere CPU's, en kan ook meerdere grafische kaarten hebben. Supercomputers zijn erg krachtig, maar het is niet altijd eenvoudig om de gezamenlijke rekenkracht van alle processoren te bundelen en samen in te zetten voor het oplossen van één en hetzelfde probleem. Om een algoritme dat bedoeld is voor een enkele processor geschikt te maken om uitgevoerd te worden door een supercomputer, moet deze geparalleliseerd worden.

Een voor de hand liggende manier om de reconstructie in tomografie te versnellen is om gebruik te maken van de gezamenlijke rekenkracht van, bijvoorbeeld, tientallen grafische kaarten. Dit blijkt echter precies een voorbeeld van een probleem dat zich niet gemakkelijk leent voor het parallel oplossen. Wanneer een naïëve aanpak wordt gebruikt voor de parallelisatie, moeten de grafische kaarten dusdanig veel met elkaar communiceren om gezamenlijk tot een oplossing te komen, dat het netto niet genoeg winst oplevert om het algoritme op een supercomputer uit te voeren. Net als wiskundigen zijn computers namelijk beter in nadenken dan communiceren.

Voor het parallel uitvoeren van tomografische reconstructie splitsen we het 3D volume op in evenveel delen als het aantal processorelementen dat we voor de berekening willen gebruiken. Ieder element is vervolgens verantwoordelijk voor het reconstrueren van één van de delen. Beeld je een röntgenstraal in die door een 3D object wordt gestuurd. Wanneer deze



Figuur 2: Het 3D volume is hier opgesplitst in 8 delen. Voor het begrip is een willekeurige kleur toegekend aan elk deel. Op de schermen achter het 3D volume, zien we in verschillende richtingen het schaduwspel van de opsplitsing. Waar schaduwen overlappen, zijn verschillende delen gekoppeld. Dit fenomeen wordt in dit proefschrift beschreven, en gebruikt om een goede opsplitsing te vinden.

straal door verschillende delen van het 3D object gaat, die elk toegewezen zijn aan verschillende elementen, worden elementen als het ware aan elkaar gekoppeld. Door deze koppeling moeten zij, tijdens de berekening, informatie uitwisselen over de tussenresultaten. Zie ook Figuur 2.

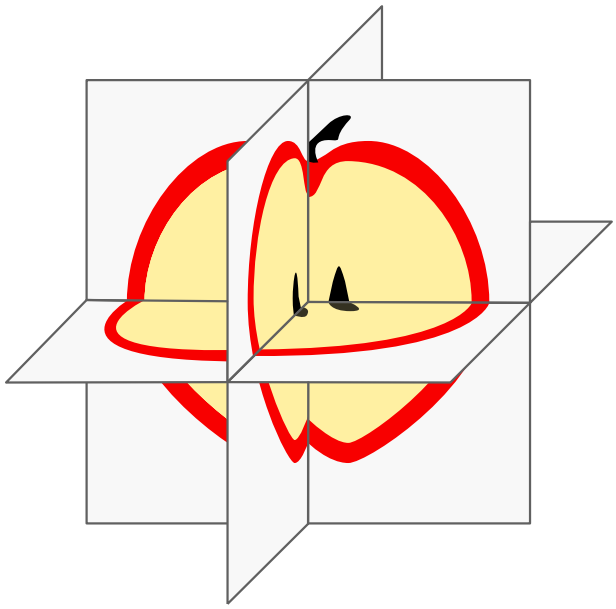
In Hoofdstuk 2 introduceer ik een softwarebibliotheek voor het implementeren van parallele algoritmes. In Hoofdstuk 3 en Hoofdstuk 4 presenteer ik mijn onderzoek naar het paralleliseren van reconstructie-algoritmes in tomografie. Dit onderzoek richt zich met name op het vinden van de beste opsplitsing van het 3D object, afhankelijk van de richtingen waarop stralen door het object worden gestuurd. Het resultaat is een opsplitsing die de koppeling, en dus communicatie, tussen elementen waar mogelijk vermijdt. Gebruikmakend van deze nieuwe methodes, is tot wel 10× minder communicatie nodig. Dit vertaalt zich naar snellere reconstructietijden.

Snelle weergave van doorsnedes

In het tweede deel van dit proefschrift bestuderen we een andere aanpak om de reconstructie te versnellen. Het idee achter deze aanpak is betrekkelijk eenvoudig. Vaak wordt de driedimensionale reconstructie weergegeven in de vorm van doorsnedes. Immers willen we in het binnenste van het object kijken. Normaal wordt eerst het 3D beeld gereconstrueerd, en vervolgens worden doorsnedes uit dit beeld berekend. Natuurlijk kan je niet naar alle doorsnedes tegelijk kijken, dus vaak wordt gekeken naar een selectie van bijvoorbeeld drie doorsnedes.

Wanneer we over een 3D videobeeld zouden beschikken van het binnenste van een object, zou dit in eerste instantie nog steeds via doorsnedes bestudeerd worden. Daarom draaien we in dit onderzoek de berekening om: we laten degenen die het experiment uitvoeren kiezen welke doorsnedes zij op elk moment willen bekijken en berekenen deze direct uit de projectiedata, in plaats van het volledige 3D volume uit te rekenen. De gekozen verzameling doorsnedes kan eenvoudig worden aangepast, en het resultaat is vervolgens direct zichtbaar. Hierdoor wekken we de illusie dat we een volledige 3D reconstructie maken, terwijl we eigenlijk alleen maar met doorsnedes en projectiebeelden hoeven te werken. Dit is een stuk goedkoper om uit te rekenen, maar is alleen mogelijk met een beperkt aantal reconstructiemethoden. Zie ook Figuur 3.

In Hoofdstuk 5 wordt deze nieuwe reconstructiemethode uitgelegd die we *quasi-3D* hebben genoemd. Hier laten we zien dat zelfs met een normale computer het minder dan een seconde kost om een hoge resolutie quasi-3D reconstructie te maken. Bij het kiezen van een andere doorsnede kunnen we zelfs in een tiende van een seconde dit nieuwe beeld laten zien. In Hoofdstuk 6 passen we deze methode toe bij een synchrotron. We demonstreren voor het eerst de mogelijkheid om een live 3D videobeeld te krijgen van het binnenste van een object bij een tomografisch experiment. In dit geval was het doel van het experiment om de wateropname te bestuderen van een synthetische stof. Door de beelden te bekijken kon bijvoorbeeld beslist worden om meer water toe te voegen aan het reservoir.



Figuur 3: Een illustratie van een quasi-3D reconstructie van een appel. We zien hier drie doorsnedes, die loodrecht op elkaar staan. Met de methode gepresenteerd in dit proefschrift kunnen deze doorsnedes eenvoudig gedraaid en verplaatst worden waarna de nieuwe doorsnede direct berekend wordt.

Curriculum Vitae

Jan-Willem Buurlage was born in 1991 in Heerenveen, The Netherlands. For his undergraduate degree at Utrecht University he was part of the TWIN program that combines the bachelor programs Physics and Astronomy, and Mathematics, and which he finished in 2013. He received his MSc degree (cum laude) in Mathematical Sciences from Utrecht University in 2016, with a specialization in scientific computing. His master's thesis, titled "Self-improving sparse matrix partitioning and bulk-synchronous pseudo-streaming", was written under the supervision of Rob Bisseling. It was awarded the Best Master's Thesis by the Graduate School of Natural Sciences. He started his PhD research at Leiden University under supervision of Joost Batenburg in 2016. The research was carried out at Centrum Wiskunde & Informatica (CWI) in Amsterdam. As part of his PhD research, he was a visiting student at the DTU in Denmark, and made research visits to PSI in Switzerland, the TU in Berlin, and EMAT in Antwerp. He attended international conferences, workshops, and summer schools in Aussois, Bologna, Grenoble, Muenster, Turin, Tokyo, and Seattle.

Acknowledgments

First and foremost, I want to thank my advisors for all the time and energy they have put into our research projects. Rob, thank you for introducing me to the the world of parallel algorithms. I would not have started as a PhD student and this dissertation would not have been written without your incredible encouragement and support. Joost, thank you for your guidance and trust. And, in particular, I would like to thank you for giving me the freedom to explore my interests and to spend time abroad throughout my PhD project.

A special thanks to my coauthors that made this research possible, both those at CWI, as well as my external collaborators at EMAT, KTH, and PSI.

At CWI, I was lucky enough to share an office with Rien, Zhichao, and later Allard, for the better half of my PhD. In addition to being great colleagues, you have also become valuable friends. Holger, Daan, Willem Jan thank you for your role in the research presented in this dissertation. I would also like to thank the other members of the computational imaging group at CWI who all contributed to making it a great research environment: Nicola, Sophia, Folkert, Maureen, Dzemila, Georgios, Alex, Francien, Jordi, Richard, Mathé, Vlad, Poulami, Adriaan, Giulia, Felix, Robert, and Tristan. I would also like to thank the support staff, and especially Nada and Duda, for all their effort. Additionally, there are many others at CWI that I met along the way and that I want to thank for making it such a great institute: Isabella who convinced me to try skiing, Prashant who helped me improve at squash, and everyone else that joined me for a game of table tennis, or that sat across from me over a chess board.

I was also lucky enough to meet many fellow PhD students during my trips abroad. Anders, Björn, and Nikolai: thanks for all the dinners, talks, and board games during my two month stay in Copenhagen. Anna, Giulia, Nurbek, Shashank and Thibault, thank you for making the summer school

in Aussois one of the highlights of my PhD, and for keeping me company during my stays in the US.

My friends were instrumental in helping me stay sane during the inevitable ups and downs of a PhD project. I would like to give a heartfelt thanks to all of them. Tom, for all the programming projects, games, and talks we shared. Thijs, who is always there when I need a healthy outside perspective. Abe, for never failing to trigger my competitive side. Erik, for having been a fantastic roommate all these years. Gijs, for transmitting your passion for cycling onto me (and thus giving my disposable income a new destination). And everyone else who played a role during my PhD, including Eveline, Daphne, Paul, Erik B, Peter, Anouk, Bodo, and Roeland.

Finally, I would like to thank my family who have been there for me throughout all my studies. My brother, for the cycling trips and video games that helped me unwind. My sister, for reminding me that there is more to life than algorithms and software. And my parents, for their unrelenting love and support.