OUT-OF-CORE CONE BEAM RECONSTRUCTION USING MULTIPLE GPUS

Fumihiko Ino[†]*, Yusuke Okitsu*[†]*, Taketo Kishi*[‡]*, Syuhei Ohnishi*[‡]*, Kenichi Hagihara*[†]

[†]Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
[‡]Analytical and Measuring Instruments Division, Simadzu Corporation
1 Nishinokyo-Kuwabara-cho, Nakagyo-ku, Kyoto 604-8511, Japan

ABSTRACT

This paper presents a graphics processing unit (GPU) based method capable of accelerating cone-beam reconstruction of large volume data, which cannot be entirely stored in video memory. Our method accelerates the Feldkamp, Davis and Kress (FDK) algorithm in a multi-GPU environment. We present how the entire volume can be efficiently decomposed into small portions to reduce the usage of video memory on each graphics card. Experimental results are also presented to understand the reconstruction throughput on an nVIDIA Tesla S1070 server. It takes approximately three minutes to reconstruct a 2048^3-voxel volume from 720 2048^2-pixel projections. The effective bandwidth of video memory reaches 137 GB/s per GPU, demonstrating a higher utilization of texture caches.

1. INTRODUCTION

Cone beam (CB) reconstruction is an imaging technique for producing a three-dimensional (3-D) volume from a series of 2-D projections acquired by a CB computed tomography (CBCT) scan. This technique plays an important role in various fields, such as clinical diagnosis and nondestructive inspection of engineering products. For many years, CB reconstruction has been a compute-intensive task to us. However, the graphics processing unit (GPU) has emerged as a powerful accelerator that achieves a 30-fold speedup over a CPU-based method optimized using SSE instructions [1]. For example, a reconstruction task of a 512^3-voxel volume can be completed within several seconds using the commodity GPU.

However, most GPU-based methods [2–4] assume that the volume is small enough to store the entire data in video memory. This assumption restricts the volume size by 512^3 voxels on current commodity GPUs with at most 1 GB of memory. Though this limitation was not so serious problem, flat panel detectors in recent CBCT systems now have more than 1024^2 detector units. Since the data size reaches 4 GB for a 1024^3-voxel volume, an out-of-core algorithm is needed to reconstruct a large volume from high-resolution projections.

This work was partly supported by JSPS Grant-in-Aid for Scientific Research (A)(2)(20240002) and the Global COE Program "in silico medicine" at Osaka University.

In this paper, we present how out-of-core reconstruction can be efficiently realized for a large volume, which cannot be entirely stored in video memory. Our algorithm decomposes the volume into small portions, which are then processed using an in-core algorithm [2] in a multi-GPU environment. The in-core algorithm is based on the Feldkamp, Davis and Kress (FDK) algorithm [5], which is widely used in many CBCT systems. Our algorithm parallelizes the filtering stage and the backprojection stage, which are the time-consuming part of the FDK algorithm. The algorithm is implemented using a general-purpose programming framework, called the compute unified device architecture (CUDA) [6].

The outline of this paper is as follows. Section 2 introduces previous work. Section 3 then describes the out-of-core algorithm. Sections 4 and 5 present results and conclusions.

2. RELATED WORK

Many researchers [2–4] are trying to accelerate CB reconstruction on the GPU. Okitsu *et al.* [2] show that texture cache optimization is the key technique to accelerate the backprojection stage on the CUDA-compatible GPU. Scherl *et al.* [3] propose another optimization strategy that minimizes the resource usage to exploit higher parallelism on the GPU. In contrast to these CUDA-based implementations, Yan *et al.* [4] propose a graphics-based method that implements the FDK reconstruction using the OpenGL library. Their implementation takes only 5.2 seconds to reconstruct a 512^3-voxel volume from 360 512^2-pixel projections. The implementations mentioned above are based on in-core algorithms that requires the entire volume to be stored in video memory.

In contrast to these in-core algorithms, out-of-core algorithms can deal with larger data. Nesterets *et al.* [7] proposes an efficient algorithm that uses as little video memory for the reconstruction of each axial slice of the object as possible. Their algorithm carries out reconstruction in a slice-by-slice manner. It takes about 81 minutes on a GeForce GTX 260 card to reconstruct a 2048^3-voxel volume from 1440 2048^2-pixel projections. A similar work is also done by Nöel *et al.* [8]. Their implementation runs on a GeForce 8800 GT card and takes 61.3 seconds to reconstruct a 1024^3-voxel volume from 106 1024^3-pixel projections. Zhao *et al.* [9] im-

plement an out-of-core algorithm using the OpenGL library. It takes about 101.9 seconds on a Quadro FX 4600 card to reconstruct a 1024^3-voxel volume from 720 1024^2-pixel projections. In contrast to these out-of-core algorithms, we show that our algorithm runs more efficiently with reaching the theoretical performance bound.

3. METHODS

Consider a reconstruction task that produces an N^3-voxel volume V from a series of K M^2-pixel projections using C GPUs. The FDK algorithm [5] solves this problem by processing the filtering stage and the backprojection stage. We currently use the Shepp-Logan filter [10] at the filtering stage.

3.1. Parallelization

The data dependence at the filtering stage and at the backprojection stage can be summarized as follows.

Filtering stage: Different pixels on the same projection can be simultaneously filtered at this stage, because there is no data dependence between them. Furthermore, different projections can also be processed at the same time, because they do not have data dependence.

Backprojection stage: The backprojection stage must be processed after the filtering stage. However, different voxels can be simultaneously processed at the backprojection stage. In addition, the backprojection operator has an associativity property, so that projections can be processed in an arbitrary order. All projections must be backprojected into every voxel in the volume.

According to the analysis mentioned above, we have decided to decompose the volume data V into multiple subvolumes $V_1, V_2, \cdots, V_B$, where B (> 1), because the volume size usually limits the maximum reconstruction size. We determine the value of B such that (1) every subvolume is small enough to store in video memory and (2) every GPU has at least one assigned subvolume ($B \geq C$). After this decomposition, the in-core algorithm [2] can be applied to each subvolume of N^3/B voxels.

Figure 1 shows an overview of the proposed method, which exploits data parallelism in each stage. As shown in this figure, each of C GPUs is responsible for K/C projections at the filtering stage and for B/C subvolumes at the backprojection stage. Both stages are processed by the in-core algorithm [2], which performs the filtering of I projections at a time and performs the backprojection of I filtered projections into a subvolume at a time, where $1 \leq I \leq K$. Therefore, the algorithm iteratively invokes the GPU program to process all projections and subvolumes. The value of I is experimentally determined as $I = 5$.

Notice here that all filtered projections are sent back to main memory before the backprojection stage. This intends to gather projections and broadcast all of them to GPUs. The

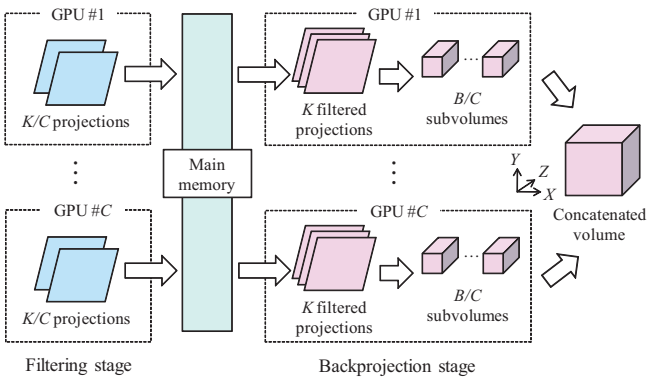

Fig. 1. Overview of proposed out-of-core algorithm.

Input: Projections $P_1, P_2, \ldots, P_K$, number B of subvolumes,
Output: Volume V

Algorithm Cone_Beam_Reconstruction()
1: Load $P_1, P_2, \ldots, P_K$ from storage device;
2: **for** $k = 1$ **to** K **do in parallel**
3: Download P_k from main memory to video memory;
4: $Q_k \leftarrow$ Filtering(P_k); // Q_k: k-th filtered projection
5: Readback Q_k to main memory;
6: **endfor**
7: **for** $b = 1$ **to** B **do in parallel**
8: Initialize subvolume V_b in video memory;
9: **for** $k = 1$ **to** K **do**
10: Download Q_k from main memory to video memory;
11: Bind Q_k as texture;
12: $V_b \leftarrow$ Backprojection(Q_k, k, b);
13: **endfor**
14: Readback subvolume V_b to main memory;
15: **endfor**

Fig. 2. Pseudocode of proposed algorithm. The actual code is optimized to process the filtering and backprojection of I projections at a time.

reason why we gather filtered projections is that all K filtered projections are needed at the backprojection stage on every GPU, though each GPU is responsible for a part of the volume. Thus, our algorithm requires data transfer but prevents us from filtering all of K projections on every GPU. After this parallel backprojection, B subvolumes are then concatenated into the final volume.

Figure 2 shows a pseudocode of the proposed algorithm. In this code, "download" represents the data transfer from main memory to video memory. Similarly, "readback" represents the transfer in the opposite direction. Lines 2–6 and 7–15 correspond to the filtering stage and the backprojection stage, respectively. In the backprojection stage, filtered projections are stored in textures [6] to use texture units for interpolation of pixels on projections. Since texture units are separated from processing elements in the GPU, using textures is necessary to maximize the entire performance by offloading workloads from processing elements to texture units.

Note here that the code is optimized using an asynchronous execution technique [6]. This technique is applied

Table 1. Analytical results of time complexity and space complexity. Parameter I represents the number of projections processed at a time. We currently use $I = 5$.

Item	Original [2]	Proposed
Download	$O(KM^2)$	$O(BKM^2/C)$
Filtering	$O(KM^2)$	$O(KM^2/C)$
Readback (projection)	—	$O(KN^3/C)$
Backprojection	$O(KN^3)$	$O(KN^3/C)$
Readback (volume)	$O(N^3)$	$O(N^3/C)$
Concatenation	—	$O(B)$
Video memory usage	$O(N^3 + IM^2)$	$O(N^3/B + IM^2)$
Main memory usage	$O(N^3 + KM^2)$	$O(N^3 + KM^2)$

to the filtering stage to overlap data transfer (lines 3 and 5) with GPU computation (line 4). In contrast, we cannot apply this technique to the backprojection stage because textures currently have to be sent in synchronous mode.

Finally, we explain how our algorithm decomposes the volume. As shown in Fig. 1, our algorithm uses a 2-D block decomposition scheme that separates the volume space with respect to the (x, y) coordinate. In other words, this scheme does not divide the Z (axial) space, so that any voxel $V(x, y, z)$ located on the same (x, y) coordinate will be assigned to the same GPU. This allows us to maximize the effect of a data reuse technique [2], which reduces the amount of computation between different axial slices. Thus, data decomposition depends on the data reuse technique because data cannot be reused between different GPUs.

3.2. Analytical Analysis

Table 1 shows an analytical analysis of the proposed algorithm and the original in-core algorithm [2] in terms of time complexity and space complexity. As shown in this table, data decomposition allows us to reduce the space complexity so that out-of-core reconstruction can be realized by selecting the appropriate value for B.

With respect to the time complexity, our algorithm basically reduces the original complexity by a factor of $1/C$ because it fully exploits data parallelism using C GPUs. However, the time complexity of data download is the exception. This can be explained by the pseudocode in Fig. 2, which has a nested loop at line 9. The loop implies that the same projections can be iteratively sent to video memory for different subvolumes. Though this redundancy increases the complexity by a factor of B, it saves the usage of video memory to deal with large volume data.

Note here that Table 1 shows results for per GPU. Therefore, the measured results can differ from them. For example, we currently cannot send data to multiple GPUs at the same time. This is due to the graphics driver, which sequentially sends the data in a first-in, first-out (FIFO) manner. Furthermore, our algorithm is optimized using stream, which overlaps data transfer with program execution on the GPU.

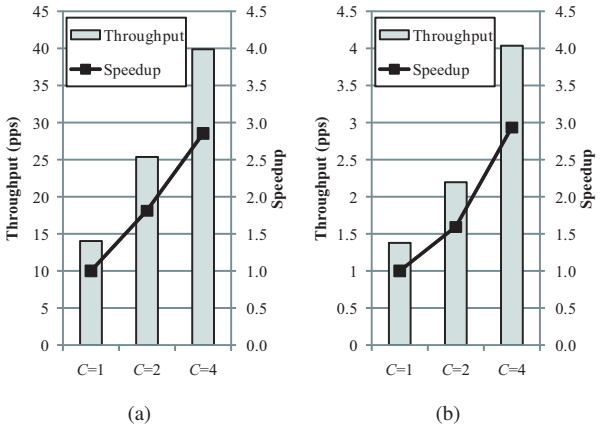

Fig. 3. Throughput and speedup ratio (a) for $N = M = 1024$ and (b) for $N = M = 2048$. The throughput is presented in terms of the number of projections per second (pps).

Table 2. Breakdown of execution time T in seconds.

Breakdown	$N = M = 1024$			$N = M = 2048$		
	$C = 1$	$C = 2$	$C = 4$	$C = 1$	$C = 2$	$C = 4$
Initialization	1.3	2.7	5.4	6.1	5.5	7.5
Download	4.4	2.4	1.3	255.8	190.7	103.6
Filtering	5.2	2.6	1.3	20.8	10.4	5.2
Readback (prj.)	1.1	0.6	0.3	3.8	3.5	1.6
Backprojection	39.7	19.9	10.0	235.3	117.5	59.3
Readback (vol.)	1.3	0.9	0.6	10.2	6.3	3.6
T (no stream)	53.0	29.1	18.9	532.0	333.9	180.8
T (stream)	51.5	28.4	18.0	523.8	328.3	178.4

4. RESULTS

We now show experimental results to evaluate the performance of the proposed method. A reconstruction of 1024^3/2048^3-voxel volume from 720 1024^2/2048^2-pixel projections is carried out on an nVIDIA Tesla S1070 server ($K = 720$). The 1024^3-voxel data and 2048^3-voxel data are decomposed into 4 portions ($B = 4$) and 64 portions ($B = 64$), respectively. The server is connected to a PC that has a Xeon E5450 CPU and 16-GB main memory. It runs on CentOS 5.3 with CUDA 2.2 [6] and driver version 185.18.08.

Figure 3 shows the reconstruction throughput in terms of the number of projections per second (pps). The throughput reaches 4 pps when using all four GPUs for $N = 2048$ ($C = 4$). On a single GPU, the throughput for 1024^3-voxel data and that for 2048^3-voxel data are 14 pps and 1.4 pps, respectively. The speedup over a single GPU version is around a factor of 3, which is slightly lower than the optimal speedup. This inefficiency can be explained by Table 2, which shows the breakdown of execution time T for each execution configuration. Although the filtering stage and the backprojection stage are linearly accelerated using multiple GPUs, the GPU initialization and data transfer emerge as performance bottle-

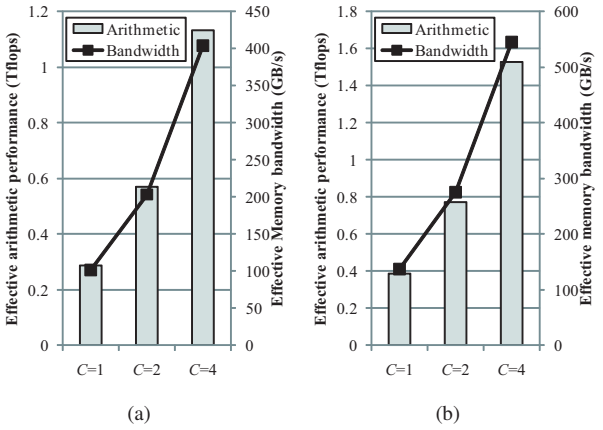

Fig. 4. Effective arithmetic performance and memory bandwidth of backprojection measured (a) for $N = M = 1024$ and (b) for $N = M = 2048$.

neck as we increase the number of GPUs. This might be due to the limitation of current graphics drivers, which are not fully multithreaded yet. Thus, GPUs are serially initialized in our implementation. Similarly, graphics drivers currently cannot simultaneously exchange data with multiple GPUs.

Figure 4 shows the effective arithmetic performance and memory bandwidth at the backprojection stage. The theoretical bound of our GPU is 102 GB/s and 0.933 Tflops (tera floating point number operations per second) for the memory bandwidth and for the arithmetic performance, respectively. As shown in this figure, we find that the effective bandwidth for $N = M = 2048$ reaches 137 GB/s per GPU, which exceeds the theoretical bound. This can be explained by texture caches, which saves the bandwidth of video memory. In contrast, the arithmetic performance is lower than 0.4 Tflops, which is equivalent to 41% efficiency. This lower efficiency indicates that processing elements in the GPU have to wait for data to be fetched from video memory. Therefore, we have to further increase the effective bandwidth to improve the effective arithmetic performance.

Finally, Table 3 shows a performance comparison with previous algorithms [7–9]. Since different GPUs are used in experiments, we have normalized the performance according to the bandwidth of video memory. The proposed algorithm achieves the highest efficiency among known algorithms.

5. CONCLUSION

We have presented an out-of-core reconstruction algorithm for large volume data. The algorithm efficiently decomposes the volume data into small portions. It also exploits data parallelism in the FDK algorithm to maximize the performance in a multi-GPU environment. In experiments, our algorithm takes about 3 minutes to reconstruct a 2048^2-voxel volume from 720 2048^2-pixel projections. We also show that the measured bandwidth exceeds the theoretical bandwidth for the sake of texture cache utilization. Therefore, we think that the measured performance is close to the theoretical bound. In the future, we plan to further minimize the amount of data transfer between video memory and main memory.

Table 3. Performance comparison with previous work.

Work	Throughput (pps) Normalized (Measured)	GPU specification Bandwidth (GB/s)	GPU specification Arithmetic (Tflops)
[8]	3.9 (1.7)	44.8	0.504
[9]	10.8 (7.1)	67.2	0.336
This paper	14.0 (14.0)	102.0	0.933
[7]	0.3 (0.3)	111.9	0.715
This paper	1.4 (1.4)	102.0	0.933

6. REFERENCES

[1] Marc Kachelrieß, Michael Knaup, and Olivier Bockenbach, "Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware," *Medical Physics*, vol. 34, no. 4, pp. 1474–1486, Apr. 2007.

[2] Yusuke Okitsu, Fumihiko Ino, and Kenichi Hagihara, "Fast cone beam reconstruction using the CUDA-enabled GPU," in *Proc. 15th Int'l Conf. High Performance Computing (HiPC'08)*, Dec. 2008, pp. 108–119.

[3] Holger Scherl, Benjamin Keck, Markus Kowarschik, and Joachim Hornegger, "Fast GPU-based CT reconstruction using the common unified device architecture (CUDA)," in *Proc. Nuclear Science Symp. and Medical Imaging Conf. (NSS/MIC'07)*, Oct. 2007, pp. 4464–4466.

[4] Guorui Yan, Jie Tian, Shouping Zhu, Yakang Dai, and Chenghu Qin, "Fast cone-beam CT image reconstruction using GPU hardware," *J. X-Ray Science and Technology*, vol. 16, no. 4, pp. 225–234, Oct. 2008.

[5] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," *J. Optical Society of America*, vol. 1, no. 6, pp. 612–619, June 1984.

[6] nVIDIA Corporation, "CUDA Programming Guide Version 2.2," April 2009, http://developer.nvidia.com/cuda/.

[7] Ya. I. Nesterets and T. E. Gureyev, "High-performance tomographic reconstruction using graphics processing units," in *Proc. 18th World IMACS/MODSIM Congress Modeling and Simulation (MODSIM'08)*, July 2009, 7 pages.

[8] Peter B. Noël, Alan M. Walczak, Kenneth R. Hoffmann, Jinhui Xu, Jason J. Corso, and Sebastian Schafer, "Clinical evaluation of GPU-based cone beam computed tomography," in *Proc. High-Performance Medical Image Computing and Computer Aided Intervention (HP-MICCAI'08)*, Sept. 2008.

[9] Xing Zhao, Jing jing Hu, and Peng Zhang, "GPU-based 3D cone-beam CT reconstruction for large data volume," *Int'l J. Biomedical Imaging*, Article ID 149079, 2009, 8 pages.

[10] L. A. Shepp and B. F. Logan, "The fourier reconstruction of a head section," *IEEE Trans. Nuclear Science*, vol. 21, no. 3, pp. 21–43, June 1974.