Ultramicroscopy 111 (2011) 1137–1143

Contents lists available at ScienceDirect

Ultramicroscopy

journal homepage: www.elsevier.com/locate/ultramic

A distributed multi-GPU system for high speed electron microscopic tomographic reconstruction

Shawn Q. Zheng, Eric Branlund, Bettina Kesthelyi, Michael B. Braunfeld, Yifan Cheng, John W. Sedat, David A. Agard *

The Howard Hughes Medical Institute and the W.M. Keck Advanced Microscopy Laboratory, Department of Biochemistry and Biophysics, University of California, San Francisco, 600, 16th Street, Room S412D, CA 94158-2517, USA

ARTICLE INFO

Article history:
Received 27 April 2010
Received in revised form
3 January 2011
Accepted 19 March 2011
Available online 1 April 2011

Keywords:
Tomography
Reconstruction
Parallel computing
Image processing

ABSTRACT

Full resolution electron microscopic tomographic (EMT) reconstruction of large-scale tilt series requires significant computing power. The desire to perform multiple cycles of iterative reconstruction and realignment dramatically increases the pressing need to improve reconstruction performance. This has motivated us to develop a distributed multi-GPU (graphics processing unit) system to provide the required computing power for rapid constrained, iterative reconstructions of very large three-dimensional (3D) volumes. The participating GPUs reconstruct segments of the volume in parallel, and subsequently, the segments are assembled to form the complete 3D volume. Owing to its power and versatility, the CUDA (NVIDIA, USA) platform was selected for GPU implementation of the EMT reconstruction. For a system containing 10 GPUs provided by 5 GTX295 cards, 10 cycles of SIRT reconstruction for a tomogram of $4096^2 \times 512$ voxels from an input tilt series containing 122 projection images of 4096^2 pixels (single precision float) takes a total of 1845 s of which 1032 s are for computation with the remainder being the system overhead. The same system takes only 39 s total to reconstruct $1024^2 \times 256$ voxels from 122 1024^2 pixel projections. While the system overhead is non-trivial, performance analysis indicates that adding extra GPUs to the system would lead to steadily enhanced overall performance. Therefore, this system can be easily expanded to generate superior computing power for very large tomographic reconstructions and especially to empower iterative cycles of reconstruction and realignment.

1. Introduction

Electron microscopic tomography (EMT), in particular cryo EMT, plays an important role in elucidating the three-dimensional (3D) biological structures of complex cellular machinery. The strong motivation to acquire numerous tomographic tilt series stems from the practical challenges that exist throughout cryo EMT and make the preparation for collecting data a lengthy process. Therefore, each grid must be thoroughly scanned and as many data sets collected as possible. In addition, concerns of structural heterogeneity also entail collecting multiple data sets in order to profile the structural variation. For single particle EMT, the goal of averaging a very large number of sub-tomograms also necessitates collecting and reconstructing many tomograms. Such demands have fueled the development of multi-scale automated EMT data collection systems [1,2]. With the capability to quickly acquire numerous EMT tilt series, efficient computation for data alignment and reconstruction then becomes critical. Our previous effort aiming at real-time data analysis was to perform a weighted back-projection reconstruction simultaneously while the data collection was in progress [3]. This concurrent reconstruction generates a 3D volume of moderate resolution at the end of each data collection. These volumes are screened for more promising tilt series that will be reconstructed later at full resolution. While this screening process reduces significantly the number of tilt series for final reconstruction, the remaining data sets are still an overwhelming load for reconstruction, especially given that it is now common place to generate 3D volumes of $2048^2 \times 512$ voxels (8 gigabytes) from a tilt series of 141 projection images of 2048×2048 pixels. Undoubtedly, future work will demand even larger reconstructions using unbinned data from 4096^2 or even 8192^2 cameras yielding reconstructions of 32 or 128 gigabytes (GBs). Besides the increase in size, efforts to improve the resolution by iterative cycles of reconstruction [4–7] and realignment of the observed projections (via comparison to projections calculated from the reconstruction) [8] greatly expand the number of reconstructions to be done. Such unrivaled increases in required data processing cannot be met by the current generation of CPU based computing.

The advent and fast evolution of programmable graphics processing units (GPUs), driven by the incessant demand from the gaming industry for more computing power, has profoundly changed how we

* Corresponding author. Tel.: +1 415 476 2521; fax: +1 415 476 1902.
E-mail address: agard@msg.ucsf.edu (D.A. Agard).

0304-3991/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ultramic.2011.03.015

envisage supercomputing systems and led to the invention of a new term, personal supercomputing, to describe the GPU-based computing technology. Characterized by the highly parallel, multi-threaded, and many-cored architectures, GPUs can generate tremendous computational horsepower and have a very high memory bandwidth. In addition, the compact design allows 4 GPU cards to fit into a personal computer. For example, a commercially available personal computer with 4 Tesla C1060 GPUs can provide nearly 4 TFLOPs of peak single precision floating point calculations and a total 16 GB memory at a remarkably affordable price point ($\sim$$8000). This highly cost effective expansion of computing power has also stimulated significant interests in the EMT field. Diez et al. [9] implemented popular reconstruction algorithms, including weighted back-projection, simultaneous iterative reconstruction technique (SIRT) and simultaneous algebraic reconstruction technique (SART), in a shading language for GPUs. Because of its versatility and improved debugging capabilities, these algorithms were re-implemented using the NVIDIA CUDA coding language [10]. Another group has made significant improvements in the algorithm implementation and developed codes for ordered subsets reconstruction on GPUs and a strategy for optimizing the number of subsets [11,12]. More recently, Vazquez et al. [13] implemented a variation of weighted back-projection from a matrix perspective on GPUs. With the ability to house multiple GPUs within a single computing node led Jang et al. [14] to develop a multi-GPU implementation of iterative reconstruction algorithms for computed tomography.

Driven by the need for speedy reconstructions of very large EMT volumes that are overwhelming to a single computing node, a cluster of computing nodes may be used to expand both the computing power and memory. Fernandez et al. [6] implemented the block-iterative component averaging (BICAV) and component-averaged row projections algorithms on a PC cluster consisting of 16 Pentium IV processors . We also routinely use a small linux cluster to perform real-time reconstruction concurrently with EMT data collection [3]. Given the tremendous computing power provided by GPU at very affordable price, we have endeavored to build a distributed supercomputing system that integrates many GPU equipped nodes to simultaneously perform EMT reconstructions. These nodes may be clustered as a centralized computing server or scattered in different locations but connected via high-speed network. The latter configuration provides an opportunity for various groups to share their computing resources and thus gain more computing power without extra expense. Each computing node reconstructs only a portion of the volume, and its size is determined dynamically based upon the total GPU memory that each node can provide. This allows the computing load to be evenly distributed over the participating nodes such that the entire reconstruction is not throttled by an overloaded node. The reconstructed subsets are then assembled together to form the complete volume. The CUDA platform, because of its much improved flexibility in developing complicated GPU kernel functions, was chosen to implement the SIRT, SART, and EM algorithms. This distributed GPU supercomputing system is linked via socket connection to our well established image processing software suite, Priism [15], which provides a convenient graphical user interface to the end users who specify which computing nodes will be used in the reconstruction and the locations of the input tilt series and the generated volumes. Upon completion of the reconstruction, the generated volume can be readily loaded into Priism for review, providing a streamlined process from volume reconstruction to visualization.

2. Iterative reconstruction algorithms

Iterative algorithms are often used to reconstruct 3D volumes from EMT tilt sets to minimize the deleterious effects of limited angular range and often very significant statistical noise. While there exist many variations [6,16–19], the basic idea is the same. The computation iteratively performs the steps of projecting the reconstruction, comparing the projection result to the measurement, and a backward projection of the corrections until the difference between the measurement and the projections of the reconstruction are sufficiently small. For our implementation of SIRT and SART, we use the algorithm of Xu et al. [12], shown in Eq. (1), to compute the value of a volume element, $v_j^{(k+1)}$, in the iteration, $k+1$, from the values, $v_j^{(k)}$, of the volume elements (voxels) in the previous iteration.

$$v_i^{(k+1)} = v_i^{(k)} + \frac{\lambda}{S}\sum_j w_{ij}\frac{\overline{r}_j - p_j}{q_j} \tag{1}$$

where S is the number (one for SART, and all of the projections for SIRT) of projections used in this iteration, and λ is an adjustable relaxation factor. The sum runs over all elements in the projections considered. w_{ij} is the interpolation weight for contribution of the ith voxel to the jth projection element; q_j, as shown in Eq. (2), is the sum of those weights for the jth projection element.

$$q_j = \sum_i w_{ij} \tag{2}$$

r_j is the measurement, adjusted as in Xu et al. [12], to compensate for the reconstruction not encompassing the entire object. Under the assumptions that the object is a slab of height, H, and that the projection holding the jth element was formed by parallel rays that make an angle, α, with the vertical axis of the slab, Eq. (3) relates the adjusted measurement value for the jth ray to the raw value.

$$\overline{r}_j = (r_j \cos\alpha) q_j / H \tag{3}$$

p_j is an element of the series of projections, as given in Eq. (4), of the reconstruction. M is the number of voxels in the reconstruction, and N is the number of elements in the subset of the projections used.

$$p_j = \sum_{i=1}^{M} v_i w_{ij} \quad \text{for } 1 \le j \le N \tag{4}$$

3. System design

The forward-projection defined in Eq. (4) is a pixel-wise computation and thus can be simultaneously performed on each pixel, as can the correction update prescribed in Eq. (1). Although the back-projection given in Eq. (1) involves integration along rays originating from various pixels, the computation performed on each voxel is completely independent of all other voxels. Therefore, parallelization can be achieved from the voxel perspective. It should be noted that the parallelization, however, is constrained by the GPU memory that in general cannot host all the elements (voxels or pixels) involved in the reconstruction. As estimated earlier, reconstructing a volume of $2048^2 \times 512$ voxels from a tilt series of $2048^2 \times 141$ pixels requires as least 9 gigabytes of memory, while even a large-memory GPU such as a Tesla C1060 (NVIDIA) provides only 4 gigabytes of memory, while the more cost-effective desktop GPUS have even less memory. Thus, another level of parallelization must be implemented to distribute the computation evenly on all available GPUs. Under the assumption of a single axis tilt geometry and parallel beam projection, it is quite standard to split volume into pieces along the axis (here taken to be the y axis) parallel to the tilt axis and have each piece reconstructed concurrently and then assembled to form the complete volume.

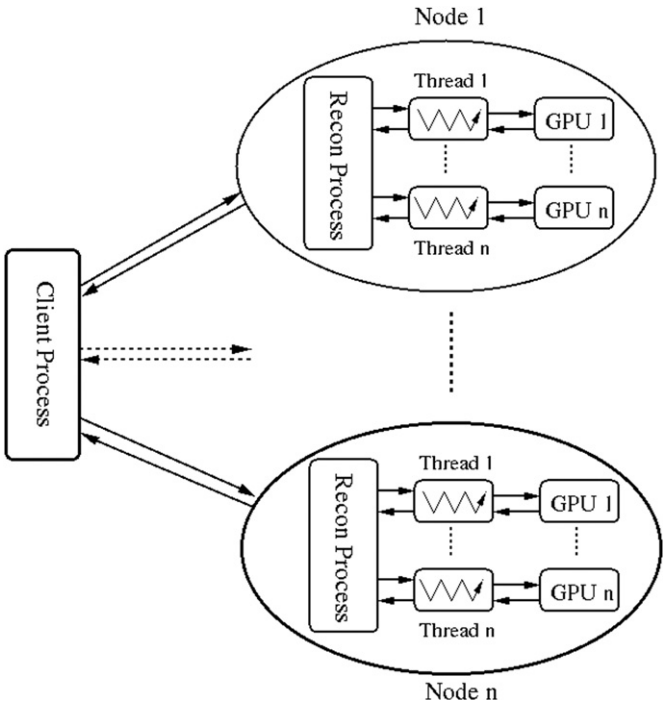

Fig. 1. Architectural schema of the distributed multi-GPU system.

Because of limits on the number of GPUs that can be used in one computer (for instance up to 4 T GPUs within a high-end personal computer), we developed a distributed system that brings together multiple GPU equipped computers (also referred to as nodes) to perform EMT reconstruction. Fig. 1 presents the schematic of the system architecture. The entire system is formed by a single host running the client process and multiple nodes of each executing the reconstruction process independently. The client host is linked to the nodes via high speed network, ideally 10 GB Ethernet or infiniband (although only gigabit Ethernet was used here). For the reconstruction no communication is required between nodes since the sub-volume reconstruction performed on each node is completely independent. End users can specify which nodes are utilized for the reconstruction depending on the input and output data sizes. Each participating node communicates with the client host via a TCP socket. Prior to reconstruction, the client process queries each node for the total GPU memory available for reconstruction. The available GPU memory determines the size of the sub-volume and thus the y subset of the tilt series required for reconstruction on the corresponding node. Note that the client process is responsible for loading and dividing the tilt series. It then distributes y subsets to these nodes and waits until each node completes the reconstruction and sends back the generated sub-volume. At the end, the complete volume is assembled by the client process, which is then ready for the next round of reconstruction.

Upon receiving the assigned y subset, the reconstruction process running on each node, further divides it evenly into smaller partitions based upon the number of GPUs installed, and each GPU computes only one partition of the sub-volume. To enable concurrent reconstruction on each GPU, the reconstruction process spawns the same number of CPU threads, each passes its allotted data to the corresponding GPU and launches the GPU computing kernels. Upon completion of the reconstruction, each CPU thread copies the sub-volume partition from the GPU into CPU memory and exits after the GPU resource has been released. Note that each CPU thread puts the sub-volume partition into the specific memory segment according to its y coordinate in the sub-volume. Therefore, when all CPU threads complete their executions, the sub-volume has been put together and is ready to be sent back to the client process for final assemblage. This system seamlessly handles the case where the input tilt series and volume cannot fit into the GPU memory on all the nodes. A portion of the volume is computed at a time, and the process is repeated until the full volume has been covered.

4. Implementation of kernel functions

The C-like programming structure makes CUDA very flexible and therefore quite appealing for implementing complicated programming logic. In addition, the unique 3D thread grouping and indexing allow straightforward access of data from multiple memory spaces for CUDA threads. This memory accessing flexibility is of particular importance since back-projection at each voxel involves contributions from each projection while forward-projection at each pixel needs to integrate the intensity of each voxel along the ray trace. Therefore, these two kernels need to read many scattered locations in global memory to retrieve the projection data for back-projection or voxel data for forward-projection. However, owing to hardware limitations, accessing the global memory (the largest on-card memory) incurs significant latency. According to NVIDIA [20], when there are no bank conflicts between threads, accessing the on-chip shared memory requires no extra clock cycle per instruction whereas reading and writing global memory consumes 400 cycles or more. While this latency can be effectively overlapped with GPU computations that manipulate only a few global memory locations, such overlap is not possible when so many scattered memory locations are accessed as within the back- and forward-projections. Fortunately, the read-only texture memory, although residing in the global memory space, is backed with an on-chip cache. Since the texture cache and registers are both on-chip, a cache-hit memory read is expected to be as fast as reading from a register while missing the cache is equivalent to accessing the global memory. Our implementation stores the input data for the kernels in texture memory to take advantage of the potential speedup of two orders of magnitude when the accessed data is in cache. Due to the read-only constraint of the texture memory, the output data must still be placed in a separate area in the global memory. When the output data from one operation serves as the input to the next, a CUDA memory copy is invoked to transfer these data to the location in the global memory designated as the texture memory. Although this strategy involves copying data between two locations in the global memory, the huge bandwidth for device memory copy dramatically improves performance compared to using global memory only. In addition to more efficient memory reads, the texture hardware also provides free linear interpolation that is much more efficient than a software implementation. Given the intensive linear interpolation involved in the operations of forward- and back-projection, the performance can be further improved using the hardware implementation of linear interpolation. However, it should be noted that the benefit of using texture memory is application-specific and choosing between global or texture memory should be tailored to the computation involved. Because our kernel functions write their results to global memory, we implemented memory coalescing techniques, as recommended by NVIDIA [20], to enhance the overall performance. This improvement originates from the capability of the CUDA driver to group multiple contiguous global memory accesses in a single transaction.

The major steps involved in the GPU implementation of SIRT reconstruction are presented in Fig. 2. The computations given in the solid boxes are implemented as CUDA kernels. Note that the

reconstruction proceeds slice by slice along the y direction on the GPU instead of being performed simultaneously over the entire volume. This weakened parallelization is due to our choice of 2D textures for the calculation. CUDA 3D textures could be used, but because they are limited to 2048 texels (elements in a texture) in any dimension, reconstructions with more than 2048 elements in the x or z dimensions would require that the algorithm partition the reconstruction in x or z.

The first kernel computes q_j given in Eq. (2). This kernel is launched only once since q_j does not change with respect to y. Once completed, q_j is placed into the texture memory for subsequent computation as defined in Eq. (1). The second kernel function calculates the normalized projection $\bar{r}_j$ shown in Eq. (3) and needs to be launched for each y slice (sinogram) of the measured projections but outside the iteration loop for the correction update. When the execution enters into the iteration loop, the forward-projection kernel is invoked to determine p_j for the volume with non-zero initial values (pre-loaded into the texture memory); otherwise this step is bypassed with p_j initialized with zeros. With $\bar{r}_j$, p_j, and q_j all in the GPU memory, the correction-update kernel is simply a pixel-wise calculation of $(\bar{r}_j - p_j)/q_j$. When completed, these results are copied from the global memory into the texture memory prior to invoking the back-projection kernel. When the back-projection kernel ends its execution, the reconstructed volume slice is then transferred into the texture memory for the next round iteration. These steps loop until the total number of iterations have been performed. The volume slice, instead of being placed into the texture memory, is copied back into the CPU memory. The next sinogram is then loaded into the GPU memory for reconstruction of the volume slice at the same y coordinate. The y-loop continues until the last sinogram has been reconstructed.

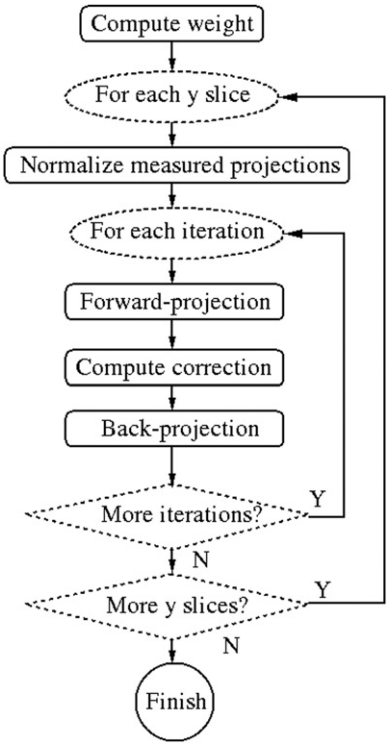

Fig. 2. Flow diagram of the GPU-based implementation of SIRT reconstruction. The operations listed in the solid boxes are implemented as CUDA kernel functions.

5. System verification and performance tests

Prior to testing the system performance, the 2D Barbara test image was chosen to verify our implementation since this image is quite often used for system verification [9,11]. Fig. 3 shows the reconstructions of the Barbara data set that contains 141 projections in the range of $\pm 70^\circ$ with the angular step of 1°, where Fig. 3(a) is the original image and Fig. 3(b) and (c) are 100 iterations of SIRT and EM reconstructions, respectively. While the overall quality looks satisfactory, it is worth pointing out that the smearing around the mouth is due to the nature of limit-angle reconstruction since the projections outside the ranges of $\pm 70^\circ$ are missing, a typical practical challenge encountered in EMT reconstruction.

We then reconstructed a 3D volume of $2048^2 \times 256$ voxels from a real tilt series to further verify the quality of reconstruction. This tilt series contains 122 projection images of 2048^2 pixels (4 bytes floating point per pixel) acquired every 1° within $\pm 60^\circ$ for a *Drosophila centrosome* with microtubules re-grown with bovine alpha/beta-tubulin and frozen in vitreous ice. This experiment was conducted at liquid nitrogen temperature on a FEI G2 Polara 300 kV F30 Helium transmission electron microscope equipped with a postcolumn GATAN energy filter and a 4096^2 pixel lens coupled CCD camera (Ultracam, GATAN). The camera was set to $2x$ on-chip binning to form the final image size of 2048^2 pixels. The energy filter was centered at 0 eV with a 20 eV energy loss window. An in-house iterative alignment method was performed on the original tilt series prior to tomographic reconstruction. Fig. 4(a) presents the aligned projection image at 0°, while the central z slice of the volume reconstructed on a cluster containing 4 GPUs is shown in Fig. 4(b). The quality of the reconstructed volume is comparable to the CPU reconstruction.

Several factors are expected to affect the overall system performance including the number of GPUs involved in the computation, the number of nodes, which decides data traffic flow over the network, and the various GPUs having different computing power. At first, the role of number of GPUs was investigated. This test was performed on a system that contains 5 GTX295 cards (10 GPUs total, referred to as GTX GPUs thereafter) distributed over 3 computing nodes running the Fedora Linux operating system (2 nodes have 2 cards and one node has a single card). Each GTX GPU has 240 CUDA

Fig. 3. Iterative reconstruction of the Barbara data set that contains 141 projections in the range of $\pm 70^\circ$ with the angular step of 1°. (a) Original Barbara image, (b) 100 iterations of SIRT reconstruction, and (c) 100 iterations of EM reconstruction.

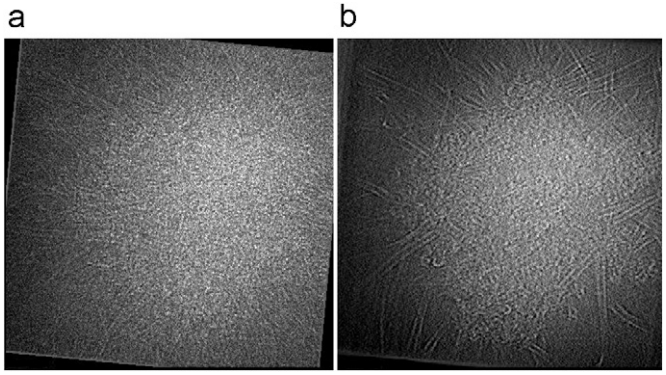

Fig. 4. *Drosophila centrosome* with microtubules re-grown with bovine alpha/beta-tubulin and frozen in vitreous ice. (a) Aligned tilt image at 0° and (b) central z slice of the tomogram reconstructed on a 4-GPU system.

cores and 896 MB memory, and gives rise to 894 GFLOPs peak single precision floating point performance. Therefore, the entire system delivers nearly 9 TFLOPs for single precision floating point calculation and approximately 9 GB memory for tomographic reconstruction. Although our system has no restriction on where the data should be stored, the input projections and generated volumes were all placed on the local disk of the client host to allow separate measurements of disk IO and network overheads. The reconstruction was performed repeatedly to test the system performance given various numbers of GPUs. (Note that users can specify not only the nodes but also number of GPUs on each node to join the computation without physically altering the hardware.) Not only is this information useful during the decision making for what scale of the GPU computing system should be built in accordance with the dimensions of reconstruction, but it also helps end users select sufficient GPUs for their jobs and share the remainder for other purposes. Fig. 5 depicts the performance change under the various GPU configurations. The total time is the duration from the end user's perspective needed to complete the reconstruction. It was measured from the moment the user issued the reconstruction command to when the volume was saved on the disk. The reconstruction time represents the computational expense that was measured between the moment the client process requests the reconstruction and the moment the last node reports its completion of the reconstruction and is interchangeably referred to as the computational time hereafter. The system overhead, i.e., the difference between the total time and the reconstruction time, stems mainly from disk IO operations and projection and volume data flow across the network. The overall system performance is given in Fig. 5(a), where the reconstruction time is also presented for comparison. Although the system performance is found improved in terms of absolute time in Fig. 5(a), the acceleration factor, the ratio of the time for one GPU to solve the problem to the time needed for a specified number of GPUs to solve the same problem, would be more specific to reflect the performance gain due to expanding the number of GPUs in the system. In the ideal case, where the workload can be distributed among all the GPUs without additional overhead, the variation of the acceleration factor with respect to the number of GPUs would be a straight line with a slope of one. In terms of the computational performance, the acceleration factor is found very close to a straight line having a slope of 0.95. This trend suggests that adding more GPUs is an effective approach for continuous enhancement of the computational performance. This is not always true, however, in terms of the overall system performance. As the computational performance keeps improving, the system overhead becomes increasingly relevant, thereby decreasing the benefits of adding additional GPUs (Fig. 5b). Even so, the overall system performance has been boosted nearly seven-fold when the system is expanded to have 10 GPUs.

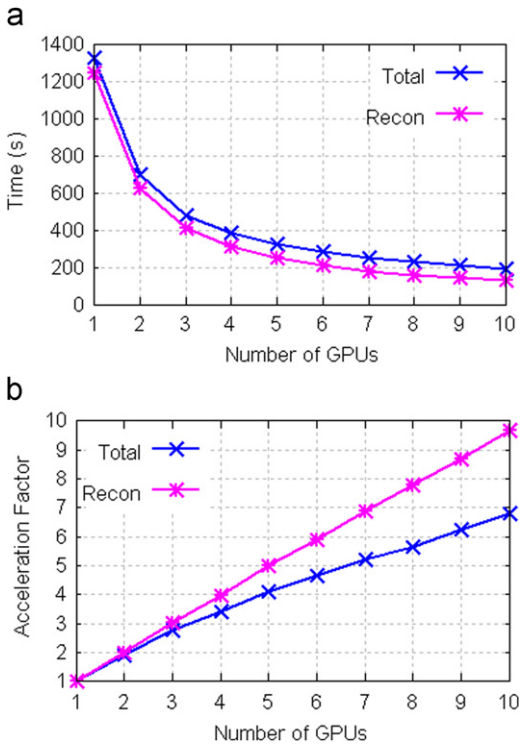

Fig. 5. Performance of the NVIDIA GTX295 based system under various GPU configurations for 10 cycles of SIRT reconstruction for a tomogram of $2048^2 \times 256$ voxels from a tilt series containing 122 projection images of 2048^2 pixels. (a) The total and reconstruction time in seconds when multiple GPUs were involved in computation and (b) the acceleration factor is defined as the ratio of single-GPU time over the multi-GPU time.

Table 1
Overhead for reconstruction performed over various number of computing nodes with four GPUs in total. The input projections have $2048^2 \times 122$ pixels whereas the generated volume is of $2048^2 \times 256$ voxels. Both the input and output are 4 bytes floating point.

Number of nodes	Local	1	2	3
GPU distribution	4	4	3+1	2+1+1
System overhead (s)	***23.2*** $\pm$ ***0.4***[a]	***77.0*** $\pm$ ***0.5***	***73.7*** $\pm$ ***1.8***	***72.5*** $\pm$ ***1.1***
Network overhead (s)	***0***	***53.8*** $\pm$ ***0.6***	***50.5*** $\pm$ ***1.8***	***49.3*** $\pm$ ***1.2***
Recon. time (s)	***316.2*** $\pm$ ***0.1***	***318.4*** $\pm$ ***0.1***	***318.9*** $\pm$ ***1.5***	***319.2*** $\pm$ ***0.1***
Total time (s)	***339.4*** $\pm$ ***0.4***	***395.4*** $\pm$ ***0.6***	***392.6*** $\pm$ ***2.7***	***391.8*** $\pm$ ***1.2***

[a] Since there is no network operation involved under the "Local" configuration, this system overhead represents the baseline overhead common to all other node configurations. By subtracting this baseline from the system overhead, the network overhead is thus obtained in the fourth row.

Fig. 5(a) also shows that the difference between the total and computational time, i.e., the system overhead, is constant regardless of the number of GPUs or equivalently, the number of nodes since the different GPU configurations correspond to various number of nodes in the cluster. As mentioned, the system overhead mainly originates from the disk IOs and data flow across the network. Since the disk IOs are performed only on the client host, the corresponding overhead is apparently independent of the number of nodes in the cluster. This suggests that the network overhead is independent of the number of nodes. To confirm this observation and identify the major contribution to the system overhead, a fixed size volume ($2048^2 \times 256$) was repeatedly reconstructed from 122 projections of a 2048^2 pixels under various node configurations that all have 4 GTX GPUs in total. The results are given in Table 1.

The first row in Table 1 indicates how many nodes involved in the reconstruction. "Local" represents the configuration where both the client and reconstruction processes were running on the same machine. The second row shows how the four GPUs are located in these nodes. For example, the configuration of "2+1+1" indicates that the four GPUs are scattered over three nodes of which one provides 2 GPUs whereas the remaining two each provides a single GPU. The system overhead is given in the third row. The "Local" configuration should not induce any network overhead, thus the system overhead under this configuration represents a baseline overhead common to all other configurations. Therefore, the network overhead given in the fourth row was calculated by subtracting this baseline (23.2 s) from the system overhead. The network overhead remains quite constant, although some slight decrease can also be observed as more nodes were involved. To understand this phenomenon, it should be noted that the bulk of the network traffic (neglecting control messages between the client and nodes) is set solely by the size of the reconstruction and projection series and not by the number of nodes. That sets a firm lower limit on the time needed for network overhead. It is also worthy of pointing out that the network operations contribute a major portion to the system overhead. Therefore, using a higher speed network should have the most impact on lowering the system overhead.

As a convenience, the corresponding reconstruction time is also included in Table 1, which is almost identical with slight fluctuation, suggesting that the computational load is evenly distributed to the nodes.

Obviously, not all reconstructions necessitate using a large number of GPUs. Having a large-scale system frequently reconstruct small volumes is certainly not an efficient use of resources. Therefore, we would like to explore what volume size makes most efficient use of the system containing 10 GTX GPUs. In the subsequent test (Fig. 6), we reconstructed volumes of various sizes by altering the z height with fixed xy dimensions (2048×2048). The reconstruction time shown Fig. 6(a) varies roughly in linear fashion for most z heights, however, the time cost per voxel (Fig. 6b) is high for small z heights. This may indicate the GPU computing power is not saturated when small volumes are reconstructed. As the z height grows, this profile decreases more slowly and becomes essentially flat around $z=640$. The system has reached its maximum computing power. Therefore, it is recommended that the balance between the performance and the cost not be overlooked when building such a system.

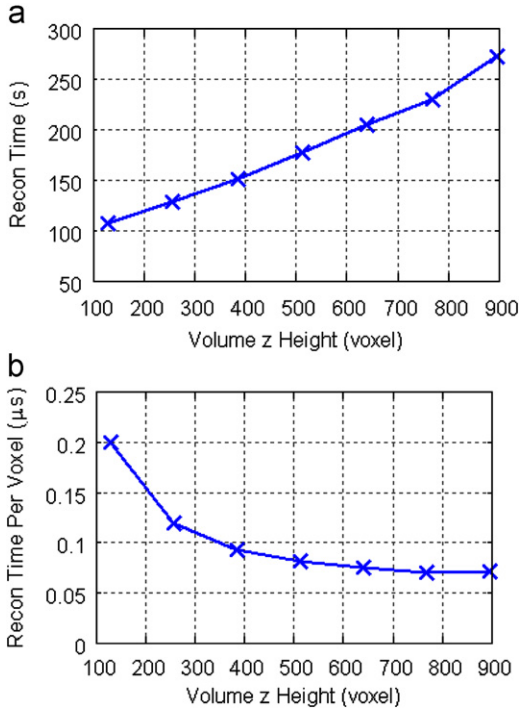

Fig. 6. Performance of a system formed by 10 GTX GPUs to reconstruct volumes of various z height with 2048^2 in x and y dimensions. (a) Reconstruction time in seconds and (b) reconstruction time per voxel in microseconds.

Although the EMT reconstruction can be successfully performed on GTX295 cards, they were originally designed as video cards. Inspired by the great success of GPUs in scientific computation, NVidia has also developed Tesla and most recently Fermi GPUs as co-processors specifically for computation. Since Fermi was not available when this work was conducted, the performance of Tesla GPUs was investigated by running the EMT reconstruction on our Tesla S1070 system with 4 T C1060 cards installed. Since a Tesla C1060 card has 1 GPU as opposed to 2 GPUs per GTX295 card, the second system was configured with 2 GTX295 cards allowing comparison of the same number of GPUs. It should be also pointed out that the Tesla and GTX GPUs have the same number of CUDA cores (240/GPU) and nearly identical processor speed (Tesla: 1.3 GHz, GTX: 1.24 GHz), but differ significantly in memory size (Tesla: 4 GB/GPU, GTX: 896 MB/GPU) with GTX295 having a ten percent higher memory bandwidth (Tesla: 102 GB/s, GTX: 112 GB/s). The same projection data was used to repeatedly reconstruct volumes with fixed xy dimensions (2048^2) but variable z height. The reconstructions were performed locally to avoid the potential biased performance measurement by network traffic. Therefore, Fig. 7 presents only the time consumed for reconstruction.

While the Tesla S1070 system performs better for smaller volumes of z height less than 400 voxels, the GTX295 catches up and eventually outpaces the Tesla S1070 system as the volume size grows. It should be pointed out that this comparison is based upon the same number of GPUs. However, since each GTX295 card has two GPUs and is less expensive than the single GPU Tesla C1060 card, a system with 4 GTX295 cards obviously performs reconstructions far faster and significantly cheaper than its counterpart with the same number of Tesla C1060 cards. According to the performance data given in Fig. 5(b), the reconstruction time is cut by nearly half when the number of GPUs is doubled.

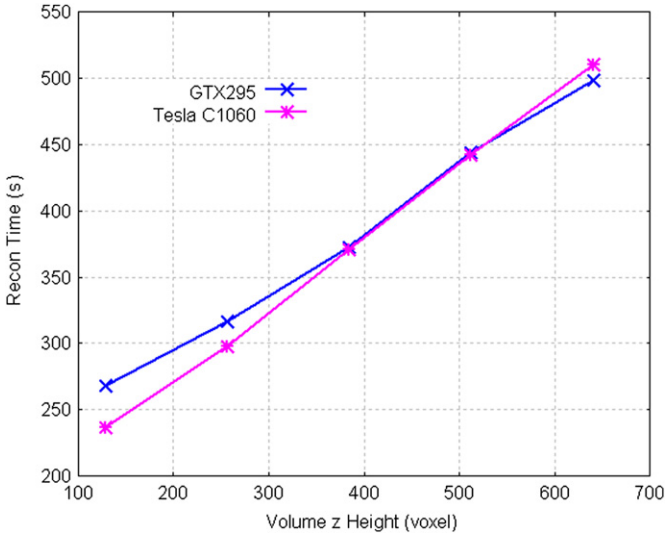

Fig. 7. Performance comparison of GTX295 cards versus Tesla C1060 graphics cards for EMT reconstruction. The reconstructions were performed on 4 GPUs provided either by 2 GTX295 cards or 4 T C1060 cards. The volumes have various z height but fixed xy dimensions (2048^2) reconstructed from projections of $2048^2 \times 122$ pixels.

Table 2
Performance of reconstructing EMT volumes of various dimensions by 10 iterations of SIRT using a GPU cluster containing three computing nodes with total 10 GPUs of GTX295.

Proj. size (pixels)	$1024^2 \times 122$		$2048^2 \times 122$		$4096^2 \times 122$	
Vol. size (voxels)	$1024^2 \times 256$	$1024^2 \times 512$	$2048^2 \times 256$	$2048^2 \times 512$	$4096^2 \times 256$	$4096^2 \times 512$
Total time (s)	***38.9***	***59.8***	***194.5***	***355.2***	***1422.2***	***1845.0***
Recon. time (s)	***22.5***	***32.4***	***129.1***	***178.6***	***821.2***	***1031.9***
Sys. overhead (s)	***16.4***	***27.4***	***65.4***	***176.6***	***601.0***	***813.1***

The less desirable performance of Tesla GPUs, in our opinion, is attributed to 2D textures used in our implementation of the kernel functions, leading to sequential reconstruction of slice by slice along the *y* direction. As a result, the potential benefit of the larger memory provided by Tesla is not realized, the number of processing cores becomes the decisive factor in the current implementation. This, combined with the significantly lower cost of the GTX295 cards, makes them the most cost effective system for high performance EM tomography.

Although this 10-GPU system formed by three computing nodes is by no means a large-scale cluster and costs less than $15,000, we would like to see how it performs for reconstructing volumes with typical dimensions in EMT. Table 2 presents both the total and reconstruction time for 10 iterations of SIRT reconstruction. Note that all the input and output data are single-precision floating point and all stored locally on the client host. It takes less than 40 seconds in total to reconstruct a volume of $1024^2 \times 256$ voxels from an EMT tilt series of $1024^2 \times 122$ pixels whereas reconstructing a volume of $2048^2 \times 512$ voxels from 122 projections of 2048^2 pixels takes less than six minutes (total time). Although at this time $4096^2 \times 512$ voxels are by no means a typical size for EMT reconstructions, we listed the corresponding performance data to show that such a small cluster can indeed reconstruct a very large volume within a reasonable amount of time ($\sim$30 min. total).

It should also be noticed that the system overhead exceeds the reconstruction time when the volume of $2048^2 \times 512$ voxels was computed from projections of $2048^2 \times 122$ pixels. This hints that further effort on improving the system performance should be balanced between the expansion of the computing power and the reduction of the system overhead. Since Table 1 indicates that the network operations contribute a major portion to the system overhead, a higher speed network, 10 GB Ethernet or infiniband, is worthy of consideration.

6. Conclusion

By grouping numerous GPUs over the network and multi-scale parallel implementation we have developed a distributed multi-GPU system that can provide tremendous computing power required by the very large EMT reconstructions that used to be impractical with CPU implementations. For a 10 cycle SIRT reconstruction of a volume of $4096^2 \times 512$ voxels from projections of $4096^2 \times 122$ pixels, it took 10 GPUs provided by 5 GTX295 cards scattered over 3 computing nodes a total of 1845 s, whereas reconstructing a volume of $1024^2 \times 256$ voxels from a EMT tilt series of $1024^2 \times 122$ pixels can be finished in less than 40 s (total time). In addition, this system is highly scalable, allowing more GPUs to be integrated when available for even larger EMT reconstructions or to be tuned with fewer GPUs for smaller scale reconstructions allowing sharing of system resources.

The current system implemented only a single-axis EMT reconstruction. Therefore, the projections must be prealigned by other means before they can be reconstructed. While this may limit its versatility, such calculations are typical for EM tomography. Given its superior performance in reconstructing large volumes, it then becomes practical to use GPU clusters like this to compute iterative cycles of reconstruction and realignment, much as is now done for single particle data. For very large volumes, the computational burden of such a strategy would be impractical even on a large CPU cluster. The net result should be a notable improvement in reconstruction quality and the elimination of the need for fiducial markers that are often used as an aid for projection pre-alignment.

Acknowledgement

This work has been supported in part by grants from the NIH (1S10RR026814-01, Y. C.), GM31627 (DAA) and the Howard Hughes Medical Institute.

References

[1] C. Suloway, J. Shi, A. Cheng, J. Pulokas, B. Carragher, C. Potter, S.Q. Zheng, D.A. Agard, G.J. Jensen, J. Struct. Biol. 167 (2009) 11.
[2] S.Q. Zheng, A. Matsuda, M.B. Braunfeld, J.W. Sedat, D.A. Agard, J. Struct. Biol. 168 (2009) 323.
[3] S.Q. Zheng, B. Keszthelyi, E. Branlund, J.M. Lyle, M.B. Braunfeld, J.W. Sedat, D.A. Agard, J. Struct. Biol. 157 (2007) 138.
[4] J. Tong, I. Arslan, P. Midgley, J. Struct. Biol. 153 (2006) 55.
[5] G.J. Jensen, A. Briegel, Curr. Opinion Struct. Biol. 17 (2) (2007) 260.
[6] J.-J. Fernandez, D.G. Gordon, R. Gordon, J. Parallel Distrib. Comput. 68 (2008) 626.
[7] J.I. Agulleiro, E.M. Garzon, I. Garcia, J.J. Fernandez, J. Struct. Biol. 170 (2010) 570.
[8] H. Winkler, K.A. Taylor, Ultramicroscopy 106 (2006) 240.
[9] D.C. Diez, H. Mueller, A.S. Frangakis, J. Struct. Biol. 157 (2007) 288.
[10] D. Castano-Diez, D. Moser, A. Schoenegger, S. Pruggnaller, A.S. Frangakis, J. Struct. Biol. 164 (2008) 153.
[11] F. Xu, W. Xu, M. Jones, B. Keszthelyi, J. Sedat, D.A. Agard, K. Mueller, Comput. Meth. Prog. Biomed. 98 (2010) 261.
[12] W. Xu, W. Xu, M. Jones, B. Keszthelyi, J. Sedat, D.A. Agard, K. Mueller, J. Struct. Biol. 171 (2010) 142.
[13] F. Vazquez, E.M. Garzon, J.J. Fernandez, J. Struct. Biol. 170 (2010) 146.
[14] B. Jang, D. Kaeli, S. Do, H. Pien, in: Proceedings of the Sixth IEEE International Conference on Symposium on Biomedical Imaging, 2009, pp. 185.
[15] H. Chen, W. Clyborne, J.W. Sedat, D.A. Agard, Proc. SPIE Int. Soc. Opt. Eng. 1660 (1992) 784.
[16] J.-J. Fernandez, A.F. Lawrence, J. Roca, I. Garcia, M.H. Ellisman, J.-M. Carazo, J. Struct. Biol. 138 (2002) 6.
[17] R. Marabini, E. Rietzel, R. Schroeder, J. Struct. Biol. 120 (1997) 363.
[18] P. Gilbert, J. Theor. Biol. 36 (1972) 105.
[19] A.H. Andersen, A.C. Kak, Ultrason. Imaging 6 (1984) 81.
[20] http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf 89, 97.