FPGA-accelerated Automatic Alignment for Three-dimensional Tomography

Shuang Wen
School of Mathematical Sciences, Peking University
Email: wenshuang@pku.edu.cn

Guojie Luo
Department of Computer Science, Peking University
Email: gluo@pku.edu.cn

***Abstract*—In the process of tomographic reconstruction, the attitude and center point of a specimen, from which the projection data are collected, suffer from misalignment due to mechanical imperfection and calibration error. Such misalignment leads to poor reconstruction quality. And effective automatic alignment approaches have been proposed. The alignment approaches are of good use for kinds of application scenarios such as X-CT and electron tomography. These scenarios demand not only high performance, but also that the component of automatic alignment can be integrated and upgraded in the whole solution. Thus, we propose an FPGA accelerator for state-of-the-art tomographic alignment algorithm. We first introduce a multi-ray access approach that modifies the order of data access for easier on-chip data management. Making use of BRAMs on FPGAs and effective local data management strategy, data reuse is reinforced, and data transfer latency with DRAM is covered by computation. Also, we introduce an FPGA-customized processing engine at a low cost to improve data throughput. Moreover, a streaming structure with multiple paralleled PEs further improves the performance of our algorithm. Experiments demonstrate that our accelerator achieves a 44.5x speed-up for the state-of-the-art alignment on Xilinx ZCU102 over a 16-thread multicore CPU implementation, and a 1.60x speed-up with 7.8x energy reduction over an OpenCL implementation on Nvidia Titan V.**

I. Introduction

Tomography is a noninvasive imaging technique that combines penetrating waves and computed tomographic reconstruction to get 3D structures of different kinds of the specimen. The specimens are firstly processed by a scanning device, such as electron microscopy or CT scanner, to obtain a series of projections of the specimen. This process can be described by a ray transform [1]. Then, the projections are reconstructed to a 3D structure using the inverse transformation of the ray transform. To obtain a high-quality 3D structure, it is necessary to know the accurate configurations with which the projections are taken. However, due to mechanical instability and specimen/scanner transformation, the recorded projections do not precisely align with the planned configurations. Thus it is necessary to recalculate the configurations before reconstruction. This process is called alignment.

One type of popular alignment methods is to introduce some fiducial markers embedded in the specimen. The high-contrast markers can be tracked so that the relative motion of the specimen is then calculated through the position change of fiducial markers [6]–[8]. Despite its ability for accurate alignment, the requirement for preset markers complicates the specimen preparation process. The markers are usually material with high-contrast like gold beads. This kind of particles leads to artifacts in reconstructions [10]. Those markers also absorb almost all electron beams during scanning, which worsens radiation damage to the specimen [11]. To get rid of marker preset and radiation damages, other methods use image features to replace the function of fiducial markers [2], [12]. Such methods require specific detectors for different kinds of datasets [8], which damages their universality. Alternatively, alignment algorithms based on maximizing the similarity between reprojection of reconstructed result and projection acquired by scanner have been proposed. Winkler and Taylor [9] try different configurations in the parameter space and choose the one with the best cross-correlation coefficient between the reprojection and the acquired projection. Then, methods that formulate the parameters' search into optimization problems were proposed to reduce time consumption [3]–[5]. These kinds of methods are both universally adaptive and accurate for tomography alignment, which makes them useful in these kinds of application scenarios, such as X-CT, electron tomography, MRI and etc.. With the all-in-one trend of tomography equipment [13], there is an urgent need for an embedded system for alignment and reconstruction, to enhance the integration level and performance.

For this purpose, we design an FPGA-accelerated stream structure for tomography alignment. To improve the performance, we use an on-chip storage and data transfer strategy suitable for the tomography alignment method. A targeted PE design is also proposed to reduce data conflict and increase data throughput. The experiments show a 44.5× speed-up over 16-thread multicore code on Intel Xeon E5-2650 v3, and 1.60× speed-up with 7.8x energy reduction over OpenCL implementation on Nvidia Titan V. Our designs for key processes used in tomographic alignment also perform better than previous works.

II. Automatic Alignment Algorithm

In the tomography alignment, we need to clarify several critical variables: we note the projection data as g, which is a stack of 2D images obtained by the scanning device. θ is the projection configuration that describes both the rotational and the translational positions where g is obtained. $f(\theta)$ is the reconstruction, which is a 3D image calculated from the g at the guess of the projection configuration θ. And $\mathcal{R}(\theta)f$ stands for the reprojection of f using the configuration θ,

2576-2621/20/$31.00 ©2020 IEEE
DOI 10.1109/FCCM48280.2020.00031

Algorithm 1 Process of automatic alignment method.

Input: Initial guess for parameters $\theta^{(0)}$, projection data g
Output: Aligned parameters $\theta^{(k+1)}$, reconstruction $f^{(k+1)}$
1: $k = 0$, $\theta^{(0)} = \bar{\theta}^{(0)} = \tilde{\theta}$
2: **while** not converged **do**
3: $\quad f^{(k)} = \mathrm{argmin}_f \, ||\mathcal{R}(\bar{\theta}^{(k)})f - g||^2$
4: $\quad \theta^{(k+1)} = \mathrm{argmin}_\theta \, ||\mathcal{R}(\theta)f^{(k)} - g||^2$
5: $\quad \bar{\theta}^{(k+1)} = \theta^{(k+1)}$, $k = k + 1$

where $\mathcal{R}$ is the ray transform operator. The state-of-the-art alignment algorithm for tomography alignment corrects θ by iteratively minimize the deviation between $\mathcal{R}f$ and g using several optimization methods. For each iteration, two core processes are carried out: $f(\bar{\theta})$ is firstly calculated from g by an inverse transformation of the ray transform with $\bar{\theta}$ which is a configuration copy of θ. Then, the configurations θ are calibrated by minimizing the deviation between $\mathcal{R}(\theta)f(\bar{\theta})$ and g. The specific expression is shown in (1). In this equation, $\bar{\theta}$ is an auxiliary variable to separate the calibration of θ and the calculation of f.

$$\begin{aligned} \min_{\theta,\bar{\theta}} \quad & \left\|\mathcal{R}(\theta)f(\bar{\theta}) - g\right\|^2 \quad \text{s.t. } \bar{\theta} = \theta \\ \text{where} \quad f(\bar{\theta}) \quad & = \mathrm{argmin}_f \left\|\mathcal{R}(\bar{\theta})f - g\right\|^2. \end{aligned} \tag{1}$$

To further explain the process, the pseudo-code for this algorithm is listed in Algorithm 1. The core processes mentioned above are at line 3 and line 4. We solve the minimization at line 3, which is the reconstruction process, by calculating the weighted sum of ray values that pass through point r. Specifically, the process is expressed as:

$$f(r, \bar{\theta}) = \sum_{\bar{\theta}} W(M_{\bar{\theta}}, r) \int_{\mathbb{R}^2} g(p, \bar{\theta}) \delta(M_{\bar{\theta}} r - p) \mathrm{d}p, \tag{2}$$

where $M_{\bar{\theta}}$ is the 4×4 Euler transformation matrix of configuration $\bar{\theta}$. W is the calculated weight related to $M_{\bar{\theta}}$ and r. $\sum_{\bar{\theta}}$ adds up integrals for every single projection image.

On the other hand, the bottlenecks of the process in line 4 are the reprojection calculation and the partial derivative projection calculation. For every projection data, the algorithm calculates a line called ray that intersects with f. Then, the basic elements of f that the line passes through are accumulated to the projection data with the weight that is related to the line length within the corresponding elements. Those basic elements of the 3D image f are called volume pixels (voxels). The calculations can also be expressed as:

$$\mathcal{R}(\theta)f = \int_{L_\theta} f(\boldsymbol{r}) \mathrm{d}|\boldsymbol{r}| \tag{3}$$

$$\nabla_\theta(\mathcal{R}f)(\theta) = \int_{L_\theta} \nabla_\theta f(\boldsymbol{r}) \mathrm{d}|\boldsymbol{r}|, \tag{4}$$

where $L_\theta = M_\theta L_0$, and L_0 is a reference line which is parallel to the z-axis and intersects with the coordinate origin. The main task is the optimization of those two core processes mentioned above.

III. FPGA-specific Optimizations

A. Multi-ray Access

For the calculation of (2), (3) and (4), the traditional way is to trace the voxels traversed by certain rays, which is called a ray-based strategy. Specifically, for the calculation of $\mathcal{R}f$, a ray-based strategy firstly calculates the rays corresponds to $\mathcal{R}f$. Then, as illustrated in a simplified 2D example in Fig. 1(a), it locates the voxels in f that is passed through by those rays. The ray-based strategy has two shortcomings: 1) As shown in Fig. 1(a), the voxels traversed by the same ray scatter in f. And due to the limited capacity of on-chip BRAMs for a typical-sized f or $\mathcal{R}f$, we can only access it from DRAM without the burst transfer's help. 2) The voxel data traversed by different rays overlap a lot. Those conflicts are hard to predict and may lead to long read latencies.

To solve these problems, as Fig. 1(c) and Fig. 1(d) show, instead of looking for voxels needed by certain rays, we access voxel data from f sequentially and find out the rays that traverse current voxel data. In this way, adjacent voxels can always be traversed by adjacent rays, and both f and $\mathcal{R}f$ can be blocked to fit in the efficient but limited BRAM resources. We buffer certain blocks of $\mathcal{R}f$ data in BRAM according to the voxel-based strategy to optimize data reuse. When part of the block data is no longer useful, we load new data and reorganize the data in BRAM.

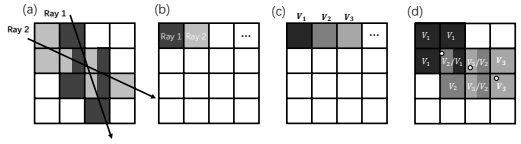

Fig. 1. (a) Ray-based: ray track in data f. (b) Ray-based: data output in data $\mathcal{R}f$. (c) Voxel-based: sequential data read from f. (d) Voxel-based: data output in data $\mathcal{R}f$.

B. Dynamic Block and Arbitrarily Direction Sliding Window

The data transfer on the FPGA platform between programmable logic (PL) and DRAM is usually expensive and untimely. In order to reduce the required AXI bandwidth and avoid waiting on data, a proper technique should be applied. With the help of the voxel-based calculation, as illustrated in Fig. 2(a), the ray positions of certain voxel blocks can be found within a window. The window's size S_{win} is determined by the voxel block size S_b. The centers of every voxel block r_0 are predetermined. And the centers of the projection blocks $M_\theta r_0$ are calculated at runtime depending on r_0 and the Euler matrix M_θ. For every voxel block, the voxels are projected to the projection block with a calculated offset. In the meantime, we load the projection data to be updated but not in BRAM through AXI. When the projection operation for the current block is done, the projection data is updated by the block projection result. Any projection data that is no longer needed

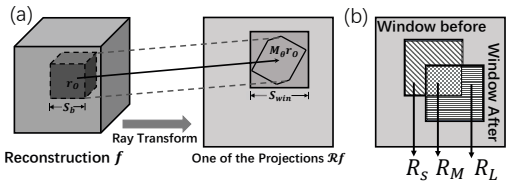

Fig. 2. Local data dependency demonstration.

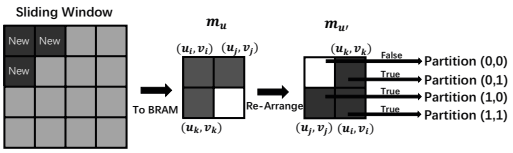

Fig. 4. Write-back operation when new data arrives. The old data is firstly re-arranged to allocate correct BRAM partitions before evicted from sliding window.

for the calculation of the next block is then saved back to the DRAM. Any data that is still useful is moved to a new place according to the offset of the next voxel block, which makes use of a sliding window technique. Specifically, as shown in Fig. 2(b), after one voxel block is calculated, we save the data within R_S to the DRAM and move the data within R_M from the bottom right corner to the upper left corner. This part of the data is well-organized, which will significantly reduce the transmission overhead.

C. Processing Element (PE) Design

In the calculation of projection/back-projection, for every voxel block, the voxels are projected to the projection block with a specific offset. For a voxel position $r = (x, y, z)$, it is projected to position $(u, v) = M_\theta r$ of the projection data. Parameters u and v are not necessarily integers, so a region sized 2×2 in the projection block needs to be accumulated by voxel value multiplied by a calculated weight. For the $\nabla\mathcal{R}f$ calculation, every single point of the integration in (4) requires at least 3 other voxels for the calculation of gradient, as shown in Fig. 3(b). However, those voxels are not saved on chip due to limited BRAM resources. Making use of our multi-

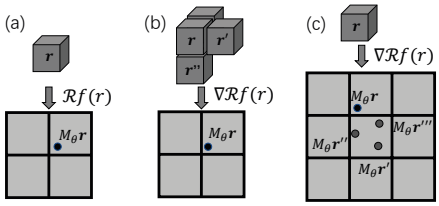

Fig. 3. Demonstration of detailed data dependency in an elementary operation. (a) Projection. (b) Partial derivatives $\nabla_\theta \mathcal{R}f$. (c) Resolved partial derivatives calculation.

ray access strategy, we split the calculation of gradient into 4 independent parts, as shown in (5).

$$\begin{aligned}\nabla_\theta f(r) =& w_0 f(r) + w_{\Delta x} f(r - \Delta x) \\ &+ w_{\Delta y} f(r - \Delta y) + w_{\Delta z} f(r - \Delta z)\end{aligned} \tag{5}$$

Thus one elementary operation of $\nabla\mathcal{R}f$ turns to four operations of $\mathcal{R}f$ with weights. As illustrated in Fig. 5(c), each operation is applied when the corresponding voxel arrives.

BRAM latency limit determines that this kind of accumulation cannot be finished in one cycle (latency = 5 cycles). So we set up a dynamic sliding window in registers as a temporary storage for accumulation .

As illustrated in Fig. 4, the sliding window prefetches the data in the projection block before it is used, along with its position in the BRAM as a reminder. When the new data covers the old data and the old data is no longer needed, the old data is written back to the BRAM at the position recorded in the corresponding reminder. For the write-back operation, there is a problem that the partition of the data in BRAM cannot be predetermined. To finish the write-back operation of m_u within one cycle, we make a re-arrangement from m_u to m'_u to obtain a fixed BRAM partition mapping. Specifically, if u_i is odd, then m_u will be vertically mirrored. If v_i is odd, m_u will be horizontally mirrored. Then we write back the mirror matrix $m_{u'}$ according to write flags.

D. Streaming Architecture

The proposed architecture is shown in Fig. 5. It is mainly composed of four parts: *dynamic block manager*, *processing element* (PE), *local data*, and the *memory controller*. The *dynamic block manager* is

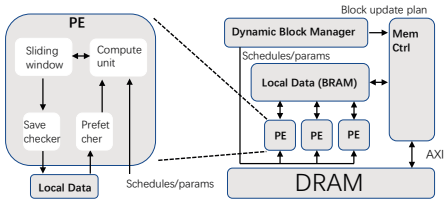

Fig. 5. Architecture overview.

designed for the following functions: (1) providing block update plan to guide the *memory controller* to update *local data*, and (2) providing work schedules and projection parameters for PEs.

The *memory controller* communicates with external DRAM to update the *local data* according to the plan provided by the *dynamic block manager*, including the update for the on-chip reconstruction block, and the maintenance of the on-chip projection sliding window shown in Fig. 2.

The *local data* module is composed of the partitioned projection block in BRAMs and the reconstruction block in a streaming buffer. The partition ensures that the PEs can do read/write with no conflicts.

The *processing element* is used for accumulating (forward projection) or updating (backward projection) the intensity/gradient of voxels. The *prefetcher* fetches $f/\mathcal{R}f$ data from the *local data* to ensure instant usage by the

compute unit. The temporary data is saved in the sliding window described in Fig. 4. The *save checker* saves any useless data that is still in the sliding window The PE is designed as such to significantly improve the data reuse and eliminate the latency of the *compute unit* data access.

IV. EXPERIMENT

To evaluate our design, the testing dataset that we choose is a $72\times$ conical-tilt projection series collected by FEI Tecnai 12 and 2048×2048 CCD Gatan camera with bin2. This means the projection image resolution for the experiments is $1024\times1024\times72$. The phantom size is set to $1024\times1024\times128$. The testing dataset is aligned using our automatic alignment algorithm implemented on a Xilinx ZCU102 platform, which consists of an UltraScale FPGA, quad Cortex-A53 processors, and 500MB DDR3. The Xilinx SDSoC Development Environment v2018.3 is used for source compilation. The configuration of our implementation is as follows: the number of PEs $n_{PE} = 24$, block size $S_b = 64$. Both the PE and the data mover frequencies are set to 299.97MHz. The fixed-point data type is set to 12bits for integer and 20bits for fraction according to the estimation of total operation count on single data. Admittedly, this data type leads to more logic consumption compared with prior work [16], in which the total word length ranges from 24bits to 29bits. But it provides sufficient precision during the whole process of calculation and convenience for data transfer. The resource utilization is listed in Table I.

TABLE I
RESOURCE UTILIZATION ON XILINX ZCU102.

Detail	DSP	BRAM	FF	LUT
Consumed	1476	1352	200,062	235,928
Available	2520	1824	548,160	274,080
Utilization	58%	74%	36%	86%

To evaluate the performance, we compare the implementation on Xilinx ZCU102, Nvidia Titan V, and a 16-threads OpenMP implementation on Intel Xeon CPU E5-2650 v3 platform. The performance data is shown in Table II. We plugged in a power meter to the FPGA platform to measure ZCU102's runtime power. The runtime power of the Titan V and that of the standalone CPU are estimated using the Nvidia-smi and s-tui tools. In the experiment, the design implemented on ZCU102 achieves a $44.5\times$ speed-up over the parallel CPU implementation, and $1.60\times$ speed-up and $7.8\times$ energy efficiency over the Titan V implementation. The bottleneck process $\nabla\mathcal{R}_\theta f$ achieves $1.74\times$ speed-up and $8.54\times$ energy efficiency over Titan V. We also evaluate implementations of fixed-points data type on CPU/GPU, where operations on fixed-point decimals are implemented by integer operations and bitshifts. Due to the design of the instruction set on modern CPU/GPUs, the execution efficiency of the float-point version is even faster than that of the fixed-point version.

Furthermore, we compare our subfunctions, (a) the forwards projection $\mathcal{R}f$ and (b) the backward projection, to existing works, using Giga Updates Per Second (GUPS). GUPS is a common metric for performance comparison of tomography related implementations evaluated by different datasets [16], [17]. Total updates can be calculated by $voxel\_num \times avg\_proj\_per\_voxel$. For forward projection and backward projection, it is $voxel\_num \times 72 \times 6.97 = 6.74E10$. For the calculation of $\nabla_\theta\mathcal{R}f$, it is $voxel\_num \times 72 \times 6.97 \times 4 \times 5 = 1.35E12$. The comparison with prior work, listed in Table III, shows that our design outperforms existing implementations of these subfunctions.

TABLE II
PERFORMANCE COMPARISON. TIMING FOR BOTH FPGA AND TITAN V INCLUDES THE DATA TRANSFER OVERHEAD.

Device	ZCU102	Titan V	CPU
Precision	32bits fixed	32 bits float	32bits float
$\mathcal{R}f$(s)	2.24	3.59	54.1
$\nabla\mathcal{R}_\theta f$(s)	11.95	20.85	645.1
Back-proj(s)	2.10	1.53	25.7
Summary(s)	16.3	26.0	725
Power(W)	24.1	118	49.0
Energy(J)	392.6	3064	3.552E4

TABLE III
GUPS COMPARISON WITH RELATED WORKS.THE GUPS APPROXIMATION IS FROM [16].

Ref	method	Device	GUPS(FP)	GUPS(BP)	GUPS($\nabla_\theta\mathcal{R}f$)
[14], 2008	FDK	Virtex-4	-	1.0	-
[15], 2012	SF	Virtex-5 LX155	0.9	-	-
[16], 2015	EM	Virtex-6 LX760	20.4	21.9	-
Our work	WBP	ZCU102	30.1	32.1	113

V. CONCLUSION

In this work, we propose an FPGA accelerator for the state-of-the-art tomographic alignment algorithm. We introduce a multi-ray access approach and an effective local data management strategy that makes use of BRAMs. They efficiently compensate for the data transfer latency with DRAM and make the best of data reuse. We then introduce an FPGA-based processing engine to improve data throughput. Besides, a streaming structure and multi-PE parallelization further improve the performance. Experiments show a $44.5\times$ speed-up for the state-of-the-art alignment on Xilinx ZCU102 over a 16-thread multicore CPU implementation, and a $1.60\times$ speed-up with $7.8\times$ energy reduction over an implementation on Nvidia Titan V. Our design for key processes in tomographic alignment also perform better than existing works. Moreover, with the development of hardwares, our design is capable of scaling with the performance of FPGAs with more powerful memories and more on-chip resources.

VI. ACKNOWLEDGMENT

This work is partly supported by National Natural Science Foundation of China (NSFC) under Grant No. 61520106004, No. 11961141007, and No. 61631001, Beijing Academy of Artificial Intelligence (BAAI), and State Grid Corporation of China under Grant No. 5500-201958484A-0-0-00 "The Research of Electric Power Artificial Intelligence Computing Components Based on Heterogeneous Instruction Set."

REFERENCES

[1] Solmon, C. Donald, "The X-ray transform," *Journal of Mathematical Analysis & Applications*, vol. 56, no. 1, pp. 61-83, 1976.

[2] R. Han, Z. Bao, X. Zeng, et al., "A marker-free automatic alignment method based on scale-invariant features." *Journal of Structural Biology* vol. 186, no. 1, pp. 167-180, 2014.

[3] T. van Leeuwen, S. Maretzke, J. K. Batenburg, "Automatic alignment for three-dimensional tomographic reconstruction," *Inverse Problems*, vol. 32, no. 2, 2018.

[4] C. Yang, G. Ng, A. Penczek, " Unified 3-D structure and projection orientation refinement using quasi-Newton algorithm," *Journal of structural biology*, vol. 149, no. 1, pp. 53-54, 2005.

[5] S. Wen, G. Luo, "An Analytical Method of Automatic Alignment for Electron Tomography," *Large-Scale Annotation of Biomedical Data and Expert Label Synthesis and Hardware Aware Learning for Medical Imaging and Computer Assisted Intervention*, 2019, pp. 106-114.

[6] D. N. Mastronarde, S. R. Held, "Automated tilt series alignment and tomographic reconstruction in IMOD," *Journal of structural biology*, vol. 192, no. 2, pp. 102-113, 2017.

[7] Q. Tian,L. G. Öfverstedt, Unit USSCB, "Semi-automatically aligned tilt images in electron tomography," *ICIIBMS*, 2017, pp. 71-75.

[8] P. Trampert, S. Bogachev, N. Marniok, et al., "Marker detection in electron tomography: a comparative study," *Microscopy and Microanalysis*, vol. 21, no. 6, pp. 1591-1601, 2015.

[9] H. Winkler, K. A. Taylor, "Accurate marker-free alignment with simultaneous geometry determination and reconstruction of tilt series in electron tomography," *Ultramicroscopy*, vol. 106, no. 3, pp. 240-254, 2006.

[10] K. Song, R. Comolli, M. Horowitz, "Removing high contrast artifacts via digital inpainting in cryo-electron tomography: an application of compressed sensing," *Journal of structural biology*, vol. 178, no. 2, pp. 108-120, 2012.

[11] M. Karuppasamy, F. Karimi Nejadasl, M. Vulovic, et al., "Radiation damage in single-particle cryo-electron microscopy: effects of dose and dose rate," *Journal of synchrotron radiation*, vol. 18, no. 3, pp. 398-412, 2011.

[12] D. Castaño-Díez, M. Scheffer, A. Al-Amoudi, et al., "Alignator: a GPU powered software package for robust fiducial-less alignment of cryo tilt-series," *Journal of structural biology*, vol. 170, no. 1, pp. 117-126, 2010.

[13] N. Bhattacharyya, "Trends in otolaryngologic utilization of computed tomography for sinonasal disorders,". *The Laryngoscope*, vol. 123, no. 8, pp. 1837-1839, 2013.

[14] N. Gac, S. P. Mancini, M. Desvignes, et al., "High speed 3D tomography on CPU, GPU, and FPGA," *EURASIP Journal on Embedded systems*, no. 5, 2008.

[15] J. K. Kim, J. A. Fessler, Z. Zhang, "Forward-projection architecture for fast iterative image reconstruction in X-ray CT," *IEEE Transactions on Signal Processing*, vol. 60, no. 10, pp. 5508-5518, 2012.

[16] Y. Choi, J. Cong, "Acceleration of EM-based 3D CT reconstruction using FPGA," *IEEE transactions on biomedical circuits and systems*, vol. 10, no. 3, pp. 754-767, 2015.

[17] H. Inoue, "Efficient tomographic reconstruction for commodity processors with limited memory bandwidth," *IEEE 13th International Symposium on Biomedical Imaging*, 2016, pp. 747-750.