FPGA Implementation of EM Algorithm for 3D CT Reconstruction

Young-kyu Choi, Jason Cong, and Di Wu
Computer Science Department
University of California, Los Angeles
Los Angeles, CA, 90095, USA
{ykchoi, cong, allwu}@cs.ucla.edu

***Abstract*—Although the expectation maximization (EM)-based 3D computed tomography (CT) reconstruction algorithm lowers radiation exposure, its long execution time hinders practical usage. To accelerate this process, we introduce a novel external memory bandwidth reduction strategy by reusing both the sinogram and the voxel intensity. Also, a customized computing engine based on field-programmable gate array (FPGA) is presented to increase the effective memory bandwidth. Experiments on actual patient data show that 85X speedup can be achieved over single-threaded CPU.**

***Keywords*-computed tomography; customized architecture; data reuse; field-programmable gate array; ray tracing**

I. Introduction

Computed tomography (CT) is a frequently used methodology that can produce 3D images of patients. However, there are serious concerns regarding the danger of high radiation exposure from CT scans. This has motivated the development of low-dose compressive sensing-based CT algorithms. Instead of traditional algorithms such as filtered back projection, iterative algorithms such as expectation maximization (EM) [4] are used to obtain a similar image quality with less radiation exposure.

However, lower radiation exposure is achieved at the cost of longer computation time. The problem is due to the iterative nature of EM. For example, literature reports 14 minutes and 1.52 hours of CPU computation to process $128\times128\times64$ and $736\times64\times500$ pixels of projection, respectively [2] [3]. This becomes more problematic with actual patient images, which require larger input data for accurate measurement. Our experiment, which uses $672\times72\times15022$ pixels of projection, requires 54 hours of computation on Intel Xeon 5138 CPU. Such long latency hinders practical usage of EM for time-critical emergencies.

Although using pipelining or parallelization techniques with GPU or FPGA can accelerate this process, a bottleneck is often reached where adding more computing elements no longer improves the performance [2]. The reason is that CT is a memory-bounded application where the performance is dominated by the system memory bandwidth [2]. As a result, previous literature concentrated on reducing the memory accesses [2] [3] [6]. However, the improvement was often limited. The potential for large amounts of speedup can be found in the repetitive access pattern of EM. For example, each voxel intensity is accessed 1,004 times per iteration, and each sinogram is accessed 831 times per iteration. This suggests that reusing the previously fetched data can potentially reduce a large amount of memory accesses.

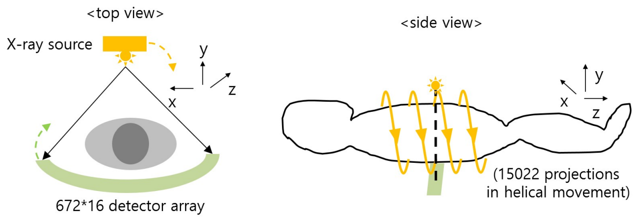

Figure 1. CT scan trajectory

The key contribution of this paper is that we propose an efficient method of reusing the data when performing EM for 3D CT reconstruction. We present a novel way of reusing both sinogram and intensity and increasing the reuse rate by 55X compared to the previous approach. Moreover, a customized FPGA architecture that efficiently supports the proposed algorithm and a strategy to increase the effective memory bandwidth is described in this paper.

II. Background

The objective of 3D CT reconstruction is to recover the object intensity from the CT scan measurements. A typical geometry of a CT scan is shown in Figure 1. When the source emits radiation, it penetrates the patient's body and is captured by an array of detector channels and detector rows. The source and the detector array proceed in a helical fashion, where the pair moves in a circular direction in the x-y plane and a lateral direction in the z plane.

EM is an iterative method of solving the above problem by progressively refining the object by finding the most likely intensity, given CT scan observation. From the forward projection, the projection of the currently estimated image, called sinogram, is obtained. This projection is compared to the observation of the CT scan. Based on the comparison, the previous estimate of the object intensity is updated using backward projection. These operations are repeatedly performed until the estimated object image converges. A more detailed explanation can be found in [2].

The problem of finding how much each voxel of an object will be projected on to the detector array of CT equipment can be solved using an efficient ray tracing method [7]. This algorithm exploits the repetitive nature to obtain the cross-sectional length of the currently traversed voxel and the

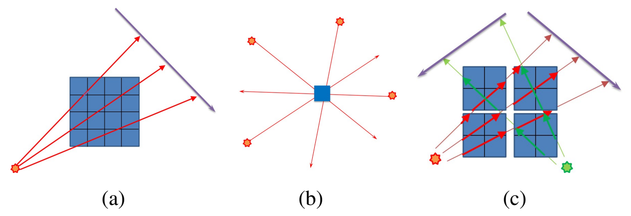

Figure 2. Various memory bandwidth reduction strategies: (a) ray-based, (b) voxel-based, (c) proposed

position of the next traversed voxel. With this information, the forward projection (1) and the backward projection (2) computation can be described as:

$$(\text{new sinogram}) = \Sigma(\text{voxel intensity}) * (\text{ray length}) \quad (1)$$

$$(\text{updated voxel intensity}) = \Sigma(\text{ray sinogram}) * (\text{ray length}) \quad (2)$$

The target platform is the Convey HC-1ex, which includes four Xilinx Virtex 6 LX760 FPGAs. This platform has been chosen because of the high coprocessor memory bandwidth (80GB/s) and efficient scatter-gather memory access capability [1] of the coprocessor memory controller.

III. Memory Access Reduction by Data Reuse

A. Reuse Approach

1) Ray-Based Reuse: When each ray passes through multiple voxels, as illustrated in Figure 2(a), the intermediate accumulated sinogram (forward projection) or sinogram for updating (backward projection) can be stored as a local variable and referenced without accessing the external storage. This strategy will be refered to as *ray-based reuse*. For our input dataset, each ray accesses an average of 1,004 voxels, which corresponds to a 1,004X reduction in memory transaction. [2] benefits from this type of reuse.

2) Voxel-Based Reuse: To further reduce the external memory access, we propose also reusing the intensity of the voxels – a strategy we call *voxel-based reuse*. After storing the intensity of a voxel in a local memory, this value is repeatedly updated with the sinogram of all rays that pass through that voxel. This is illustrated in Figure 2(b).

The voxel-based reuse rate is defined as the number of ray accesses per voxel. For analysis, the trajectory of the CT scan should be taken into account. The rays in the 672 channels $\times$ 16 detector rows access each voxel only 2.3 times on average. The reason for such a low reuse rate is that radiation emission spreads out in a cone shape, and very few rays access the same voxel far away from the source. On the other hand, there is almost a linear increase in the reuse rate across adjacent projections ($2.9 \rightarrow 5.7 \rightarrow 11.4$). The reason for this is that although there is a circular movement in a source-detector pair, the rays of nearby projections intersect almost identical x-y planes. The difference is in the angle of penetration, which does not severely change the actual voxels through which the rays pass. The voxel-based reuse rate among all projections is 831, which is adequate for large performance improvement.

However, the challenge in implementing voxel-based reuse is that there is no known efficient algorithm that can, on the fly, compute the list of rays that pass through a particular voxel. As a result, a huge list of rays that pass through each voxel must be read, which defeats the purpose of bandwidth reduction. Also, employing the voxel-based reuse scheme alone results in a loss of sinogram reuse.

3) Tiled Hybrid Reuse: We propose a hybrid approach, illustrated in Figure 2(c), that takes advantage of these two opposite reuse approaches. Similar to the voxel-based reuse case, we store voxel intensity in a local memory. To address the problem of having a huge list of rays for each voxel, we first propose storing only the range of rays that pass through them (*e.g., highest/lowest detector channel #, highest/lowest row #*). The reason for this is that the object of interest in a CT scan is mostly continuous and occupies a range of space, rather than being spread out randomly. Second, we propose storing a tile of nearby voxels in the internal memory. Since nearby voxels share the list of rays that pass through them, we can take advantage of the tendency for nearby voxels to access a similar set of sinograms. These two techniques reduce the overhead of storing the list of rays from 302GB to 58MB for a $64 \times 64 \times 1$ tile.

To further decrease the memory transaction, the tiled ray-based reuse technique is also employed. That is, we perform ray tracing inside a tile. The intermediate sinogram variable is stored in flip-flops while traversing in a ray. Note that compared to the original ray-based reuse scheme, the reuse rate is decreased because many tiles need to read a single sinogram value from DRAM.

Table I
Comparison of DRAM data transaction (per iteration)

		Ray Reuse	Voxel Reuse	Hybrid Reuse ($64 \times 64 \times 1$ tile)	
		DRAM	DRAM	DRAM	BRAM
Forward	Intensity	324	0.39	0.39	324
	Sinogram	0.65	649	11.23	0
Backward	Intensity	649	0.39	0.39	649
	Sinogram	0.65	324	5.62	0
Total Transfer (GB)		974	974	17.6	973
Available BW (GB/s)		80	80	80	204

4) Comparison: The DRAM data transaction comparisons among ray-based, voxel-based, and hybrid reuse schemes are shown in Table I. The ray-based reuse scheme requires large off-chip memory bandwidth to transfer intensity, whereas the voxel-based scheme requires a large sinogram transaction. Since voxels in a tile are reused, the hybrid reuse scheme transfers each voxel only once, similar to the voxel-based reuse scheme. Moreover, the sinogram is fetched approximately 17.4 times for a $64 \times 64 \times 1$ tile size. Considering the combined value of sinogram and intensity, there is a 55.3X reduction in external memory access.

Table I also shows that there is little difference between the three reuse schemes for a combined amount of internal

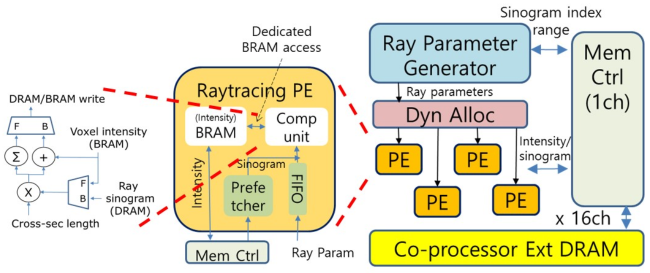

Figure 3. The architecture of the proposed system

and external memory access. However, we can still expect a high performance increase in the proposed scheme, because the customized architecture, which will be explained in Section IV, has a much higher internal memory throughput than the external memory.

B. *Tile Shape and Size Exploration*

The reuse rate presented in the previous section assumes that an infinte amount of internal memory is available. But in reality, there is only a limited amount. Thus, for practical implementation, two parameters need to be determined: tile shape and tile size.

Since the scan proceeds in a helical fashion, it is likely that adjacent rays will access similar (x,y) voxels with a rotated ray vector. Therefore, it is advantageous to use a flat surface tile rather than a cube tile to maximize reuse between adjacent rays. A tile size of 64×64×1 was selected to fit the amount of internal memory available (12.8MB) in the Convey HC-1ex platform. This tile size reduces the total data transfer from 974GB to 17.6GB.

IV. Architecture

The proposed architecture is shown in Figure 3. It is divided into three parts: ray parameter generator, ray tracing processing element (PE), and the external memory controller.

The *ray parameter generator* provides following parameters to PEs : ray index, ray vector, entry point, and the sinogram value. It also provides the coordinates of the processed tile. The list of tiles and the range of rays are read from the external memory.

The *ray tracing processing element (PE)* is responsible for accumulating (forward projection) or updating (backward projection) the intensity of voxels. Since both operations are based on the ray tracing algorithm, they are combined into one PE to save logic resources.

Since the target application is bounded by the memory bandwidth, we propose tightly coupling the ray tracing computation unit with the internal BRAM. As a result, every PE is guaranteed to have a one-cycle access to the memory. Moreover, the effective bandwidth increases from 80 GB/s to 284 GB/s due to the added internal memory bandwidth.

On average, 95 tiles are mapped to a single PE. Since the workload varies from tile to tile, a dynamic work allocator is used to distribute tiles to available PEs.

The *external memory controller* arbitrates access requests from the ray parameter generator and four ray tracing PEs in round-robin fashion. Note that such sharing became possible because the external memory traffic was reduced with a dedicated BRAM inside each PE.

V. Experimental Result

A. *Development Flow and Dataset Description*

The core IPs, such as PEs and parameter generators, are described in C and were transformed into RTL using the Vivado HLS 2013.1. The code was carefully optimized to achieve a throughput of 1. Some designs, such as the FIFOs, controllers, and glue logic, were hand-written in RTL. The Vivado-generated and hand-wirtten RTLs were synthesized, mapped, placed, and routed using a Convey development toolchain that is based on the Xilinx ISE.

To evaluate our algorithm, we used a chest CT dataset from actual patients at our institution using a protocol approved by our institution's review board. The scan was acquired using a helical cone beam protocol on a Siemens Somatom Sensation 16 scanner (1040mm focus-detector distance, 570 mm focus-isocenter distance). Two input data was used - M.12500 and S.21800, as described in Table II. Fifty iterations of the EM algorithm were performed.

Table II
Test Images

Image	Detector Array Size	Rotation Speed	# of Proj.	Reconstr. volume
M.12500	672×16	1,160 proj / rotation	15,022	512×512×372
S.21800			12,238	512×512×75

B. *Performance, Area, and Accuracy*

The execution time for Intel Xeon 5138 CPU, Tesla C1060 GPU, and Xilinx Virtex 6 LX760 FPGA is shown in Table III. It shows that the proposed architecture has 85X and 54X speedup over the single-thread CPU result. Even though the FPGA has a much smaller DRAM bandwidth and a 28X slower clock frequency than the CPU, the proposed system has a higher performance because of higher parallelism (256), data reuse optimization, and the customized circuitry. S.21800 has less speedup than M.12500, because the reuse rate was reduced due to smaller input data size.

Table III
Execution time comparison (unit: minutes, 50 iterations)

Platform	Image	Forward	Backward	Total
CPU	M.12500	1253	2025	3280
	S.21800	623	1039	1663
GPU	M.12500	19.8	90.6	110
	S.21800	15.8	68.3	84.1
FPGA (our work)	M.12500	19.4	19.4	38.8 (84.5X↑)
	S.21800	15.5	15.5	31.0 (53.6X↑)

For comparison, we also implemented a GPU version using the same parameters and the same input data as the FPGA version. We found that the proposed reuse scheme degrades the performance in GPU architectures, because the

Table V
COMPARISON WITH OTHER FPGA/GPU WORKS ON 3D CT RECONSTRUCTION

	Reference (Year)	Platform	Clock Freq	Test Image (Projection Size)	Algorithm	Speedup
GPU-based	[3] (2003)	GeForce 4 Ti 4600	300MHz	MCAT ($128\times128\times64$)	OS-EM	4.2X
	[6] (2008)	GeForce 8800 GTX	575MHz	Shepp-Logan ($512\times512\times360$)	FDK	12X
	[5] (2011)	GeForce GTX 280	602MHz	patient image ($1240\times960\times543$)	FDK	22X
FPGA-based	[5] (2011)	8 Virtex-4 SX35	200MHz	patient image ($1240\times960\times543$)	FDK	35.8X
	[2] (2012)	4 Virtex-6 LX760	150MHz	phantom image ($736\times64\times500$)	EM-TV	8.4X
	This paper (2014)	4 Virtex-6 LX760	75MHz	patient image ($672\times72\times15022$)	EM	85X

Table IV
RESOURCE CONSUMPTION PER FPGA

-	LUTs	FFs	DSP48s	BRAM18K
Total	331,173	284,691	416	1,072
Utilization	60%	34%	48%	74%

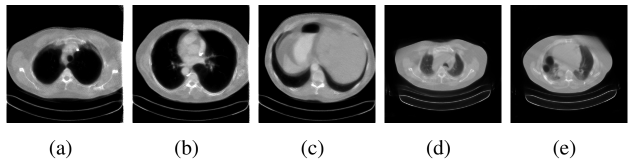
Figure 4. Reconstructed result image: M.12500 (a) slice 100 (b) slice 200 (c) slice 300; and S.21800 (d) slice 25 (e) slice 50

ray parameter generation has cosine operations that require long computation time in GPU architectures. Therefore, the GPU version only uses the ray-based reuse scheme, similar to [2]. Thus, despite having 8X slower clock frequency, FPGA has a performance advantage over GPU due to a higher reuse opportunity and the customized architecture.

The resource consumption is shown in Table IV. Placing multiple PEs and parameter generators led to a 60% utilization ratio of LUTs. This complicated placement and routing, and as a result, the clock frequency had to be reduced to 75MHz.

The original CPU software version has single-precision floating-point variables, whereas the FPGA version has variables with various bitwidth. Experiment shows that the FPGA approaches the floating-point image quality within 36dB after 50 iterations.

Some slices ($512\times512\times1$) of the final reconstructed image are shown in Figure 4.

C. Comparison With Other Work

Table V shows the comparison between our work and other FPGA and GPU implementations. Since direct comparison is difficult, the speedup factor over CPU is shown for relative comparison. It shows that this work has a higher speedup factor than any other works.

The closest work is [2], which uses the same algorithm and platform. The main difference is the reuse approach and the customized architecture, which accounts for about a 10X difference in performance. However, compared to the 55.3X improvement observed in Table I, the performance gain seems to be relatively small. This suggests that the DRAM bandwidth is now under-utilized, and DRAM bandwidth is no longer a performance-limiting factor.

A natural way of solving this problem is to add more PEs per memory channel. However, Table IV shows that there are not enough LUTs for more PEs. Thus, it can be inferred that the next step of improvement would be to find a suitable way of raising the utilization ratio of DRAM access and PEs, without using too many logic resources.

VI. CONCLUSIONS

This paper describes the acceleration of the EM algorithm on the Convey FPGA platform. A data reuse scheme based on tiling is proposed to reduce external memory bandwidth. A dedicated architecture suitable for the FPGA is presented to increase the system memory bandwidth. The experimental results show that the proposed system has a 85X speedup over single-thread CPU implementation. It also shows that the performance bottleneck has been changed to require more logic resource for PE implementation and to increase utilization ratio of PEs. This remains as future work.

ACKNOWLEDGMENT

The authors would like to thank Dr. Ming Yan and Dr. Yi Zou for sharing the CT reconstruction C code. They are also grateful to Dr. William Hsu for supplying the chest CT dataset. This work was supported by the Center for Domain-Specific Computing (CDSC) under the NSF Expeditions in Computing Award CCF-0926127.

REFERENCES

[1] J. Bakos. High-performance heterogeneous computing with the Convey HC-1. *IEEE Computing in Science and Engineering*, 12(6):80–87, 2010.

[2] J. Chen, J. Cong, L. A. Vese, J. Villasenor, M. Yan, and Y. Zou. A hybrid architecture for compressive sensing 3-D CT reconstruction. *IEEE J. Emerging and Selected Topics in Circuits and Syst.*, 2(3):616–625, 2012.

[3] K. Chidlow and T. Möller. Rapid emission tomography reconstruction. in *Proc. 2003 Eurographics/IEEE TVCG Workshop on Volume Graphics*, 15–26, 2003.

[4] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[5] H. Scherl, M. Kowarschik, H. Hofmann, B. Keck, and J. Hornegger. Evaluation of state-of-the-art hardware architectures for fast cone-beam CT reconstruction. *Parallel Computing*, 38:111–124, 2011.

[6] G. Yan, J. Tian, S. Zhu, Y. Dai, and C. Qin. Fast cone-beam CT image reconstruction using GPU hardware. *J. X-Ray Science and Technology*, 16:225–234, 2008.

[7] H. Zhao and A. J. Reader. Fast ray-tracing technique to calculate line integral paths in voxel arrays. in *IEEE Nucl. Sci. Symp. Conf. Record*, 4:2808–2812, 2003.