Universiteit
Leiden
The Netherlands

Real-time tomographic reconstruction

Buurlage, J.

Citation
Buurlage, J. (2020, July 1). *Real-time tomographic reconstruction*. Retrieved from https://hdl.handle.net/1887/123182

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/123182

Note: To cite this publication please use the final published version (if applicable).

Cover Page

Universiteit Leiden

The handle http://hdl.handle.net/1887/123182 holds various files of this Leiden University dissertation.

Author: Buurlage, J.
Title: Real-time tomographic reconstruction
Issue Date: 2020-07-01

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.

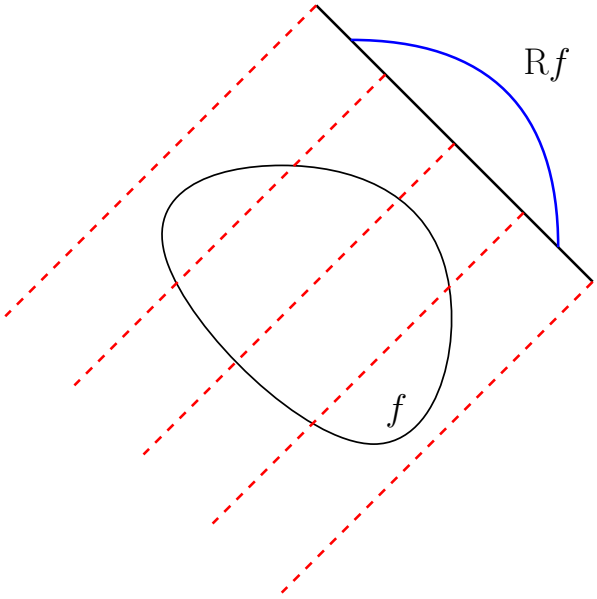


Figure 1.1: The Radon transform of a function are its line integrals. Here, we visualize 5 lines passing through the domain of a function f. The blue line represents the line integral values along this specific angle, assuming the function f is constant. This corresponds to a projection image in a tomographic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

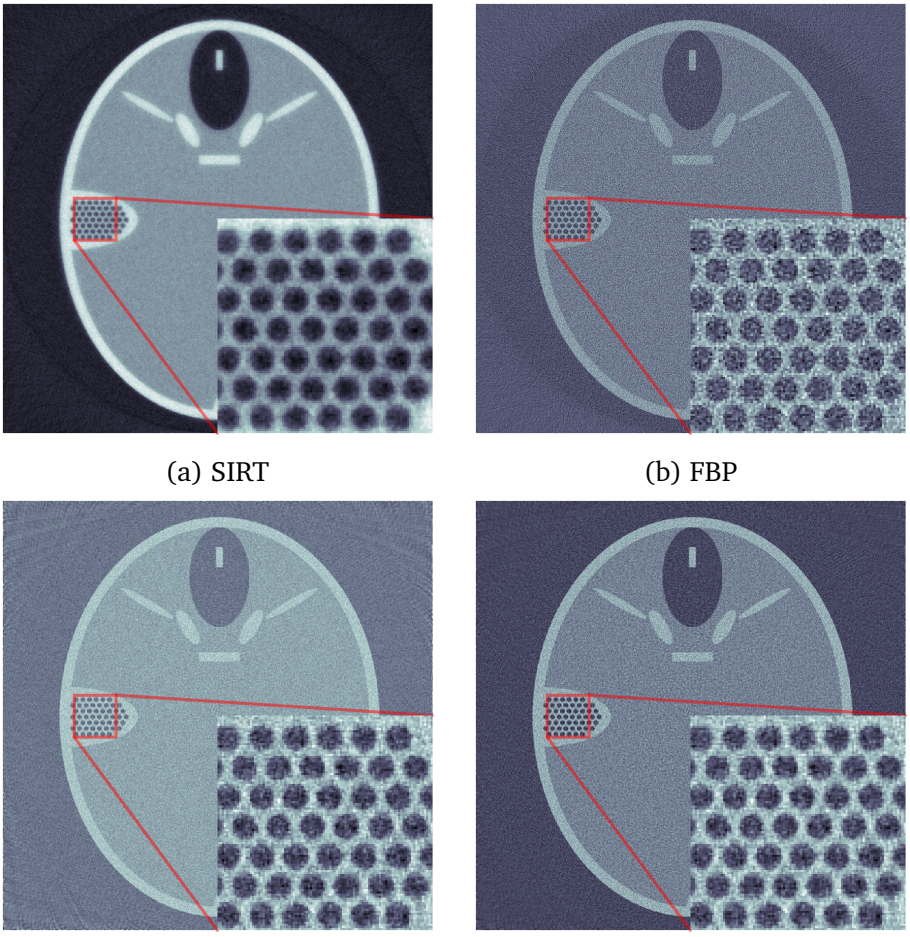
An important distinction between iterative and direct reconstruction methods, is that direct methods are *local*: each volume element can be reconstructed independently and efficiently from the (filtered) projection data.

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.

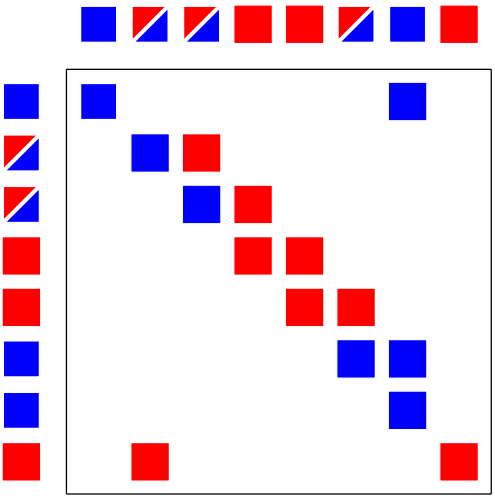


Figure 1.3: Sparse matrix partitioning. Each nonzero element of the 8×8 sparse matrix is indicated by a colored square. The elements are partitioned in two: a red part and a blue part. On the left and top of the matrix, the colors that are present in each row and column are indicated. The communication 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.