Journal of Real-Time Image Processing (2021) 18:1979–1991
https://doi.org/10.1007/s11554-020-01069-5

ORIGINAL RESEARCH PAPER

Accelerating iterative CT reconstruction algorithms using Tensor Cores

Mohsen Nourazar[1] · Bart Goossens[1]

Received: 15 July 2020 / Accepted: 28 December 2020 / Published online: 28 January 2021

Abstract
Tensor Cores are specialized hardware units added to recent NVIDIA GPUs to speed up matrix multiplication-related tasks, such as convolutions and densely connected layers in neural networks. Due to their specific hardware implementation and programming model, Tensor Cores cannot be straightforwardly applied to other applications outside machine learning. In this paper, we demonstrate the feasibility of using NVIDIA Tensor Cores for the acceleration of a non-machine learning application: iterative Computed Tomography (CT) reconstruction. For large CT images and real-time CT scanning, the reconstruction time for many existing iterative reconstruction methods is relatively high, ranging from seconds to minutes, depending on the size of the image. Therefore, CT reconstruction is an application area that could potentially benefit from Tensor Core hardware acceleration. We first studied the reconstruction algorithm's performance as a function of the hardware related parameters and proposed an approach to accelerate reconstruction on Tensor Cores. The results show that the proposed method provides about $5\times$ increase in speed and energy saving using the NVIDIA RTX 2080 Ti GPU for the parallel projection of 32 images of size 512×512. The relative reconstruction error due to the mixed-precision computations was almost equal to the error of single-precision (32-bit) floating-point computations. We then presented an approach for real-time and memory-limited applications by exploiting the symmetry of the system (i.e., the acquisition geometry). As the proposed approach is based on the conjugate gradient method, it can be generalized to extend its application to many research and industrial fields.

Keywords Parallel iterative CT reconstruction · NVIDIA Tensor Core · GPU Acceleration · Mixed-precision real-time computations · CUDA · Morton space filling curve

1 Introduction

Graphics Processing Units (GPUs), as one of the most feasible parallel structured processors, have proven their power in facilitating research in a wide range of fields, including high-performance computing, data centers, medical imaging, and machine learning. Among these, GPU-based machine learning applications, and more specifically deep learning, have significantly grown in use in recent years [18]. To address this need, NVIDIA has introduced a specialized computing unit called Tensor Core that speeds up neural network training and inferencing operations in deep learning by offering enormous acceleration in matrix computations [21]. Tensor Core-powered GPUs can offer more than a hundred TFLOPS performance [4].

Although the Tensor Core's tremendous performance is tempting to use, its application is more restricted than general CUDA core. First, Tensor Cores only perform a specific form of matrix multiply–accumulate operation. Second, the precision of Tensor Core floating-point operations is limited to half-precision. Since every application with matrix operations at its core, including non-machine learning image and video processing applications, could potentially benefit from the performance improvements these newly introduced Tensor Cores provide, it will be a valuable study to analyze the performance and the effects of these constraints for these non-machine learning applications.

Computed tomography (CT) is one of these applications that has been widely applied to non-destructive testing and contact-free inspection methods, such as medical imaging,

✉ Bart Goossens
bart.goossens@ugent.be

Mohsen Nourazar
mohsen.nourazar@ugent.be

[1] Department of Telecommunications and Information Processing, imec-IPI-Ghent University, 9000 Ghent, Belgium

age determination, and industrial materials testing [16]. Two main challenges in CT include decreasing radiation exposure to reduce risk of biological and material damage, and improving the reconstruction process by decreasing processing time while increasing resolution. In this regard, several hardware- and software-based approaches have been proposed to address these concerns [10, 25]. Compared to the traditional methods used to reconstruct CT images, Iterative Reconstruction (IR) methods provide superior images but require more processing resources. Thanks to improvements in processor performance, IR is now considered the preferred reconstruction method due to the simultaneous noise reduction and quality improvement it can provide [25]. Although it has recently received increasing attention from the high-performance computing domain, reconstruction is still a resource-hungry application [27, 28]. Thus, these novel parallel architectures provide new opportunities for accelerating IR. And given the increasing real-time applications in both medical and non-medical scanning, there are many areas that would benefit from this technology.

In this regard, we demonstrate the application of NVIDIA Tensor Cores to accelerating CT forward-projection (FP) and back-projection (BP) algorithms, which are of the most demanding kernels in iterative reconstruction approaches. For this purpose, the distance-driven projection method is used to build a system matrix. Although any other projection method can be used in our approach, distance-driven projection is used because of the advantages of fast and accurate reconstruction [5, 6]. To take advantage of GPU hardware acceleration, the Tensor Cores are used to perform the forward-projection by multiplying the system matrix with a vector consisting of the image pixel intensities and the back-projection by multiplying the transpose of the system matrix with a vector consisting of the sinogram intensities. Because Tensor Cores operate in a mixed 16-bit/32-bit floating-point precision, we expect a loss in accuracy compared to a purely 32-bit floating-point implementation. Therefore, our goal is to also investigate whether this loss in accuracy is acceptable for CT reconstruction applications.

The contributions of this paper can be summarized as follows:

- Application of Tensor Cores to a non-machine learning application; in particular, we discuss how to deal with several algorithmic challenges when using Tensor Cores.
- A pseudo-Morton ordering algorithm is proposed to improve the sparsity pattern of the system matrix, which will improve performance and memory utilization.
- An approach is proposed for exploiting the symmetry of the system and to further reduce the required system matrix memory.

Although—as far as we are aware of—applying Tensor Cores to this purpose is not reported yet, GPU-accelerated (non-Tensor Core) CT reconstruction algorithms have already been reported for both CT and Positron Emission Tomography (PET) applications, e.g., [13, 17, 26]. The work presented in [9] developed a fast GPU-based algorithm for reconstructing high-quality images from under-sampled and noisy data. Xie et al. [29] proposed a 10–16 times faster CUDA-based implementation for the projection in iterative CT reconstruction algorithms. Both Li et al. [16] and Sabne et al. [23] implemented a GPU-optimized Model-based Iterative Reconstruction (MBIR). Finally, Hidayetoğlu et al. [14] proposed a GPU-based memory-centric X-ray CT reconstruction which is implemented using multi-stage buffering, pseudo-Hilbert ordering, and sparse matrix–vector multiplication (SpMV). Because of its memory-centricity, it has some similarities with our work, such as dealing with sparse matrix data structures and using space-filling curves. However, our work focuses specifically on the application of the Tensor Cores and its limitations (e.g., programming limitations and precision limitations) in a non-machine learning application. We use different data structures and a different ordering method with lower computational complexity than the ordering algorithm used in their work.

This paper is organized in five sections. Background information on the concepts used later in the paper is presented in Sect. 2. Section 3 presents the proposed method in detail. The experimental results and a discussion are presented in Sect. 4. Section 5 contains the conclusion.

2 Background information

In this section, we give some background information on the three main concepts we build our approach on: (1) the distance driven projection method, (2) representations for large sparse matrices and (3) exploiting symmetries in the system.

2.1 Distance-driven projection

Projection and back-projection in CT reconstruction are typically conducted using one of these three main methods: pixel driven, ray driven, and distance driven [5]. The pixel-driven method works by passing a ray from the source through the center of each pixel of interest to the detector cells and then interpolates the contribution of two adjacent detector cells to this projected pixel value. The ray-driven method works by passing a ray from the source through the image to the center of each interest detector cell and then integrates the pixel values along this ray to calculate the projection value for the detector cell. The distance-driven method, on the other hand, converts the problem of projection–backprojection into a one-dimensional re-sampling problem by mapping both the

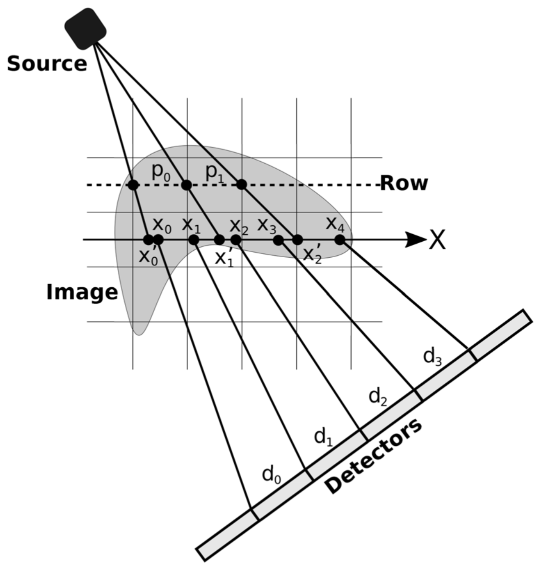

Fig. 1 Distance-driven projection, where the detector and the pixel boundaries are mapped onto the *x*-axis

detector cell boundaries and the pixel boundaries onto a common axis, e.g., the *x*-axis [5, 6]. It then calculates the length of the overlap to determine the contribution of each pixel to the corresponding detector cell.

Because of grid artifacts that tend to appear in the image domain when using the ray-driven back-projection and in the projection domain when using pixel-driven projection, the ray-driven method is well suited for forward-projection, while back-projection works well with the pixel-driven method [5, 6].

The distance-driven methods usually lead to better reconstruction image quality than both ray-driven and pixel-driven methods. Its projection and back-projection operations are matched, or symmetric, and can be used in iterative reconstruction algorithms where many projection and back-projections are performed [24]. In addition, this method has the potential for better performance because of its sequential memory access pattern (similar to the pixel-driven method) and relatively low arithmetic complexity (comparable to the ray-driven method), which make it more suitable for hardware implementations [5, 6].

Figure 1 shows a schematic of the distance-driven projection for a given row, in which the pixel boundaries and the detector cell boundaries are both mapped onto the *x*-axis. In Fig. 1, detector boundaries are denote by X_i, pixel boundaries by X'_i, detector values by d_i, and pixel values by P_i. In this example, the value of detector d_1 can be calculated by:

$$d_1 = \frac{(X'_1 - X_1)P_0 + (X_2 - X'_1)P_1}{X_2 - X_1}, \tag{1}$$

which is a linear equation in P_0 and P_1. This means that the distance-driven projection describes a linear transformation from the image domain to the projection domain. As described in section 3, this linear transformation can be represented by a large sparse system matrix.

2.2 Representation of large sparse matrices

When multiplying a large sparse matrix by a dense vector, even if the sparse matrix can be stored in memory, the memory bandwidth for reading from memory would limit its performance. Consequently, most algorithms store the sparse matrix in a compressed format. Several compressed data structure formats do compression by storing only the non-zero values in a list, and the efficiency of these methods usually depends on the required memory bytes for storing the indices or pointers of the non-zero values. However, when the application's performance is essential, the required memory bandwidth must also be taken into consideration when choosing the appropriate data structure. The memory bandwidth depends highly on the memory access pattern of the application, and as a result, there is a trade-off between memory usage and performance.

There are several well-known compressed data structures for storing sparse matrices in scientific computing such as COO, CSC, CCS, and CSR [22]. Among these formats, compressed sparse rows (CSR) is one of the most efficient and famous standard formats.

The CSR format stores the non-zero values of each matrix row in consecutive memory locations as well as an index to the first stored element of each row. In a common and general-purpose version, the CSR directly stores sparse matrix using a floating-point array and two one-dimension integer arrays for storing the column indices (CI) and row indices (RI). The non-zero values are stored in the floating-point array, and the column indices of these non-zero values are stored in the CI array. Then, the RI array stores the position of the first element of each row inside the floating-point array. Since the RI array contains an index for each row of the sparse matrix, only $m + \text{nnz}$ indices are required to store an $m \times n$ sparse matrix containing nnz non-zero values.

Another compression format that is more suitable for applications that require fetching a block of data from memory is the Block Compressed Sparse Row (BSR) format [22]. The BSR method stores two-dimensional non-empty blocks of elements instead of storing only the non-zero values, and uses a similar indexing format as the CSR method. Although the block-based formats are less memory efficient, they can provide better performance for the applications that work on blocks of data, such as the application we present here. Additional details on the data structure used in this work are presented in Sect. 3.2

Table 1 Available configurations based on the fragment size

Fragment size	16.16.16	32.8.16	8.32.16
Tile of A	16×16	32×16	8×16
Tile of B	16×16	16×8	16×32
Tile of C	16×16	32×8	8×32

2.3 Tensor Cores

As mentioned in the Introduction, expanding AI and deep-learning applications motivated NVIDIA to release its recent architectures, Volta, Turing, and Ampere, with new specialized execution units designed specifically for performing matrix operations, called Tensor Cores. A Tensor Core is capable of performing one matrix-multiply-and-accumulate operation on a 4×4 matrix in one GPU clock cycle. In its first version introduced by the Volta architecture, the Tensor Core was able to perform computations on half (according to the IEEE 754-2008 standard) and mixed-precision floating-point numbers. In both modes, Tensor Cores perform matrix multiplication in half-precision, but the accumulation could be performed in half or single precision. NVIDIA's Turing architecture, as the successor to the Volta, introduced a new and enhanced version of the Tensor Core unit by adding new integer modes, INT8 and INT4, for applications that can tolerate inaccurate computations, such as neural networks. In this architecture, the Streaming Multiprocessor (SM) consists of four processing blocks that NVIDIA refers to as Sub-Cores, and each Sub-Core has two Tensor Cores and one warp scheduler.

Although Tensor Cores perform 4×4 matrix multiplication at a hardware level, the programming interface exposes the matrix operations only at the warp level and in terms of larger matrices. This interface, the Warp Matrix Function (WMMA) API, gives the programmer access to the Tensor Cores as warp-wide operations. This means that all threads in a warp must cooperatively work together to perform the computation of $\boldsymbol{D} = \boldsymbol{A} \times \boldsymbol{B} + \boldsymbol{C}$, where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{D}$ can be tiles of larger matrices. However, the programmer does not have direct access to the 4×4 matrix multiplications. Instead, the possible matrix sizes, called fragment sizes, are specified by a WMMA API.

The fragment sizes are represented using the notation $M \times N \times K$, where $M \times K$ is the dimension of Tile $\boldsymbol{A}$, $K \times N$ is the dimension of Tile $\boldsymbol{B}$ and thus $\boldsymbol{C}$ and $\boldsymbol{D}$ have dimension $M \times N$. Table 1 presents the possible configurations according to the available fragment sizes on the latest version of CUDA 10.2 for half-precision mode.

To program the Tensor Cores, CUDA code must be adapted to (1) use the WMMA API and (2) to use the available fragment sizes. For matrix multiplications of sufficiently large matrices, the fragment size can be chosen freely. However, for other tasks, the algorithm may have to be modified to fit the Tensor Core architecture.

From a reconstruction quality perspective, accuracy may be reduced due to the use of half-precision floating-point used by matrix multiplication, compared to single-precision floating-point calculations. Therefore, the choice of using Tensor Cores presents an 'accuracy versus speed-up' trade-off. The question is then whether this accuracy decrease is acceptable for practical applications. We investigate this further in Sect. 4.

3 The proposed approach

3.1 Overview

The distance-driven projection describes a linear transformation from the image domain to the projection domain, as described in Sect. 2.1. Thus, reconstruction from projection can be considered as solving a system of linear equations:

$$\boldsymbol{Ax} = \boldsymbol{b}, \tag{2}$$

where $\boldsymbol{A}$ is a system matrix that describes the distance driven projection, $\boldsymbol{x}$ is an $N \times 1$ column vector that represents the intensities of the image, and $\boldsymbol{b}$ is the projection data for each angle and detector position collected by sensors. In this notation, vectors are obtained from images using column stacking. The dimensions of the system matrix are determined by the number of projection angels, m, the number of projections per each angel, k, and the image dimensions, $w \times h$. Therefore, each entry of the system matrix has two sub-indices, ij and rq, where sub-indices ij refer to the pixel location and indices rq refer to projection number and ray angle in each projection. That is, $\boldsymbol{A}$ is an $M \times N$ rectangular matrix where $M = mk$ and $N = wh$.

The reconstruction problem (2) may be either over-determined or under-determined. Consequently, the system is not uniquely solvable in general. Instead, we use least squares minimization of $||\boldsymbol{Ax} - \boldsymbol{b}||^2$; its solution is given by the pseudo-inverse:

$$\hat{\boldsymbol{x}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}. \tag{3}$$

Due to the size of the matrix $\boldsymbol{A}$, direct inversion of $(\boldsymbol{A}^T\boldsymbol{A})^{-1}$ is not practical; therefore most methods in literature revert to iterative approaches. In this work, the pseudo inverse is calculated iteratively using the Conjugate Gradient method (Algorithm 1). The resulting iterative CT reconstruction is carried out as iterations of computing residual and gradient matrices by performing forward-projection ($\boldsymbol{Ax}$) and its adjoint, back-projection ($\boldsymbol{A}^T\boldsymbol{b}$), between image and projection domains. Therefore, each iteration contains two system matrix multiplications. Even if the image and projection

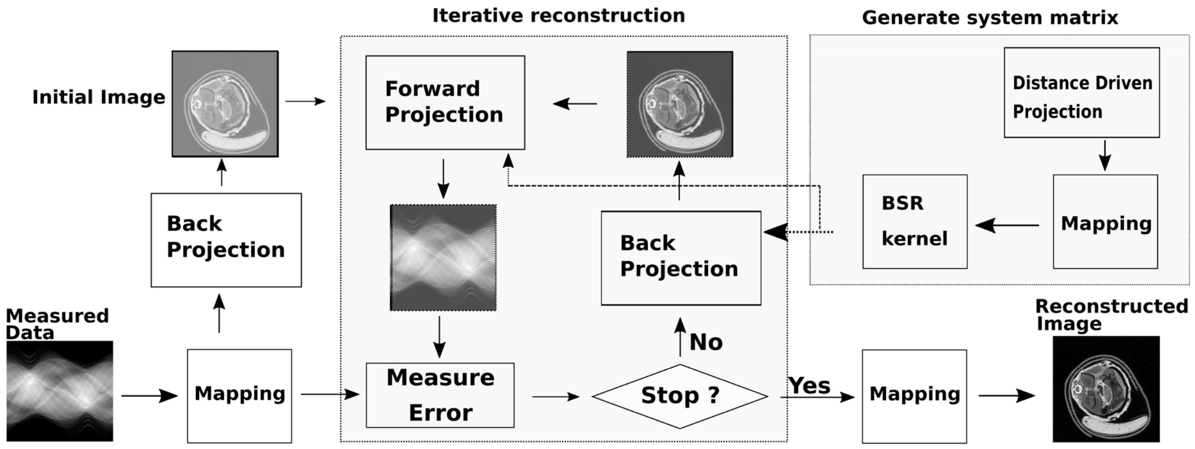

Fig. 2 High-level block diagram of the proposed approach

sizes are as small as 512×512 and 720×512, respectively, more than 380×10^9 bytes of memory is needed to store the system matrix when a 4-byte word is dedicated for each element, which itself can be a great challenge. While the memory problem could be skipped by instead performing on-the-fly computations, the more than 2×10^{16} floating-point operations which would be required for each projection is beyond the tolerance for most of the applications even with the acceleration provided by the NVIDIA Tensor Cores.

Algorithm 1 The conjugate gradient method.

```
1:  procedure CG($\boldsymbol{x}$(input), $\boldsymbol{A}$(system matrix), $\varepsilon$(tolerance))
2:      $\boldsymbol{x} \leftarrow 0, \boldsymbol{r} \leftarrow \boldsymbol{A}^T\boldsymbol{y}, \boldsymbol{p} \leftarrow \boldsymbol{r}, \boldsymbol{\phi}' \leftarrow \boldsymbol{r}^T\boldsymbol{r}$
3:      while $\phi > \varepsilon \cdot \text{length}(\boldsymbol{x})$ do
4:          $\phi \leftarrow \phi', \boldsymbol{q} \leftarrow \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{p}$
5:          $\alpha \leftarrow \phi/(\boldsymbol{p}^T\boldsymbol{q})$
6:          $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\boldsymbol{p}$
7:          $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha\boldsymbol{q}$
8:          $\phi' \leftarrow \boldsymbol{r}^T\boldsymbol{r}$
9:          $\beta \leftarrow \phi'/\phi$
10:         $\boldsymbol{p} \leftarrow \boldsymbol{r} + \beta\boldsymbol{p}$
11:     return $\boldsymbol{y}$
```

Luckily, as we will explain below, the system matrix is sparse and significantly less memory and floating point computations will be required by performing some pre-computations.

As previously mentioned, the distance-driven projection method is used in this work to generate the elements of the system matrix. Accordingly, each row of the system matrix gives the proportion of all image pixels from a single ray going through the image. Therefore, most of the system matrix elements are equal to zero, since only a small percent of these pixels contribute to any individual ray. This means the system matrix $\boldsymbol{A}$ can be considered as a large sparse matrix and therefore, the required memory can be greatly reduced by the use of the data formats described in Sect. 2.2. Due to the requirements of Tensor Cores, the BSR format with a non-square block size is used in this work. The block sizes (size of Tile B) are defined based on the fragment size of Tensor Cores, as presented in Table 1.

Since the efficiency of the sparse matrix data structures depends on the sparsity pattern of the system matrix, an application-specific coordinate mapping algorithm is used in this work to improve the efficiency of the proposed approach. The high-level block diagram of the proposed approach is presented in Fig. 2. As we will explain, the coordinate mapping algorithm is also applied to the image and the projection matrices in addition to the system matrix.

Finally, we will show that by exploiting system's symmetry, even further improvements are possible in both computational performance and the forward-projection's memory requirements.

In the following sections, we will explain the BSR data structure used to store the sparse system matrix, the application-specific coordinate mapping algorithm to improve the sparsity pattern, the method used to exploit the symmetry of the system, and finally, the complete structure of the proposed method.

3.2 Data structure

Due to the hardware characteristics of Tensor Cores and the requirements that are specified by the WMMA APIs, the BSR data structure has been implemented in this work for efficient storage of the system matrix. In the BSR format, non-empty blocks are the building elements, instead of the non-zero values that are the basic elements of the CSR method. In this format, a block containing at least one non-zero element is considered a non-empty block.

Given α and β as the block size parameters, the BSR method partitions the $M \times K$ sparse matrix $\boldsymbol{A}$ into

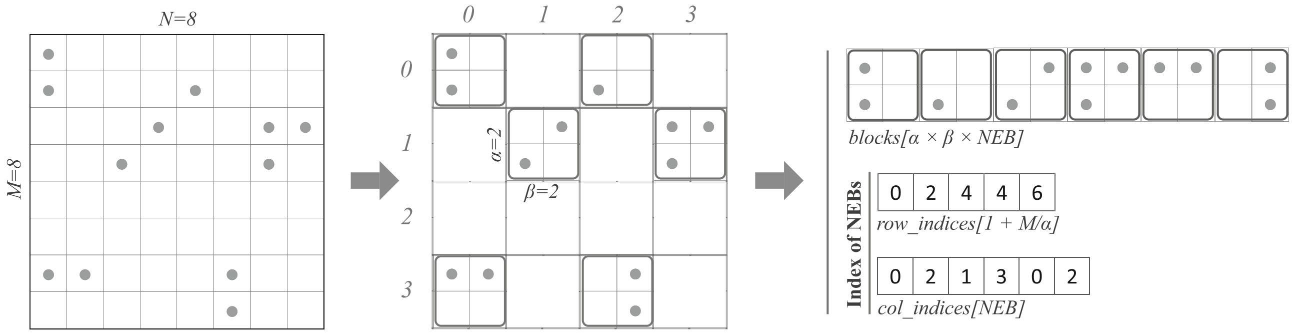

Fig. 3 An example of storing a sparse matrix using the BSR method

$M/\alpha \times N/\beta$ blocks. No compression is applied inside blocks, and blocks contain both zero and non-zero values. Therefore, a fixed amount of memory is allocated for each non-empty block. An example application of the BSR method is displayed in Fig. 3. As shown, a one-dimension floating-point array, *blocks*$[\alpha \times \beta \times NEB]$, is allocated to store the values of non-empty blocks, and two one-dimension integer arrays are dedicated for storing the indices of these non-empty blocks, *row_indices*$[1 + M/\alpha]$ and *col_indices*$[NEB]$, where NEB is the number of non-empty blocks.

Finally, since the Tensor Cores perform mixed-precision matrix operations, the data type for storing the *blocks* array is considered to be a half-precision floating point.

3.3 Pseudo-Morton ordering algorithm

The efficiency of using Tensor Cores for sparse matrix multiplications highly depends on the sparsity pattern of the matrix. Maximum performance will be achieved when the density of the non-empty blocks is *maximal*, which also means having the minimum number of non-empty blocks.

Although the sparsity pattern of the system matrix and, consequently, the sparsity pattern of non-empty blocks depends on the physics of the system, this pattern can be improved by applying an appropriate mapping algorithm. However, the system matrix pattern must first be analyzed to determine the optimal mapping algorithm, .

As mentioned in Sect. 3, the system matrix entries give the proportion of pixel *ij* from the ray *rq* passing from source to the detector. Therefore, pixels close to each other will likely have a similar proportion of rays passing through the region.

According to the locality properties of the system matrix, space-filling curves and especially Morton space-filling curves are a good candidate [3, 8, 14, 19]. Morton codes are constructed by interleaving bits of integer coordinates representing a spatial position. Therefore, Morton codes map n-dimensional data points to a one-dimensional space while preserving the spatial locality of the data points. As a result, coordinates that are spatially close to each other in the N-dimensional space will have Morton numbers that are also close to each other.

In addition, since the Morton code has a lower computational cost compared to other space-filling curves, a pseudo-Morton mapping has been applied to the system matrix in this work. For efficiency, our pseudo-Morton mapping does not convert the coordinate indices all the way down to Morton codes, but instead performs bit shuffling operations directly onto the coordinate indices themselves, hence the name *pseudo-Morton*. We assume blocks of size B_x and B_y where B_x and B_y are both powers of 2. The main idea is that for a subsequent range of B_xB_y 1D indices, the corresponding 2D coordinates are all within the same block of size $B_x \times B_y$. It is not necessary to map the $\log_2(B_x) + \log_2(B_y)$ least significant bits onto Morton codes, as this does not bring any improvements to the rest of our algorithm, allowing the operation to be implemented using a limited number of bitwise operations. The proposed mapping algorithm is shown in Alg. 2, where B_x and B_y are the mapping block sizes.

The mapping algorithm is applied to both the row indices as well as the column indices. Hence, to reverse the effect of the applied permutation to the system matrix, the mapping algorithm was also applied to the images before performing the multiplications and to the result matrices.

Algorithm 2 Pseudo-Morton mapping algorithm

Parameters: B_x: block width, B_y: block height, W: system matrix width (if applied along the rows) or system matrix height (if applied along the columns)

```
1:  procedure MAPPER($a$, $b$)
2:      $t_x \leftarrow a \mod B_x$
3:      $t_y \leftarrow b \mod B_y$
4:      $b_x \leftarrow \lfloor a/B_x \rfloor$
5:      $b_y \leftarrow \lfloor b/B_y \rfloor$
6:      $t_x^{(2)} \leftarrow bx \mod B_x$
7:      $t_y^{(2)} \leftarrow by \mod B_y$
8:      $b_x^{(2)} \leftarrow \lfloor b_x/B_x \rfloor$
9:      $b_y^{(2)} \leftarrow \lfloor b_y/B_y \rfloor$
10:     $i_1 \leftarrow (t_x^{(2)} \cdot B_y + t_y^{(2)}) + (b_x^{(2)} \cdot (W/(B_y \cdot B_y)) + b_y^{(2)}) \cdot B_x \cdot B_y$
11:     $i_2 \leftarrow (t_x \cdot B_y + t_y) + i_1 \cdot B_x \cdot B_y$
12:     return $i2$
```

Note: because B_x and B_y are powers of two, the division and modulo operations are optimized by respectively bitwise shift and bitwise AND operations.

3.4 Exploiting symmetries of the system matrix

Symmetry can be considered one of the most important properties of any system, especially when dealing with system matrices. In a symmetric system,[1] a part of a system matrix becomes exactly the same as the other part by applying some kinds of transformation such as rotation and flipping of the input image; as a result, the size of the system matrix can be greatly reduced.

In addition, symmetric system matrices are of great interest for parallel structured computing architectures. By loading a single part of the system matrix from memory, several parallel kernels can perform their computations. However, the extra computation required for performing the transformation can itself limit the maximum achievable performance [7].

In Sect. 3.1, it is explained that in forward-projection, $\boldsymbol{x}$ is a column matrix and the representative of a single input image. Since the Tensor Cores perform matrix–matrix multiplications instead of matrix–vector multiplications, $\boldsymbol{x}$ must be used as a 2D matrix with dimension according to the selected fragment size for Tensor Core operations. Filling the other columns of $\boldsymbol{x}$ with zeros is not an option, because then the advantages of using Tensor Cores for acceleration disappear. A better alternative relies on filling the matrix $\boldsymbol{x}$ with several input images. This works well, e.g., when multiple CT slices or volumes must be reconstructed in parallel (assuming the data fits in CPU or GPU memory). However, for real-time applications in which a time dimension is present (e.g., real-time reconstruction during a surgical

[1] Note that we here also consider symmetries other than symmetry along the matrix diagonal, as in linear algebra.

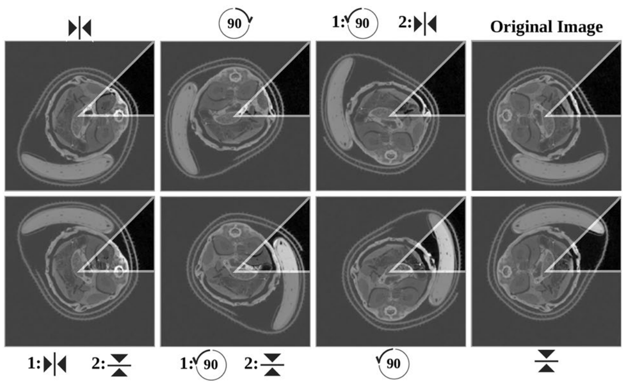

Fig. 4 The eight available transformations, obtained by mirroring, flipping and 90° rotation

procedure), it can not be considered an ideal method, since latency is increased.

Our approach relies on splitting a single input image into several columns. Due to the symmetry of the system, it is possible by splitting the input image into several sub-images and then use the transformed version of the resulting sub-images as the columns of matrix $\boldsymbol{x}$. In addition to exploiting the maximum performance of Tensor Cores, this approach reduces the size of the system matrix by the same factor, which is advantageous for memory consumption.

However, there is one practical restriction on the possible transformations: the symmetry transformations should not impose heavy computations by requiring re-sampling of the pixel grid. Therefore, only rotations of 90° and reflections that do not require re-sampling of the pixel grid are acceptable [7]. While other inexpensive schemes could be considered [1], these sampling schemes are not required in our application because 8 symmetries nicely lead to the minimal required number of columns of $\boldsymbol{x}$ (i.e., eight, see Sect. 2.3). Fig. 4 shows the eight available transformations (combinations of 90° rotations and reflections, taking into account the restriction) and the section that will be extracted from each transformed image to fill the columns of x.

In addition to the transformations applied to the input image, as displayed in Fig. 5, a new transformation kernel including shifts and flipping is applied to the columns of the projection matrix to reverse the initial image transformations. Then, the transformed columns of the projection matrix are summed to build the final projection image. Finally, a masking kernel is applied to the system matrix to remove the unused elements.

In this work, the application of this idea is only proposed for forward-projection. Therefore, the extension of this idea to back-projection and consequently to the entire iterative reconstruction process is considered as future work.

Although the aforementioned system matrix symmetries are not present in all CT systems, there do exist solutions to

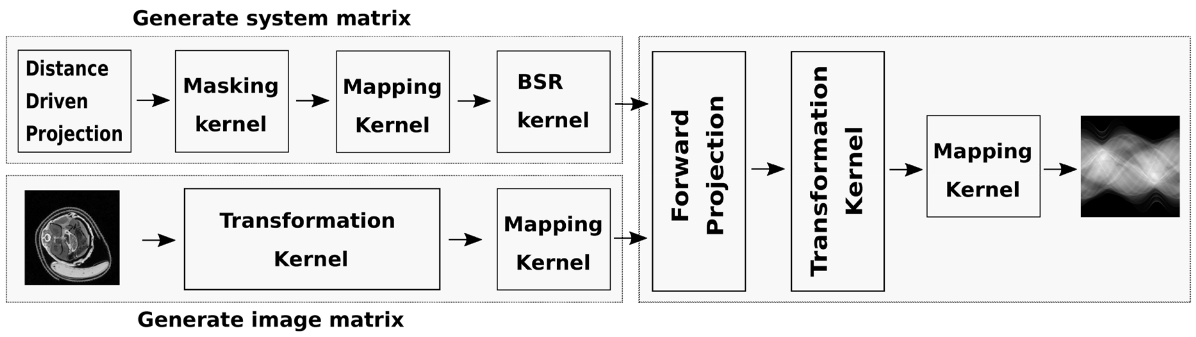

Fig. 5 High-level block diagram of the forward-projection using the proposed approach for exploiting the symmetry of the system

Table 2 System matrix specifications

	Image size					
	64×64	128×128	256×256	320×320	384×384	512×512
System matrix size ($\times 10^9$)	1.5	6	24	37.8	54.3	96.6
Non-zero values ($\times 10^6$)	37.9	78.4	159.5	200	240.8	321.9
Sparsity (%)	97.49	98.7	99.34	99.47	99.56	99.67

mitigate the lack of symmetry for certain cases. In particular, CT systems may contain a detector pixel offset with a fractional (non-integer) part, causing the symmetry to break. By performing band-limited re-sampling of the detector array (e.g., sinc-interpolation with a Kaiser window), any asymmetries due to the offsets can be avoided. In addition, yet another solution consists in quantizing the slopes of the rays (i.e., the tangents of the ray angles) to rational numbers—this will induce the weight functions to be periodic as a function of the system matrix row and/or column. While this technique can also be exploited to reduce the system matrix memory requirements, we did not consider it for this work because it involves an approximation of the ray angles which may degrade the accuracy of the reconstruction.

Finally, the back-projection uses the transposed version of the forward-projection. Since the transposition can be done on-the-fly when reading out the matrix coefficient blocks, no separate data structure is required for the backward-projection. Remarkably, this observation also applies to the implementation with eight symmetry transformations.

4 Results

In this section, the experimental setup and the sizes considered for images and projection are explained. After that, the efficiency of the applied mapping algorithm on reducing the size of the system matrix is presented. The results of the proposed iterative reconstruction are given next, followed by the measured error of reconstruction. As mentioned in Sect. 3.4, the idea proposed for exploiting symmetries of the system matrix is only implemented for the forward-project. Thus, the last results are dedicated to the projection of a single image split into 8 sectors based on the proposed idea.

4.1 Experimental setup and configurations

To evaluate the performance and efficiency of the proposed method, the experiments have been carried out on a system containing a GeForce RTX 2080 Ti, an i7-9800X CPU, and 32GB of DDR4 RAM. The operating system was Ubuntu 18.04 and CUDA 10.2 was used to build the kernels.

The NVIDIA GeForce RTX 2080 Ti has a TU102 GPU, which is designed based on the Turing architecture. The full implementation of the TU102 includes more than 18 billion transistors produced on the production process of TSMC's 12 nm FFN and contains 72 Streaming Multiprocessors (SMs), with each SM containing 64 CUDA Cores and 8 Tensor Cores. In total, it includes 4608 CUDA cores and 576 Tensor Cores. However, the GPU integrated in RTX 2080 Ti includes 68 SM and 544 Tensor Cores. Each Tensor Core can perform up to 64 floating-point fused multiply–add (FMA) operations per clock using FP16 inputs. The 8 Tensor Cores in an SM can perform a total of 512 FP16 multiply and accumulate operations per clock, or 1024 total floating-point operations per clock.

The evaluations are performed by considering a fixed size for projections, 720×512, and six different sizes for input image matrix. The image sizes are: $512 \times 512, 384 \times 384, 320 \times 320, 256 \times 256, 128 \times 128$, and 64×64. Specifications of the system matrix for different image matrix sizes are presented in Table 2.

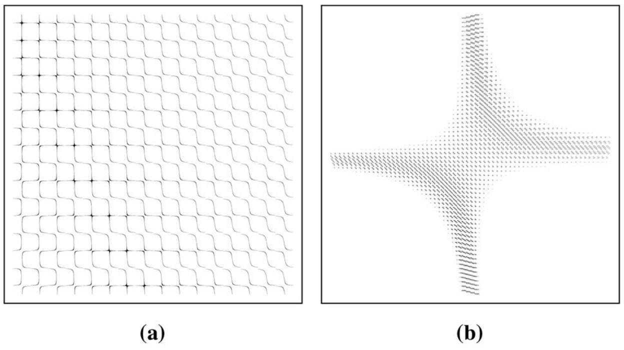

Fig. 6 Visualizing the effect of mapping algorithm on the sparsity pattern of the system matrix; (**a**) a random section of system matrix before mapping; (**b**) the same random section after mapping

Table 3 Number of non-empty blocks (NEBs) before and after applying the pseudo-Morton mapping code

Fragment size	16.16.16	32.8.16	8.32.16
Block size (α, β)*	16×16	32×16	8×16
Blocks count ($\times 10^6$)	377.5	188.7	755
NEBs before (%)	5.19	8.83	3.36
NEBs after (%)	2.01	2.57	1.55

* Block size (α, β) is equal to the dimension of Tile $\boldsymbol{A}$

4.2 Mapping algorithm

Our approach's computational and memory efficiency strongly depends on the sparsity pattern of the system matrix, which can be controlled by adjusting the parameters of the mapping algorithm. In practice, we found the choice $B_x = 4, B_y = 2$ to give good sparsity patterns. A visualization of its effect is shown in Fig. 6. Figure 6a, b display a random section of the system matrix before and after applying the mapping algorithm, where the black points indicate the presence of non-zero values.

Table 3 presents the percent of non-empty blocks (which in fact determines the size of system matrix) depending on the available fragment sizes. As shown in Table 3, the mapping code reduces the matrix size by a factor 2 to 3. In addition, it shows that the minimum percent of non-empty blocks and consequently, the smallest system matrix, is achieved when the fragment size is 8.32.16. Therefore, it can be expected that the best performance will also be achieved for this fragment size.

As Table 3 shows, fragment size 32.8.16 presents the worst result among the other fragment sizes. Whereas, to exploit the symmetry of the system using the eight available transformations as explained in Sect. 3.4, it is required to use the fragment size 32.8.16. This restriction justifies the value of the further investigation on other inexpensive sampling schemes to increase the number of available transformations up to 16 or even 32, respectively, for the fragment size 16.16.16 and 8.32.16.

4.3 Evaluation of Tensor Cores for iterative reconstruction

In this section, the results of the proposed iterative reconstruction which consists of both accelerated forward and back-projection are presented. To provide a better comparison of the performance of Tensor Cores for this application, several non-tensor approaches are evaluated: ASTRA [26], Tigre [2], Tomopy [12]. All three are open-source libraries for high-performance 2D and 3D tomographic data processing and image reconstruction. For the ASTRA toolbox, we have tested three different reconstruction methods: CGLS, SIRT, and SART. We have used the CGLS method for Tigre toolbox and the SIRT method for Tomopy.

In addition to these, we have implemented four other non-tensor GPU accelerated approaches. The half2 implementation employs the NVIDIA's direct half-precision arithmetic [15] by using the half2 datatype. This is because the maximum throughput for half-precision operations can be achieved when using the half2 datatype [15]. The second implementation uses the CUDA sparse matrix library (cuSPARSE), in which the basic CSR with single (32-bit) precision floating-point data type is used. The third approach, implemented in Quasar [11], performs the iterative reconstruction using on-the-fly computation without pre-computation of the system matrix. This reference method is added to assess the impact of memory latency versus latency of the computations: the method performs the weight calculation on the fly, which is computationally much more costly but at the same time avoids weight memory accesses. Finally, the last implementation is a Quasar implementation for the MLEM reconstruction method [20].

In this experiment, by assuming a multi-slice fan-beam method, several images (slices) are reconstructed in parallel by loading into the columns of the image matrix. The image matrix $\boldsymbol{x}$ must have at least 8, 16, and 32 columns for the fragment size 32.8.16, 16.16.16, and 8.32.16, respectively, to allow straightforward mapping onto the GPU Tensor Cores. However, to reconstruct the same number of images, it is assumed that 32 images are reconstructed in parallel for all the fragment sizes.

For single image reconstruction, we have explained in Sect. 3.4 that the symmetry of the system can be exploited to reconstruct single images in the configuration 32.8.16. We also note again that for this experiment, the approach explained for exploiting the symmetry of the system is not applied.

The average reconstruction time per image for a single iteration is presented in Table 4 for the evaluated approaches.

Table 4 Reconstruction time and energy consumption per image (for a single iteration)

		Precision	Image size					
			64 × 64	128 × 128	256 × 256	320 × 320	384 × 384	512 × 512
Time (ms)	Tensor Core (32.8.16)	Mixed	0.08	0.12	0.28	0.38	0.50	0.80
	Tensor Core (8.32.16)	Mixed	0.7	0.11	0.22	0.29	0.38	0.62
	Tensor Core (16.16.16)	Mixed	0.06	0.11	0.23	0.33	0.44	0.73
	cuSPARSE	32-bit	0.28	0.56	1.30	1.73	2.16	3.12
	Half2 arithmetic	Mixed	0.19	0.37	0.86	1.16	1.50	2.30
	On-the-fly	32-bit	1.25	1.61	2.55	3.24	4.22	6.72
	ASTRA (CGLS)	32-bit	0.92	0.97	1.58	2.08	3.03	4.55
	ASTRA (SIRT)	32-bit	0.81	0.87	1.46	1.94	2.90	4.40
	ASTRA (SART)*	32-bit	0.43	0.44	0.44	0.44	0.48	0.48
	Tigre (CGLS)	32-bit	210	210	226	230	240	250
	TomoPy (SIRT)	32-bit	40	60	80	90	120	140
	MLEM	32-bit	0.47	0.75	1.73	2.51	3.01	4.79
Energy (nJ)	Tensor Core (32.8.16)	Mixed	7	17	60	87	117	190
	Tensor Core (8.32.16)	Mixed	6	15	50	70	93	151
	Tensor Core (16.16.16)	Mixed	6	14	55	80	106	179
	cuSPARSE	32-bit	47	117	306	412	530	782
	Half2 arithmetic	Mixed	29	82	213	296	383	592

* Each iteration of SART processes one single projection direction

Table 5 Memory usage for storing the system matrix (GigaBytes)

	Format	Image size					
		64 × 64	128 × 128	256 × 256	320 × 320	384 × 384	512 × 512
Tensor Core (32.8.16)	BSR(Half)	0.26	0.63	1.68	2.35	3.13	4.98
Tensor Core (8.32.16)	BSR(Half)	0.23	0.52	1.20	1.60	2.04	3.04
Tensor Core (16.16.16)	BSR(Half)	0.25	0.57	1.42	4.94	2.53	3.92
cuSPARSE	CSR(32-bit)	0.23	0.47	0.96	1.2	1.45	1.93

Since our implementation is a gradient-based method, it is better to be compared with other gradient-based implementations. Because, each method requires a different number of iterations for convergence and also has a different error at convergence. Thus, while we present the execution time of different methods (including CGLS, SART, SIRT, and MLEM), it is better to consider reconstruction error (see Sect. 4.4) when comparing the execution time of different methods. It must be emphasized that our approach is not limited to the conjugate gradient and is a general approach that can work with other reconstruction methods.

According to Table 4, the maximum performance is achieved for the fragment size 8.32.16, which is faster than all other implementations and is about 5 times faster than the cuSPARSE implementation. Compared to the on-the-fly method, this result shows the advantage of storing the weights in memory and hence using the proposed BSR data structure. Compared to the ASTRA SART implementation, it should be noted that the SART method processes only one single projection direction in each iteration.

Table 4 also presents the average reconstruction per image energy that is consumed by the NVIDIA GPU during the execution of a single iteration. The average energy consumption values are obtained by polling the nvml APIs during kernel execution. According to Table 4, the proposed approach consumes 3–5 times less power compared to other implementations.

Finally, Table 5 presents the required memory for storing the system matrix for the cuSPARSE implementation (the basic CSR with 32-bit single-precision floating-point data type) and Tensor Core implementation (the BSR data structure explained in Sect. 3.2 with half-precision floating-point data type). Since the Half2 implementation uses the same data structure as the Tensor Core method, it is not presented in Table 5.

4.4 Reconstruction error

Since Tensor Cores perform mixed-precision floating-point operations, a larger numerical error is expected than in the

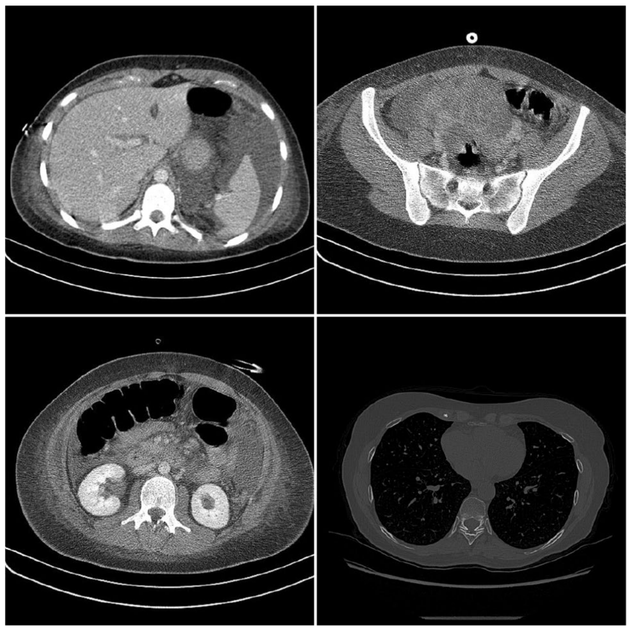

Fig. 7 Four example images of the dataset used for error measurement

case of single-precision floating-point calculations. Therefore, we measure the reconstruction error in comparison with 32-bit floating-point reconstructions. In this paper, the relative reconstruction error is calculated as follows:

$$\text{Error} = \|P - P^{o}\|_F / \|P^{o}\|_F, \tag{4}$$

where P^{o} is the original image and P is the reconstructed image generated using the different reconstruction implementations. $\|.\|_F$ denotes the Frobenius norm.

For measuring the error, a dataset containing 23 input images of class uint8 with the size of 512×512 was used. The input images are reconstructed images, collected from existing CT scanners (e.g., the X-O CT system, Gamma Medica-Ideas, Northridge, California, USA). Figure 7 displays four examples of the dataset. To generate the projection data required for reconstruction, we have used the distance driven forward-projection for the Tensor core and Quasar implementations, and we have used the "line-fanflat" algorithm for the ASTRA implementation.

Figure 8 presents the average error versus the number of iterations measured from different implementations. It shows that the conjugate gradient based methods perform well for the provided dataset. Although it was expected to have a larger error using the Tensor cores due to the mixed-precision datatype, as shown in Fig. 8, the accuracy of both the Tensor CG and the non-Tensor (32-bit CG) implementations are nearly equivalent—indicating that the mixed-precision computation offered by Tensor Cores has the potential to be used for CT reconstruction.

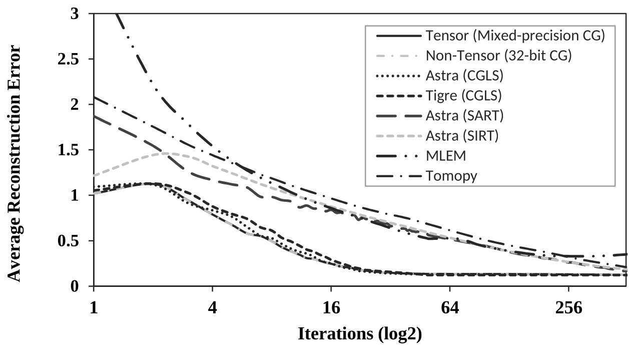

Fig. 8 Average relative reconstruction error

4.5 Forward-projection by exploiting the symmetry of the system

The last set of results is dedicated to evaluating the projection of a single image split into 8 sectors based on the proposed method and the transformations for exploiting the symmetry of the system, as explained in Sect. 3.4. This approach has identical numerical accuracy with multi-slice approach in case the detector pixel offset is zero. The execution time of the forward-projection, the consumed energy, and the memory usage for storing the system matrix are presented in Tables 6, 7. The achieved results not only show the efficiency of the proposed method for real-time applications, but also highlight that the reasonable size of the system matrix warrants extending this method to the whole reconstruction process.

5 Conclusion

In this paper, we demonstrated the feasibility of using NVIDIA Tensor Cores for accelerating iterative CT reconstruction, as an example of a non-machine learning applications. For this purpose, we used the Tensor Cores to perform the sparse matrix–matrix multiply–accumulate operations of the projection and back-projection kernels. To overcome the hardware and software limitations of using Tensor Cores and also the challenges of implementing a memory-centric iterative reconstruction, we proposed and implemented an approach using compressed BSR data structure for storing the sparse system matrix, a pseudo-Morton encoding to improve the sparsity pattern of the system matrix, and a method to exploit the symmetry of system which makes this approach a practical solution for real-time or memory limited CT applications.

The proposed approach is evaluated by measuring the performance, power, accuracy and memory usage. The presented performance results indicate approximately 5x speedup and energy saving is achieved with the proposed

Table 6 Execution time and energy consumption of the forward-projection kernel

		Precision	Image size					
			64 × 64	128 × 128	256 × 256	320 × 320	384 × 384	512 × 512
Time (ms)	Tensor Core (32.8.16)	Mixed	0.10	0.19	0.44	0.59	0.78	1.22
	cuSPARSE	32-bit	0.72	0.71	1.13	1.23	1.55	2.10
	Half2 arithmetic	Mixed	0.22	0.35	0.72	0.97	1.25	1.83
Energy (nJ)	Tensor Core (32.8.16)	Mixed	6	11	29	42	62	108
	cuSPARSE	32-bit	130	150	260	300	390	530
	Half2 arithmetic	Mixed	20	40	117	193	266	431

Table 7 Memory usage for storing the system matrix (GigaBytes)

	Image size						
	Format	64 × 64	128 × 128	256 × 256	320 × 320	384 × 384	512 × 512
Tensor Core (32.8.16)	BSR(Half)	0.04	0.09	0.23	0.31	0.41	0.63
cuSPARSE	CSR(32-bit)	0.04	0.08	0.16	0.20	0.25	0.33

method, compared to a cuSPARSE-based implementation on the same GPU. The accuracy results show that using mixed-precision computing of Tensor Cores does not impose any significant reconstruction error.

In future work, in addition to extending the proposed idea of exploiting symmetry to the entire reconstruction process, we plan to investigate the extension of our approach to cone-beam (3D) reconstruction and helical CT reconstruction.

Acknowledgements The authors would like to thank Asli Kumcu for proofreading the manuscript. This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

References

1. Akenine-Moller, T.: An extremely inexpensive multisampling scheme. Aug 15, 03–14 (2003)
2. Biguri, A., Dosanjh, M., Hancock, S., Soleimani, M.: TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction. Biomedical Physics & Engineering Express 2(5), 055010 (2016)
3. Buluc, A., Fineman, J.T., Frigo, M., Gilbert, J.R., Leiserson, C.E.: Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In: Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, pp. 233–244 (2009)
4. Carrasco, R., Vega, R., Navarro, C.A.: Analyzing GPU tensor core potential for fast reductions. In: 2018 37th International Conference of the Chilean Computer Science Society (SCCC), pp. 1–6. IEEE (2018)
5. De Man, B., Basu, S.: Distance-driven projection and backprojection. In: 2002 IEEE Nuclear Science Symposium Conference Record, vol. 3, pp. 1477–1480. IEEE (2002)
6. De Man, B., Basu, S.: Distance-driven projection and backprojection in three dimensions. Physics in Medicine & Biology 49(11), 2463 (2004)
7. Donne, S., Meeus, L., Quang Luong, H., Goossens, B., Philips, W.: Exploiting reflectional and rotational invariance in single image superresolution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 47–53 (2017)
8. Engel, W.: GPU PRO 360 Guide to GPGPU. CRC Press (2018)
9. Flores, L.A., Vidal, V., Mayo, P., Rodenas, F., Verdú, G.: CT image reconstruction based on GPUs. Procedia Computer Science 18, 1412–1420 (2013)
10. Geyer, L.L., Schoepf, U.J., Meinel, F.G., Nance Jr, J.W., Bastarrika, G., Leipsic, J.A., Paul, N.S., Rengo, M., Laghi, A., De Cecco, C.N.: State of the art: iterative CT reconstruction techniques. Radiology 276(2), 339–357 (2015)
11. Goossens, B.: Dataflow management, dynamic load balancing, and concurrent processing for real-time embedded vision applications using Quasar. International Journal of Circuit Theory and Applications 46(9), 1733–1755 (2018)
12. Gürsoy, D., De Carlo, F., Xiao, X., Jacobsen, C.: TomoPy: a framework for the analysis of synchrotron tomographic data. Journal of synchrotron radiation 21(5), 1188–1193 (2014)
13. Herraiz, J., España, S., Cabido, R., Montemayor, A., Desco, M., Vaquero, J.J., Udías, J.M.: GPU-based fast iterative reconstruction of fully 3-d pet sinograms. IEEE Transactions on Nuclear Science 58(5), 2257–2263 (2011)
14. Hidayetoğlu, M., Biçer, T., de Gonzalo, S.G., Ren, B., Gürsoy, D., Kettimuthu, R., Foster, I.T., Hwu, W.M.W.: MemXCT: memory-centric X-ray CT reconstruction with massive parallelization. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 85. ACM (2019)

15. Ho, N.M., Wong, W.F.: Exploiting half precision arithmetic in nvidia GPUs. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7. IEEE (2017)
16. Li, X., Liang, Y., Zhang, W., Liu, T., Li, H., Luo, G., Jiang, M.: cuMBIR: An Efficient Framework for Low-dose X-ray CT Image Reconstruction on GPUs. In: Proceedings of the 2018 International Conference on Supercomputing, pp. 184–194. ACM (2018)
17. Ma, B., Ca, V., Im, C., et al.: Cuda parallel implementation of image reconstruction algorithm for positron emission tomography. Open Med. Imaging J. **6**(1), 108–118 (2012)
18. Markidis, S., Der Chien, S.W., Laure, E., Peng, I.B., Vetter, J.S.: Nvidia tensor core programmability, performance & precision. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 522–531. IEEE (2018)
19. Nocentino, A.E., Rhodes, P.J.: Optimizing memory access on GPUs using morton order indexing. In: Proceedings of the 48th Annual Southeast Regional Conference, pp. 1–4 (2010)
20. Nuyts, J., Michel, C., Dupont, P.: Maximum-likelihood expectation-maximization reconstruction of sinograms with arbitrary noise distribution using NEC-transformations. IEEE transactions on medical imaging 20(5), 365–375 (2001)
21. NVIDIA: NVIDIA Tesla V100 GPU Architecture (2017)
22. NVIDIA: The API reference guide for cuSPARSE, the CUDA sparse matrix library. http://docs.nvidia.com/cuda/cusparse/index.html (2020). Accessed 24 Jan 2020
23. Sabne, A., Wang, X., Kisner, S.J., Bouman, C.A., Raghunathan, A., Midkiff, S.P.: Model-based iterative CT image reconstruction on GPUs. ACM SIGPLAN Notices 52(8), 207–220 (2017)
24. Schlifske, D., Medeiros, H.: A fast GPU-based approach to branchless distance-driven projection and back-projection in cone beam CT. In: Medical Imaging 2016: Physics of Medical Imaging, vol. 9783, p. 97832W. International Society for Optics and Photonics (2016)
25. Stiller, W.: Basics of iterative reconstruction methods in computed tomography: A vendor-independent overview. European journal of radiology 109, 147–154 (2018)
26. Van Aarle, W., Palenstijn, W.J., De Beenhouwer, J., Altantzis, T., Bals, S., Batenburg, K.J., Sijbers, J.: The ASTRA toolbox: A platform for advanced algorithm development in electron tomography. Ultramicroscopy 157, 35–47 (2015)
27. Wang, X., Sabne, A., Kisner, S., Raghunathan, A., Bouman, C., Midkiff, S.: High performance model based image reconstruction. In: ACM SIGPLAN Notices, vol. 51, p. 2. ACM (2016)
28. Wang, X., Sabne, A., Sakdhnagool, P., Kisner, S.J., Bouman, C.A., Midkiff, S.P.: Massively parallel 3d image reconstruction. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 3. ACM (2017)
29. Xie, L., Hu, Y., Yan, B., Wang, L., Yang, B., Liu, W., Zhang, L., Luo, L., Shu, H., Chen, Y.: An effective cuda parallelization of projection in iterative tomography reconstruction. PloS one 10(11), e0142184 (2015)

Mohsen Nourazar was born in Zanjan, Iran, in 1986. He received the B.Sc. degree in communication engineering and the M.Sc. and Ph.D. degrees in electronics engineering from electrical and computer engineering department, the University of Zanjan, Iran in 2009, 2011, and 2018, respectively. Currently, he is a postdoc researcher in the imec-IPI group of Ghent University, Belgium. His research interests include hardware accelerators, alternative computing, computer architecture, memristive systems, and FPGA implementation.

Bart Goossens is a professor in the Image Processing and Interpretation group of Ghent University, where he currently supervises research on image/video processing, computer vision, AI and heterogeneous platforms mapping tools. He is also a core principal investigator at imec. He earned his master's degree in Computer Science Engineering from Ghent University, Belgium in 2006 and the Ph.D. degree from the same university in 2010. In Oct. 2013 he became a professor at Ghent University. His interests in the efficient mapping of image processing and computer vision techniques onto hardware architectures such as a GPU have resulted in the design and development of Quasar (gepura.io/quasar), a brand new high-performance CPU/GPU-programming solution that offers greatly reduced development time. He is author of more than 100 scientific papers in journals and conference proceedings.