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

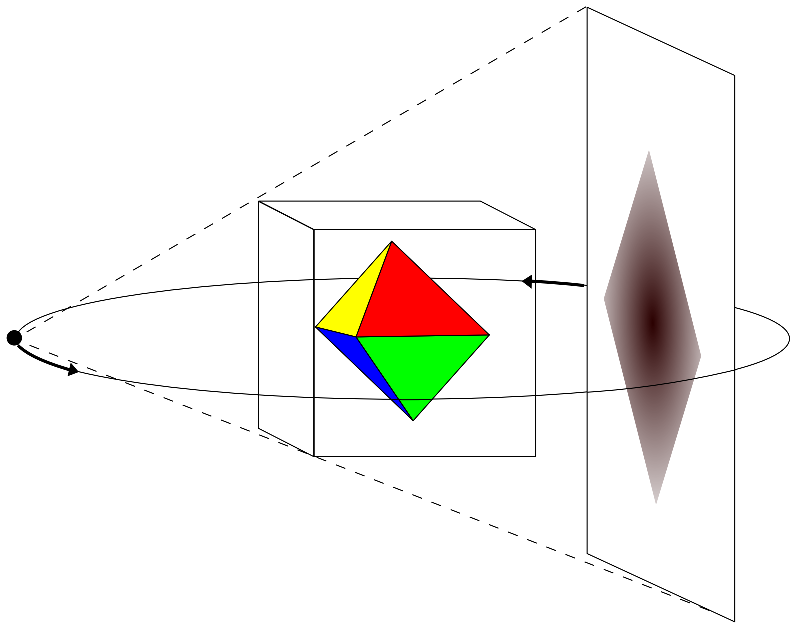


Figure 3.1: Schematic overview of a 3D tomography setup. Here, we show a single projection. On the left, we have a point source marked by a disk, which is emitting penetrating radiation. A cone-shaped collection of rays penetrates a cubic region of space shown in the center. As an example, we let it 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 $\mathscr{V}$ 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 $\ell_i \in \mathscr{G}$ 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.

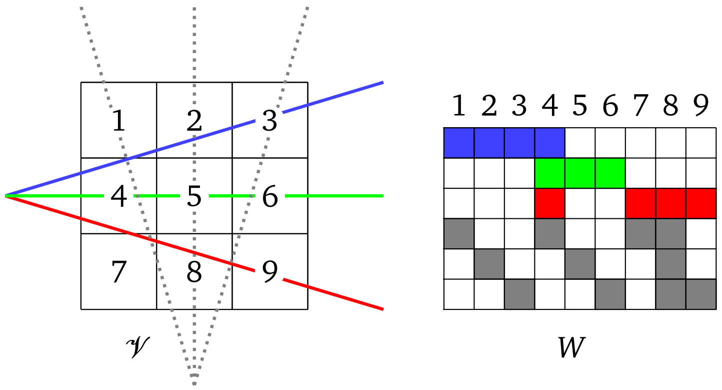


Figure 3.2: Construction of a tomography matrix in two dimensions. On the left, the object volume is shown together with two sets of three lines, corresponding to two projection images. One of these sets is shown in red, green and blue. The other projection is shown as dotted gray lines. The corresponding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

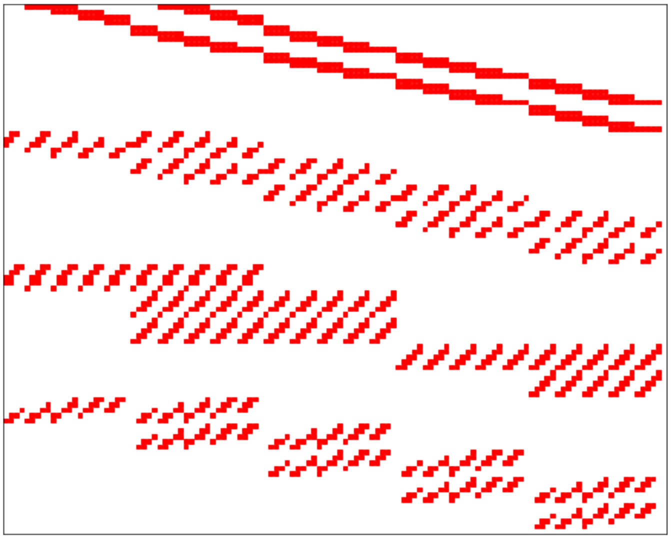


Figure 3.3: The nonzero pattern of the matrix W for a very small tomographic reconstruction problem. We consider a discretized object volume of $5 \times 5 \times 5$ voxels, with a detector shape of 5×5 pixels. The matrix was generated using a slice-interpolated DIM and a standard parallel geometry with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

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

We identify multiple options. First, we can partition the object volume $\mathscr{V}$. Each processor is then assigned a *subvolume* $\mathscr{V}^{(s)}$, 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 $\mathscr{G}$, i.e., assign a collection of lines to each processor. In this case, each processor is assigned a *subgeometry* $\mathscr{G}^{(s)}$, 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 **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 $\mathscr{G}|_{\mathscr{V}^{(s)}}$ 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: $\mathbf{x}^{(s)}$, $W^{(s)}$, $\phi$.
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 $\mathscr{V}^{(s)}$ 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:

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

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 $\mathscr{G}$ be an acquisition geometry, $\mathscr{V}$ 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:*
>
> $$\begin{aligned} \text{minimize}_{\pi \in \Pi} \quad & V(\pi) \\ \text{subject to} \quad & \epsilon(\pi) < \epsilon_{\text{max}}. \end{aligned}$$

Since an acquisition geometry $\mathscr{G}$ 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 $\mathscr{V}$. 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.

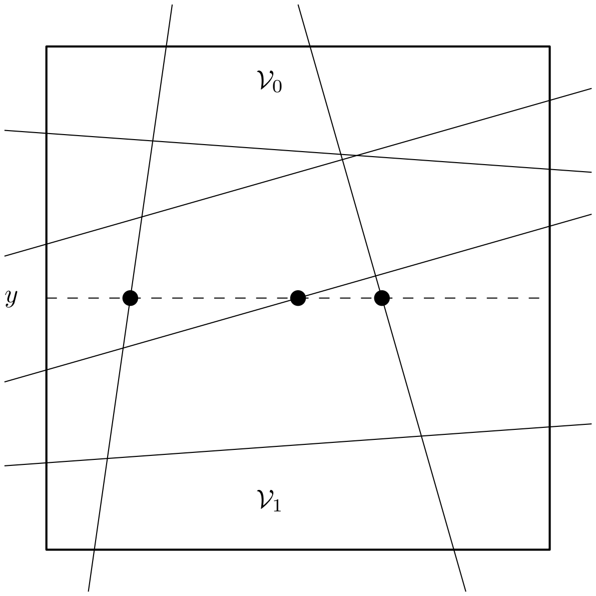


Figure 3.4: A set of lines through a square two-dimensional object volume $\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1$. The lines intersecting both subvolumes are exactly those lines that cross the horizontal interface at height y, shown here with a dashed line, between $\mathcal{V}_0$ and $\mathcal{V}_1$. In this case, three of the six lines have an intersection point (shown as •) 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($\mathcal{V}_1$, $\mathcal{G}|_{\mathcal{V}_1}$, $q-1$, $\epsilon_1$)
$n_2 \leftarrow$ PARTITION($\mathcal{V}_2$, $\mathcal{G}|_{\mathcal{V}_2}$, $q-1$, $\epsilon_2$)
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 $\mathcal{W}$. 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 $d$th coordinate
  $V \leftarrow 0$
  for $\mathbf{x}$ in $E$ do
    if event $\mathbf{x}$ is incoming then
      $V \leftarrow V + 1$
    else if event $\mathbf{x}$ 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.

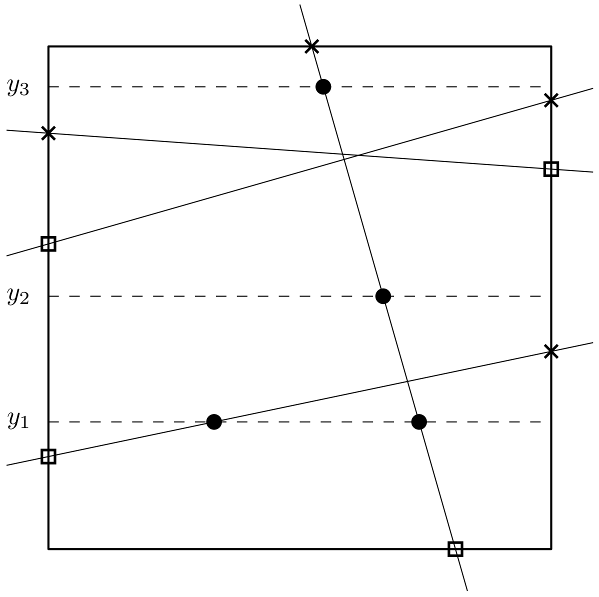


Figure 3.5: Visualization of the 2D equivalent of the 3D plane sweep described in algorithm 4. Imagine that we are considering a horizontal candidate interface which we are moving upwards, i.e., we gradually increase the y coordinate of the interface. If we were to split the volume according to the current 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

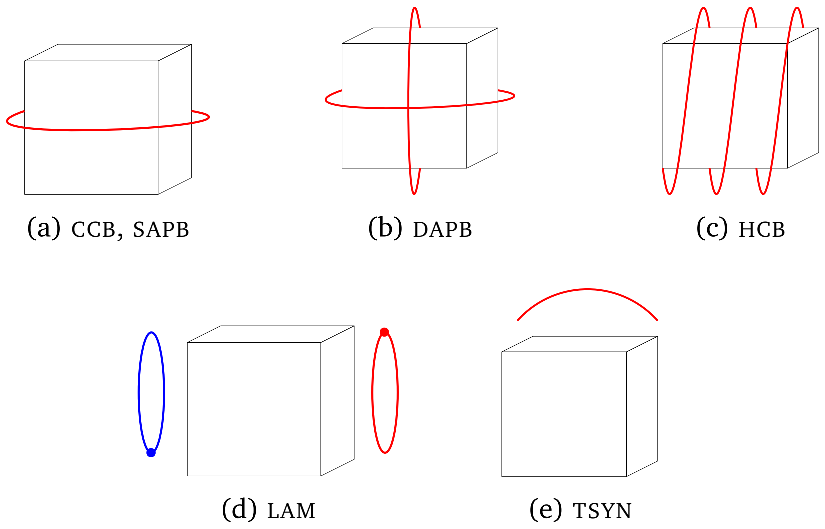


Figure 3.6: Schematic overview of the acquisition geometries that we consider. Here, the source trajectory is shown with a fat red line. The center of the detector is assumed to be at the antipodal point, except in (d) where the detector position is shown in blue. In (a) and (b), we indicate both parallel-beam 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/`

$\mathcal{G}$		$p = 16$ ($\times 10^5$)	$p = 32$ ($\times 10^6$)	$p = 64$ ($\times 10^7$)	$p = 128$ ($\times 10^8$)	$p = 256$ ($\times 10^9$)
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.

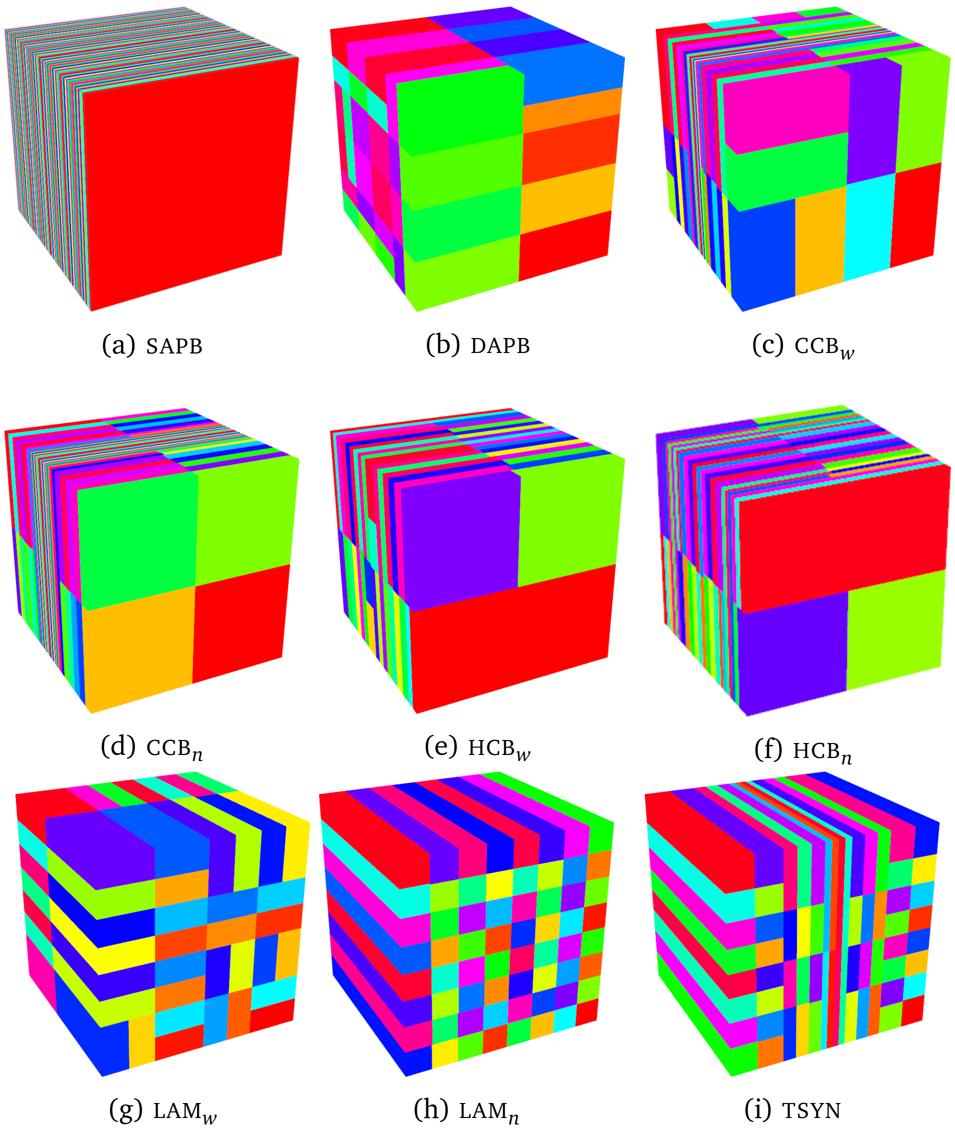
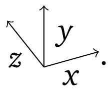
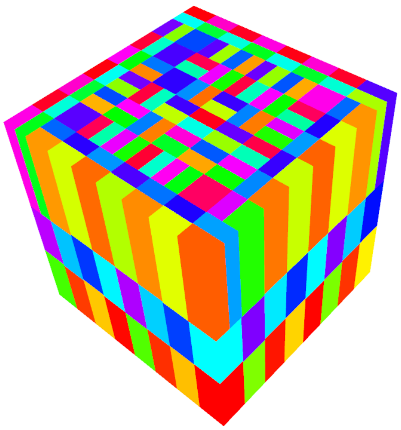


Figure 3.7: Resulting GRCB partitionings for $p = 64$ processors. The axes are as in . If there is a main rotation axis, it corresponds to z. For TSYN, the stationary detector is placed perpendicular to the z-axis.

	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$_w$

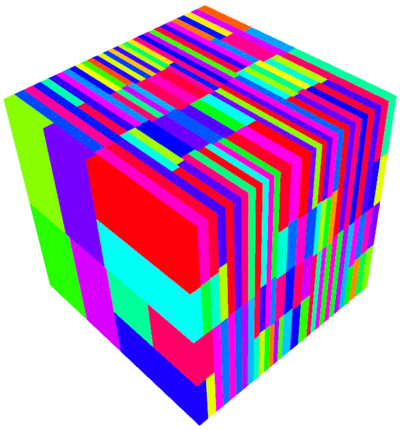

(b) HCB$_w$

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-

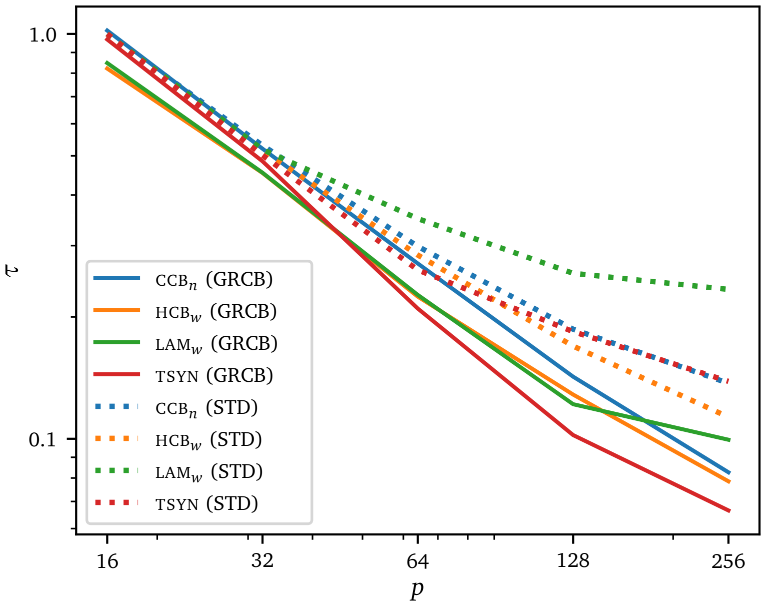


Figure 3.9: The runtime of one SIRT iteration plotted against the number of processors. Vertically, the relative runtime τ is shown on a logarithmic scale, defined for each geometry as the time compared to the runtime of reconstructing using a standard partitioning with $p = 16$ processors. The reconstruction 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

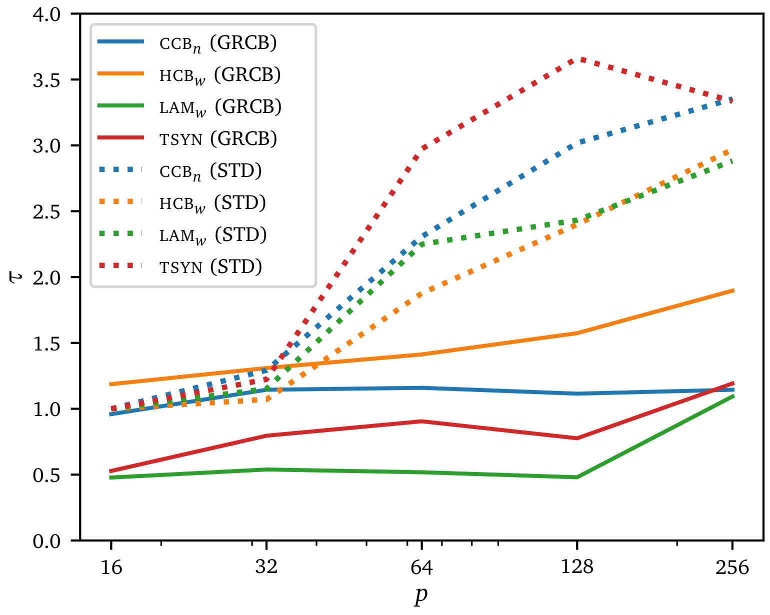


Figure 3.10: The communication time of one SIRT iteration plotted against the number of processors. Vertically, the relative communication time τ is shown, defined for each geometry as the time compared to the communication time for a standard partitioning with $p = 16$ processors. The communication 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 $\mu_{\max}$.

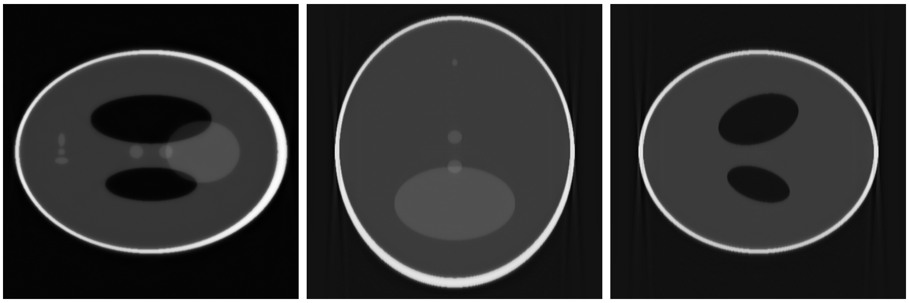


Figure 3.11: Reconstructed slices for an object volume of $512 \times 512 \times 512$ voxels with the CCB$_n$ acquisition geometry using 64 processors. For the reconstruction, 100 iterations of SIRT were applied with a slice-interpolated DIM. Here we used a modified 3D Shepp–Logan phantom. The left, 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 computations 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

$\mathscr{G}$	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.