Parallel Processing and Large-Field Electron Microscope Tomography

Albert Lawrence, Sébastien Phan and Rajvikram Singh
National Center For Microscopy and Imaging Research, University of California, San Diego,
9500 Gilman Drive, La Jolla, CA, 92093-0608 USA

Abstract

We review some recent progress in improving the speed of electron microscope tomography through highly parallel algorithms implemented on parallel computers, clusters and graphics processor boards in desktop computers.

1. Introduction

The electron microscope (EM) has long been used to probe the structure of materials beyond the resolution of the light microscope, down to the molecular level. High resolution EM images reveal a complex ultrastructure inside cells from living organisms. Combined with tomographic techniques [2], electron microscopy has been an indispensable tool for probing the three-dimensional structure of cells and sub-cellular organelles.

Because of the extreme magnification of EM reconstructions and small field of view it is difficult to determine the organization of sub-cellular components on longer length scales. Living beings are organized on every length scale, from the whole organism down to collections of molecules. Thus, immense volumes of data must be collected to build a picture coherent with light microscope data. The process of data collection has been somewhat simplified by moving to larger image detectors: 8K X 8K pixels or larger.

Still, in order to develop a 3D atlas corresponding to the light microscope images, many tomographic reconstructions (serial sections or montages) are required. Raw data for tomographic reconstructions comprises between 60 images for lower quality reconstructions and 360 images for higher quality reconstructions. Therefore for the larger format images a typical data set for a single reconstruction may occupy as much as 50 gigabytes on disk, and a single reconstruction may approach a terabyte. Data for montages and serial sections will scale accordingly.

In order to get some estimate of the magnitude of the computational problem, we also note that the standard method, filtered backprojection, is of order, $N^3$ where raw data images are $N \times N$ [4], [5]. A second complication arises from the circumstance that the usual assumption in X-ray tomography that the illuminating radiation travels along straight-line rays is not true in the electron microscope. This is because focusing is performed by means of magnetic fields and electrons travel in helicoidal trajectories under the influence of the magnetic fields [5].

The problem of trajectory curvilinearity becomes progressively worse as image sizes increase, with deviations in from 50 pixels or more at the periphery of large images taken at high sample tilts [5]. Thus, a second complication arises from the necessity of applying nonlinear corrections to compensate for the trajectories of the electrons. This requires the evaluation of polynomials of three variables and degree up to five or six, so a naïve implementation of the backprojection could increase the computational burden by a factor of a several hundred. We discuss our solution of this problem by means of algorithms suited to a high degree of parallelism. These algorithms have implemented on parallel computers, computer clusters and GPU boards in computer workstations.

Finally, the problem of obtaining reconstruction atlases from large format images is compounded by the problem of stitching the digital reconstructions together. A careful analysis of the alignment problem shows that the three dimensional geometry of the reconstructed object can be determined only up to a “gauge ambiguity” [2]. This has been discussed in the case where the maps are projective; the general case follows in a similar way. In order to handle the problem of gauge ambiguity [2, 6], we have developed a method for choosing the best representative corresponding to orthogonal projections out of all possible geometric structures for a given reconstruction, but there is no good way to guarantee that 3D reconstructions of overlapping regions can be chosen in such a way that the geometries correspond in the overlap region. The solution to this problem is to appeal to the “gauge ambiguity” to modify the model of the electron trajectories. In essence, we calculate synthetic images, which would be formed by strictly linear trajectories. This requires solution of a second
order partial differential equation. Here, again, parallel processing offers a solution to the computational problem. We will discuss plans for future work in this context.

Much of this work has been implemented via a parallel version of the software package, TxBR, [5] which has been written to address the special issues of large scale EM tomography. TxBR is in production use and has been adapted for various parallel computers, computer clusters and processors with multiple graphical processor unit (GPU) boards. We review some recent progress in improving the speed of electron microscope tomography through highly parallel algorithms implemented on parallel computers, clusters and graphics processor boards in desktop computers.

2. Approaches

As outlined in the introduction, a single data set for a 3D reconstruction would comprise 60 images taken at various rotations around a single tilt axis. For a field of view of $8K \times 8K$ pixels this gives $3.8$ GPixels per tilt series. At present, a complete data set may comprise as many as 6 tilt series, so the total pixels per data set approaches $2.5 \times 10^{10}$ pixels.

On the other hand, a backprojection calculation requires a forward projection for every tilt and every voxel in the reconstructed object. Reconstruction of the imaged portion of a section requires 10 to $20 \times 10^{21}$ calculations of the forward projection maps corresponding to the electron trajectories, if we assume the reconstruction is between 500 and 1000 pixels along the z-axis and 8K pixels along the x- and y-axes. Calculations of this magnitude require both efficient algorithms and parallel processing. We review one such algorithm in the next section. If this particular algorithm is employed, typical computation time for a six axis tilt series would run about five days on a quad processor workstation. Because delays of this magnitude are generally unacceptable in situations where relatively large volumes must be mapped, requiring tens or hundreds of overlapping reconstructions, parallel processing must also be employed to attain the necessary speedup.

2.1. Fast Recursion Algorithms

As we note in the introduction, electrons travel along curvilinear trajectories in an electron microscope, so we must employ a nonlinear backprojection scheme. This entails polynomial approximations to the forward projections. In particular a bundle adjustment procedure is employed to model the locations of distinguished points in the object and projection maps from these distinguished points in the to corresponding points in the image [5]. These polynomial projections may be applied to arbitrary object points given by coordinates $(x_1, x_2, x_3)$. In particular this point is mapped to a point $(x_1, x_2)$ on each electron micrograph through a non-linear mapping:

$$x_{col} = P_{col}(x_1, x_2, x_3) = \sum_{n, j, k, l \geq 0} b_{ijkl} x_1^j x_2^k x_3^l$$

where $\omega$ identifies the tilt and $n$ is the polynomial approximation order for this mapping. During the backprojection step, the density of every object point is then evaluated from its corresponding pixel values in the electron micrographs; the latter having been applied an r-weighted filter beforehand. We refer to [5] for more details.

Evaluating polynomials like $P_{col}(x_1, x_2, x_3)$ at each point of a large regular grid can be computationally very intensive. A significant speed improvement is however achieved with the use of a simple recursive scheme. Polynomial functions can be computed using only additions when moving from neighbor to neighbor along the $X_1$ direction. The variables $X_2$ and $X_3$ are held constant. The problem is here reduced to a single variable recursion problem. In the case of a polynomial function $q$ of order $n$ (of a single variable), knowledge of the $n$ first finite differences at one node $q_m$ is sufficient to be able to calculate their values on the next node $i+1$:

$$q_{m+1} = q_m + q_{m+1}.$$  

Note that the table of values is built from the bottom. Finite differences of order $n$ (and higher) are a constant (or zero) over the entire grid making this scheme possible. Evaluating $q_{m+1}$ from $m = n - 1$ to 0 ends up to the polynomial evaluation $q = q_{m+1}$ on node $i+1$. More details may be found in [6].

Eventually (for $n \geq 2$), this evaluation method will deteriorate after many recursions as the rounding error propagates, resulting in a blurring of the reconstruction at high $X_1$ value. This effect becomes stronger as $n$ increases. To avoid this issue, we split a full $X_1$ line in as many segments as needed for the rounding error to remain acceptable within the whole section range. When a new segment begins, the $n$ first finite differences are initialized, meaning calculated directly from their polynomial expression, and the recursion
scheme starts from scratch. Section length is calculated prior to the reconstruction by enforcing the direct polynomial evaluation and its recursive value to be less than a given value \( \epsilon \) on a specific test case (e.g. at \( X_2 = 0 \), \( X_3 \) and sample orientation parallel to the CCD).

### 2.2. Parallel Processing

In recent months the high performance computing community has embraced GPGPUs (General Purpose Graphics Processing Units) for high performance desktop computers. With the latest GPU cards approaching a top speed of a teraflop per unit and costing under $500 USD per unit, it is possible to run computationally intensive codes that were considered to be in the realm of clusters and supercomputers on inexpensive PC hardware. Each processing unit is capable of running a thread of code making the architecture ideal for highly parallel problems.

The TxBR back-projection algorithm is by nature embarrassingly parallel. Any sub-set of the final volume can be reconstructed independently, given a data set of electron micrographs and their associated projection maps. We have parallelized the backprojection portion of the TxBR code to run either on computer clusters or on GPU cards. In both cases, the algorithm is quite similar. As we note below, the main difference between the two approaches resides in the amount of available memory (RAM) during computation, which de facto defines the granularity of the reconstruction done at every node. The architectural differences are illustrated in Figure 2.2.1.

During the back-projection routine, density at each voxel of the reconstruction receives a contribution from a corresponding pixel intensity of a pre-filtered micrograph. This correspondence is given by the projection maps, which are calculated during alignment. During the process, one micrograph, its associated projection map as well as the subset being reconstructed should be available in the RAM of each of the nodes. Electron micrographs can be as big as 8x8k pixels (256Mb if encoded with float numbers). In today's clusters, each node should therefore be able to reconstruct a volume containing at least several sections of the micrograph size.

Porting the TxBR back-projection code to run on a computer cluster is straightforward, the application running at each node being the exact original single-threaded code. In practice, we used mpi4py, a python module that provides bindings to MPI (Message Passing Interface) and allows simultaneous initialization of the reconstruction threads in this parallel environment.

In GPU cards, on the other hand, the available RAM per node is hundred times smaller: the atomic reconstruction subset at each individual node has to be smaller and was chosen to be one pixel line. All the computation for each particular row and its backprojection into the 3D block are calculated in parallel with the other threads.

![Figure 2.2.1 CPU an GPU architectural layouts.](image)

Aside from the number of processors, the main difference is the larger hierarchical memory cache in the typical CPU.

An additional problem with GPU cards is that the lack of cache increases the latency for memory accesses, which simply means that a thread trying to fetch a data block for computation has to wait several hundred CPU cycles for the data to arrive. This might seem like a severe handicap in terms of performance, but architectures such as Nvidia’s CUDA (Compute Unified Device Architecture) are capable of scheduling thousands of light threads. By running an optimally large number of threads, it is possible to keep the processing units busy and effectively hide the large memory latency in the system.

### 3. Results

Table 3.1 shows the speed-ups seen by running the parallel version of the algorithm on a GPGPU with different data sizes and orders of approximations. A speed-up for an algorithm is defined as the ratio of computation time on a parallel architecture to the computation time of a fast serial implementation. For the results shown in Table 3.1, the parallel computation time was observed by running several hundreds of CUDA threads on an Nvidia GTX280 graphics card. The serial times were computed by running the same jobs on a 2.8 GHz AMD Opteron processor.

Table 3.1 shows the speed-ups seen by running the parallel version of the algorithm on a GPGPU with different data sizes and orders of approximations. A speed-up for an algorithm is defined as the ratio of computation time on a parallel architecture to the computation time of a fast serial implementation. For the results shown in Table 3.1, the parallel computation time was observed by running several hundreds of CUDA threads on an Nvidia GTX280 graphics card. The serial times were computed by running the same jobs on a 2.8 GHz AMD Opteron processor.

As can be seen from the table below, the trend is toward greater speedup as the order of the polynomial approximation increases. This is consistent with the conjecture that as the order of the polynomial increases the processor units spend relatively more time in arithmetic operations and relatively less time in accessing off-board memory. This would imply less
memory conflict and thus a higher effective computation rate. At present the profiling capabilities of software supplied by Nvidia does not give sufficient information to explain other aspects of the data in the table.

Table 3.1 Speed-ups seen on different data sizes and orders of approximations. Order is the degree of the polynomial approximation of the forward projection and the data size is the size of the EM images in the raw data.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Order 1</th>
<th>Order 3</th>
<th>Order 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K x 1K</td>
<td>3.7 X</td>
<td>4.6 X</td>
<td>16.2 X</td>
</tr>
<tr>
<td>2K x 2K</td>
<td>4.6 X</td>
<td>6.2 X</td>
<td>44.9 X</td>
</tr>
<tr>
<td>4K x 4K</td>
<td>4.2 X</td>
<td>5.1 X</td>
<td>31.8 X</td>
</tr>
<tr>
<td>8K x 8K</td>
<td>4.8 X</td>
<td>6.8 X</td>
<td>45.7 X</td>
</tr>
</tbody>
</table>

4. Discussion

The obvious benefits of using GPUs for algorithms such as backprojection are significant speed-ups on relatively inexpensive hardware that can be put together using commercial off-the-shelf components. Unlike supercomputers and clusters hosted by computer centers that are shared by hundreds of users at any given time, these desktop supercomputers provide the advantage of dedicated compute cycles for applications requiring a guarantee on the quality of service.

The other major advantage of GPU based processing is the energy efficiency exhibited by the watts/flops numbers of these units. At the time of writing, it is possible to purchase a 4-GPU 4 Tflop unit that consumes 800 Watts. This makes the running and operating cost of GPU based solutions a fraction of clusters providing the same level of computational capability. Energy efficiency also translates to reliability and more efficient space utilization.

Beyond these considerations, we expect that future implementations of GPU cards with on-board cache should improve computational speeds for a wide range of image processing algorithms. The backprojection algorithm we implemented on present hardware was ported from a C version with very little modification. Further speedup could also be obtained by further modification for GPU architectures.

EM tomography presents many other opportunities for parallel processing. In addition to numerous standard image processing problems such as noise filtering, feature recognition and tracking there are more specialized problems. One example is the implementation of the proper filtering operation for the generalized Radon transform. Although we use a standard one-dimensional filter in the inversion of the transform, and obtain satisfactory results, we believe that we can gain further improvements in reconstruction quality by implementing a spatially dependent filter, either before or after the backprojection [6]. This is because the theory of the generalized Radon transform dictates that the correct filter is actually an elliptic pseudodifferential operator [2] and its implementation of a computer would be an integral kernel that behaves like a spatially dependent convolution. Needless to say such an operation on the data would be computationally expensive, so GPU cards might offer a hardware solution to the problem of computational resources.

A second computational problem peculiar to EM tomography is the montaging problem. Because the electron trajectories are curvilinear, matching the reconstructions along the overlaps is a three-dimensional problem and some “straightening” of trajectories may be necessary to make the overlap regions match. In other words, it may be necessary to make the overlap regions match. Thus, it may be necessary to compute “synthetic” EM images, which would correspond to straight-line electron trajectories. These synthetic images may be computed as the solution of a boundary value problem for a system of ultrahyperbolic partial differential equations, also known as John’s equations [7]. Since solution of PDEs in two or more dimensions is computationally expensive, GPU hardware would offer a cost-effective solution.

5. Conclusions

The use of GPU cards gives many-fold speedup compared with the same tomography algorithms run on standard workstations. Given the relative cost of computer clusters, using one or more GPU cards appears to be the most cost effective way to attain the necessary throughput for large format EM tomographic reconstructions. Further work on implementation of algorithms and hardware improvements could lead to further significant improvements in computational speed. We also note that future work in this area would also bring other computationally expensive algorithms associated with EM tomography into the realm of practical application.

6. Credits

This work was supported by grants from the NIH National Center for Research Resources (NCRR) under award number P41RR008605, the National Biomedical
Computation Resource to Peter Arzberger, and award number P41RR004050, the National Center for Microscopy and Imaging Research to Mark Ellisman. The work described in this paper was conducted at NCMIR, UCSD and is supported by NIH Grant RR004050

7. References


