A distributed SIRT implementation for the ASTRA Toolbox

Willem Jan Palenstijn*†, Jeroen Bédorf* and K. Joost Batenburg*†‡
*CWI, Amsterdam, Netherlands
†iMinds-Vision Lab, University of Antwerp, Belgium
‡Universiteit Leiden, Netherlands
Email: willem.jan.palenstijn@cwi.nl, jeroen.bedorf@cwi.nl and joost.batenburg@cwi.nl

***Abstract*—The ASTRA Toolbox is a software toolbox that enables rapid development of GPU accelerated tomography algorithms. It contains GPU implementations of forward and backprojection operations for common scanning geometries, as well as a set of algorithms for iterative reconstruction. These algorithms are currently limited to using a single GPU.**

A drawback of iterative reconstruction algorithms is that they are slow compared to classical backprojection algorithms. As a result, using only a single GPU can result in prohibitively long reconstruction times when working with large data volumes.

In this paper, we present an extension of the ASTRA Toolbox with implementations of forward projection, backprojection and the SIRT algorithm that can be distributed over multiple GPUs and multiple workstations to make processing larger data sets with ASTRA feasible.

I. Introduction

Iterative tomographic reconstruction methods have become increasingly popular in recent years. Compared to classical backprojection methods, they provide more flexibility in dealing with various scanning geometries and image priors. As a drawback, iterative methods are computationally demanding, resulting in long reconstruction times. Advances in compute hardware, and in particular the rise of *Graphics Processing Unit* (GPU) implementations, have resulted in substantial reductions in the required computation time. However, with ever growing experimental data sets, speed remains a major problem.

The ASTRA Toolbox [1]–[3] is a software toolbox for the development of tomographic reconstruction algorithms. It provides a set of GPU-accelerated flexible building blocks, which are currently limited to using a single GPU. For large volume sizes, even an optimized single-GPU implementation can still result in prohibitively long reconstruction times, making it necessary to split the computation across multiple GPUs, possibly also using multiple physical compute nodes.

Two common 3D tomography geometries are parallel beam, used for example in synchrotrons and electron microscopy, and circular cone beam, used for example in micro-CT. Large 3D parallel beam data sets can easily be split into a stack of 2D slices, which makes it easy to distribute the reconstruction over multiple workstations and multiple GPUs. For cone beam geometries, most rays intersect multiple volume slices, so such a distribution into independent slices is no longer a possibility for iterative algorithms [4]–[6].

In this paper we describe an implementation of the SIRT algorithm for a cone-beam geometry within the ASTRA Toolbox that is distributed over multiple workstations and GPUs.

The SIRT algorithm [7], like most current iterative methods, models the tomography reconstruction problem as a linear system $p = Wx + \varepsilon$, where p is the measured projection data, W is the system matrix, x the unknown 3D volume, and ε the measurement noise. SIRT then iteratively minimizes the following (weighted) least squares problem.

$$\min_x ||Wx - p||_W$$

SIRT starts from an initial solution x_0, often with $x_0 = 0$. We write R for the diagonal matrix with the inverse row sums of W on the diagonal, and C for the diagonal matrix with the inverse column sums of W on the diagonal. The update iteration of SIRT can then be written as follows.

$$x_{i+1} = x_i + C \cdot W^t \cdot R \cdot (p - Wx_i)$$

Multiplication by the system matrix W is often called *forward projection* (FP), and computes the projections for a given volume. Multiplication by its transpose W^t is commonly called *backprojection* (BP).

For all but the smallest data sets, the FP and BP operations are by far the most time-consuming parts of the SIRT algorithm, and are therefore the operations that benefit most from parallelisation. The paper is structured as follows. In Section II, we present our implementation and discuss its design choices. Experimental results on the performance of our implementation are provided in Section III. Section IV concludes the paper.

II. Methods

A principal difficulty when developing a GPU implementation of the SIRT algorithm for large tomography datasets, is that the projection and volume data required by SIRT may not fit into the available GPU memory. A basic approach is to keep a copy of all projection and volume data in system memory, and use the GPU for accelerating the FP and BP operations, which are by far the most expensive parts of SIRT. It is possible to divide both the volume data and projection data into multiple independent blocks. By processing these blocks

on a GPU one by one we can perform GPU-accelerated FP and BP operations that do not fit on a single GPU.

This method extends naturally to the use of multiple GPUs. Instead of sending all blocks to the same GPU, we can distribute them over the available GPUs. This allows for the reconstruction of larger volumes as well as improved reconstruction performance as more compute resources are available to execute a single reconstruction. This method relies on the GPUs being installed in the same physical system and having access to the full data in host memory. However, if the GPUs are distributed over multiple physical systems, also called nodes, a more sophisticated data distribution method is required.

A. Domain decomposition

To go beyond the use of a single node we have to distribute the data. For this distribution we make a distinction between the volume and detector data sets. For efficiency reasons, we assume that we have a circular cone beam geometry, rotating around the z-axis. First of all, we split the volume into N_{node} independent sub-volume blocks, where each node is assigned a different set of slices orthogonal to the z-axis to reconstruct (we use blocks and slices interchangeably). Next we compute for each block the projection extent on the detector; this is the region of the projection data that is affected by a FP of the sub-volume, and that affects a BP to the sub-volume. Using this information we know which subset of detector data is required by which sub-volume. This way memory usage is reduced, as each node only has to store the required data. After the block configuration and projection extents have been determined, the nodes can obtain the required input data, either by receiving it from a master node, or by retrieving it from a central repository. Depending on the geometry and projection method, it is likely that the assigned detector regions of different nodes overlap with each other. If detector regions overlap, this means that multiple nodes read from the same detector location (during BP) or write to the same location (during FP). Therefore we determine the exact overlap between the detector regions of the different nodes. This is illustrated for the case of a single projection direction in Fig. 1.

For the communication between the nodes we use the Message Passing Interface (MPI), the next set of subsections describe in more detail which communication is required when. The flowchart for the resulting distributed SIRT algorithm is shown in Fig. 2.

B. Forward projection

Computing the result of an FP operation onto the overlapping regions on the detector requires volume data from multiple nodes. Since FP is a linear operation, we can perform the FP operation for each node separately, and afterwards sum the results in the overlapping detector regions by exchanging data between nodes. This is achieved using the overlap configuration as computed during the domain decomposition. If required, each node copies the overlapping slices from the GPU to the host. Next, these slices are exchanged with

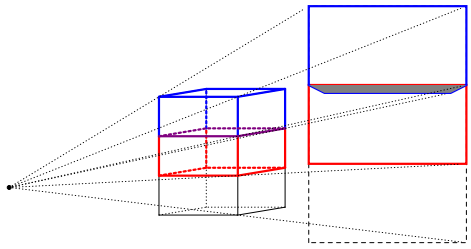

Fig. 1. The gray area shows the projection overlap between two adjacent volume blocks.

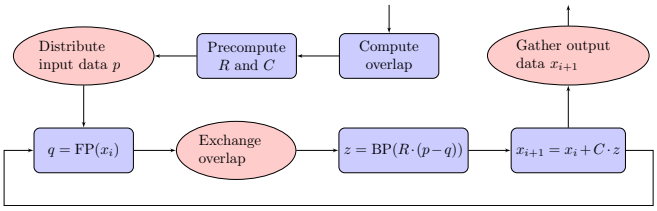

Fig. 2. Distributed SIRT. Computation steps are rectangular, and communication steps are ellipses.

the neighbouring processes. Finally, the overlapping detector regions are combined. The result is sent back to the GPU. By exactly computing the domain extents we minimize the amount of data that has to be exchanged, while ensuring that afterwards, each node has a consistent and correct copy of its detector region.

C. Backprojection

Since each node locally stores the part of the detector data needed to perform a BP operation, all the following operations can be performed locally and independently on each node: (i) computing the residual of the forward projection with respect to the measured projections; (ii) scaling this with inverse row weights R; (iii) performing the BP operation; (iv) scaling with inverse column weights C; and (v) updating the reconstruction volume. Therefore, no further communication is required during an iteration of the SIRT algorithm.

D. Stand-alone FP and BP

The FP and BP operations described above have been presented as sub-operations of the SIRT algorithm, but they can also directly be used as stand-alone operations.

For ease of implementation, we have made no changes to the FP and BP operations when they are used stand-alone, even though the domain decomposition and communication choices are not necessarily optimal for this situation.

III. Experiments and Results

In this section we test the scaling of three different methods, a single FP (including the required communication), a single BP and a SIRT reconstruction using 10 iterations. Each SIRT iteration contains an FP (including communication), a BP, and auxiliary functions required for the reconstruction algorithm. For SIRT we present the average time of a single iteration. With these methods we performed two different experiments.

In the first experiment we tested the multi-GPU scaling on a fixed sized volume using 1 to 16 GPUs. In the second experiment we scale the volume size from 256 to 2048 and measured the time that each method takes using 1, 4, 8 and 16 GPUs.

For all computational experiments, we used a cubic reconstruction volume of size N^3, and a square detector of size N^2, where the number of angles is fixed at 496. We used a cone angle of approximately 7.8 degrees.

The hardware used for the experiments consists of 4 machines, connected using QDR (40Gbit) InfiniBand. Each machine has 64GB RAM, two Xeon E5-2650 CPUs and 4 Tesla K20m NVIDIA GPUs. This allows us to scale from 1 tot 16 GPUs. We always fill a single node before we add a second node. For example, with 4 GPUs a single machine is used and with 5 GPUs two machines are used with 4 processes on the first and 1 process on the second node. We used CUDA 5.5 and CentOS 5.11.

For practical volume sizes and GPU counts, the overlapping regions form only a small fraction of the total detector size and the exchange of these regions forms only a fraction of the time required to compute the actual projection of the volume slices. To give an indication of the exchange sizes we present the detector and overlap sizes for an $N = 1024$ volume in Table I. As we can see, if we use a small number of GPUs the average overlap is less than 18% of the total detector size. However, if we split the volume over 8 or more GPUs then the overlap size becomes significant, over 70%, which will have a substantial impact on the execution time for smaller volumes. This will also become clear in the results below.

The results of the first experiment are presented in Fig. 3, for the case $N = 1024$. On the horizontal axis we indicate the number of GPUs and on the vertical axis the time it takes to complete one BP (solid line), FP (dashed line) or one SIRT (dotted line) operation. The BP scales nearly linearly from 1 to 16 GPUs as there is no communication required and the sub-volumes are large enough to saturate the GPU. For the FP the scaling is affected by network communication. We can see that the scaling is less ideal than that of the BP. But eventhough the network communication negatively impacts the scaling the execution time keeps decreasing when more GPUs are added. The SIRT iteration, which consists of both an FP, BP, network communication and host operations also benefits from using more GPUs and continues to scale. But as with the FP operation we see the influence of network communication, but here the effect of adding GPUs becomes neglible when using 10 GPUs. With 11 or more GPUs we hardly see any improvement in the execution time as we are dominated by the communication time. The more GPUs we have the smaller the blocks per GPU and the lower the computation time, but the number of slices that overlap will form a larger fraction of the total block size on a GPU. So with more GPUs we have to exchange relatively more data with more neighbours while the GPU has less data to process.

In Fig. 4 we present the results of the second experiment. Each of the three panels shows a different operation; BP in the top, FP in the middle and SIRT in the bottom panel[1]. For each we present the execution time for $N = 256$ up to $N = 2048$ using the 4 different GPU configurations. The increase of the execution time for the operations are as expected where doubling N results in an $\sim 8\times$ increase in execution time. Since the BP is not influenced by communication we only see benefit of using more than 1 GPU, with a near perfect speed-up for the largest volume sizes. For FP the story is slightly different: the timings for 8 and 16 GPUs are higher than those of 1 and 4 GPUs when $N < 1024$, because of the communication overhead. For $N \gtrsim 1024$ the communication forms a smaller fraction of the total execution time and it becomes more efficient to use larger GPU counts. For SIRT we see a mix of the BP and FP results, as expected as SIRT is built on top of those operations, with lower efficiency when using 16 GPUs on small volume sizes and better scaling on larger volume sizes.

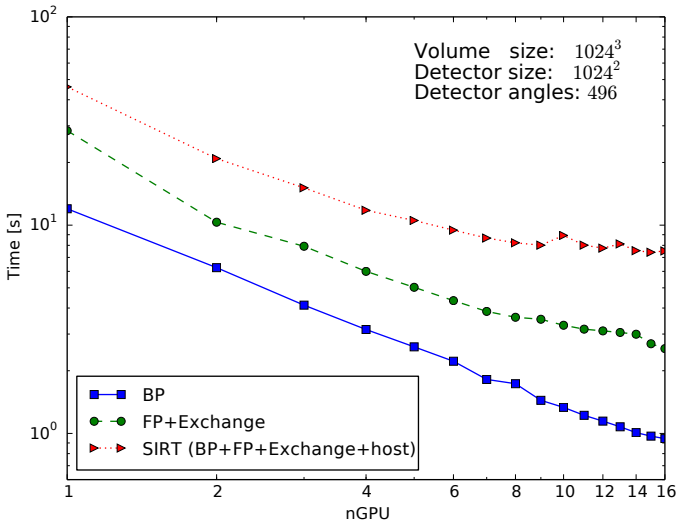

Fig. 3. Performance scaling of the BP, FP and SIRT routines over 16 GPUs. Presented is the time required, in seconds, to execute a single BP (solid line, square), single FP (dashed line, circle) and single SIRT iteration (dotted line, triangle). We start with a single GPU where, because of GPU memory limits, we keep part of the data in host memory. Next we increase this, in steps of 1, to 16 GPUs whereby we always fill a node before adding more.

IV. Discussion and Conclusion

The experiments of Section III indicate that the implemented parallel distribution method scales well for practical volume sizes and GPU counts. The larger the volume, the more GPUs can be used before communication overhead prevents a speedup from adding additional GPUs.

However, there is still room for improvement. In particular, better scaling might be achieved when performing the exchange of the overlap regions in parallel with computation, rather than sequentially.

[1]To get SIRT data for the 1536^3 and 2048^3 volumes when using 4 GPUs, we used 1 GPU per node. For the 2048^3 volume on 8 GPUs we used 2 GPUs per node. A single node does not have enough memory to hold all the buffers required for a SIRT reconstruction with $N \gtrsim 1536^3$.

TABLE I

EXAMPLE OF THE SIZES OF THE OVERLAPPING DETECTOR REGIONS FOR A 1024^3 VOLUME AND DIFFERENT GPU COUNTS. THE SECOND COLUMN SHOWS THE SIZE OF THE SUB-VOLUME BLOCK ASSIGNED TO EACH OF THE GPUS. THE NEXT SET OF COLUMNS INDICATES THE DISTRIBUTION OF THE CORRESPONDING PROJECTION REGION SIZE. THE FOLLOWING THREE COLUMNS INDICATE THE SIZE OF THE OVERLAPPING REGION. THE FINAL COLUMN INDICATES WHICH PERCENTAGE OF THE TOTAL DETECTOR SPACE THE OVERLAPPING REGION OCCUPIES.

#GPU	Volume sub-block size	Detector size ($\times 1024$)			#Overlapping slices			Overlap versus total size
	$\times 1024^2$	min	max	avg	min	max	avg	%
1	1024	1024	1024	1024	0	0	0	0
2	512	513	513	513	2	2	2	0.4
4	256	278	283	281	48	50	49	17.4
8	128	143	189	165	28	70	120	72.7
16	64	72	143	107	15	158	90	84.1

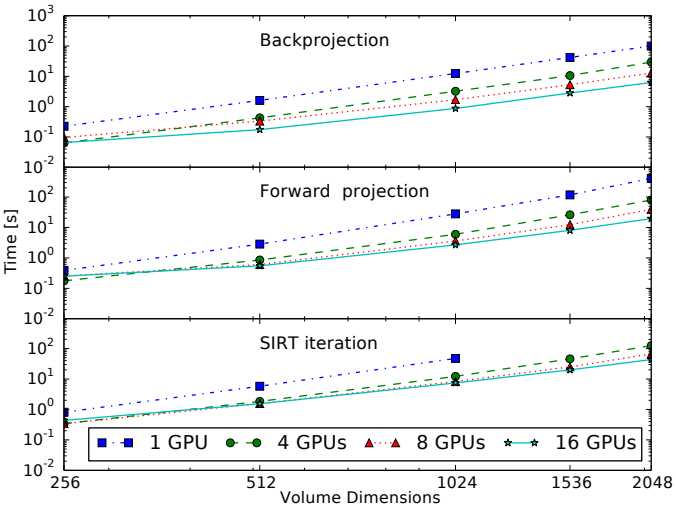

Fig. 4. Performance scaling of the BP, FP and SIRT routines over a range of volume sizes. Presented is the time required, in seconds, to execute a single BP (top panel), single FP (bottom panel) and single SIRT iteration (middle panel). We increase the volume size from 256^3 to 2048^3. The detector size increases from 256^2 to 2048^2 with 496 projection angles.

The proposed strategy should also generalize to different iterative methods than SIRT, and we plan to explore this in the future. Algorithms that include other operators — for example, blurring kernels or Total Variation — may then also require exchanging neighbouring slices of volume data. In that case, it may be beneficial to create a domain decomposition that only requires the exchange of volume data instead of only projection data. This change is straightforward and does not require significant changes in our methods; only the communication step is performed at a different moment on a different set of data.

Separate from the improvements related to multi-GPU support, there is room for further optimization inside the existing ASTRA GPU FP and BP operations that we have used, especially on current GPUs. For example, [8] and [9] give an overview of recent improvements to GPU implementations of the BP method. Most improvements there will automatically apply to the present work.

We conclude that this work extends the functionality of the ASTRA Tomography Toolbox by allowing efficient reconstructions of volumes that do not fit in the memory of a single GPU, on either a single node or using multiple nodes of a GPU cluster. We have shown that the method scales to a volume size of $N = 2048$ using 16 GPUs. This works for low level C++ implementations as well as for the high-level Matlab and Python interface.

Similar to the current operations implemented in the ASTRA Toolbox, our work will enable the rapid design and implementation of advanced reconstruction algorithms, using the distributed FP, BP, and SIRT implementations as building blocks.

ACKNOWLEDGMENTS

This work was supported by the Netherlands Organisation for Scientific Research NWO (grants #612.071.305 and #639.072.005), the Flanders digital research center iMinds (project MetroCT) and IWT. Networking support was provided by the EXTREMA COST Action MP1207.

REFERENCES

[1] W. J. Palenstijn, K. J. Batenburg, and J. Sijbers, "Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)," vol. 176, no. 2, pp. 250–253, 2011.

[2] ——, "The ASTRA tomography toolbox," in *13th International Conference on Computational and Mathematical Methods in Science and Engineering*, 2013.

[3] https://sf.net/projects/astra-toolbox.

[4] T. M. Benson and J. Gregor, "Framework for iterative cone-beam micro-CT reconstruction," *IEEE Transactions on Nuclear Science*, vol. 52, no. 5, pp. 1335–1340, 2005.

[5] J. Gregor, "Distributed CPU multi-core implementation of SIRT with vectorized matrix kernel for micro-CT," *Proceedings of the 11th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, 2011.

[6] J. M. Rosen, J. Wu, J. A. Fessler, and T. F. Wenisch, "Iterative helical CT reconstruction in the cloud for ten dollars in five minutes," *Proceedings of the 12th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, 2013.

[7] J. Gregor and T. M. Benson, "Computational analysis and improvement of SIRT," *IEEE Transactions on Medical Imaging*, vol. 27, no. 7, pp. 918–924, 2008.

[8] T. Zinsser and B. Keck, "Systematic performance optimization of cone-beam back-projection on the Kepler architecture," *Proceedings of the 12th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 225–228, 2013.

[9] E. Papenhausen and K. Mueller, "Rapid rabbit: Highly optimized GPU accelerated cone-beam CT reconstruction," in *Nuclear Science Symposium and Medical Imaging Conference, 2013 IEEE*, Oct 2013, pp. 1–2.