GViewer: GPU-Accelerated Graph Visualization and Mining

Jianlong Zhong and Bingsheng He

Nanyang Technological University

1 Introduction

Visualization is an effective way of identifying the patterns of interests (such as communities) in graphs including social networks and Web [8,6]. There have been a number of tools developed for graph visualizations, e.g., Tulip, Gephi and GMine [8]. All of these tools use the CPU as the main power to calculate the graph layouts for visualization, such as force-directed layout [2]. However, the layout calculation is usually computation intensive, for example, the force-directed layout has the complexity of $O(N^3)$, where N is the number of vertexes in the graph. In our experiments, the CPU-based solution takes more than half one hours on the CPU to layout a graph with 14.5 thousand vertexes.

Instead of laying out the entire graph, existing tools usually address this performance issue with an *off-line* multi-scale approach, where the entire graph is partitioned with the multi-level partitioning algorithm. The graph layout is limited to the graph data at the lowest level, and each partition consists of dozens of vertexes. While the multi-level approach improves the response time, the static graph partitioning has limited the flow and the scope of graph exploration. Users can only follow the pre-computed multi-level graph layout to explore the graph. Additionally, there is little information visualized for boundary vertexes at each graph partition. The limited flexibility hurts the effectiveness of visualization on graph mining.

With the limitations of existing graph visualization tools in mind, we propose to accelerate the graph layout calculation with graphics processors (GPUs), and further to support interactive graph visualization and mining. The adoption of GPU is motivated by the recent success of GPGPU (General Purpose computation on GPUs), where GPUs have become many-core processors for various database tasks [4,5]. As a start, we develop a graph layout library on the GPU. The library includes multiple commonly used graph layouts [8], such as force-directed layout [2], spectral layout [1] and tree layout [3]. The inherent data parallelism of calculating the graph layouts facilitates implementing the algorithm on the GPU. Moreover, we utilize the GPU hardware features to reduce the memory latency. As a result, the GPU-based graph layout calculation on a NVIDIA Quadro 5000 GPU is over 8.5 times faster than its CPU-based counterpart on the Intel quad-core W3565 CPU. As a side product, calculating the graph layout on the GPU eliminates the overhead of data transfer between the main memory and the GPU memory. Note, existing approaches need to transfer the graph layout data from the main memory to the GPU for rendering.

With the accelerated layout calculation as a building block, we develop user interactions for graph visualization and mining. Currently, user interactions include the simple graph operations, i.e., filtering, vertex selections, zooming in/out, and drilling in/out. Thanks to the GPU acceleration, these user interactions offer good interactive user experiences.

We have implemented these techniques into a system named *GViewer*. We will demonstrate the following two key aspects of GViewer: (1) Efficient graph layout calculation. GViewer performs the graph layout calculation *at runtime* for the subgraph specified in the user interaction. Additionally, we also perform a side-by-side comparison between the GPU-based algorithm and its CPU-based counterpart. (2) User interactions in GViewer to support graph visualization and mining.

2 System Implementation

We implement GViewer with a recent GPU programming framework named CUDA. Currently, GViewer supports the commonly used graph layouts [8], such as force-directed layout [2], spectral layout [1] and tree layout [3]. We use OpenGL for graphics rendering and the CUDA-OpenGL inter-operability support for collaboration between computation and visualization.

GPU-Accelerated Graph Layout. The force-directed layout has good quality layout result, strong theoretical foundations, simplicity and interactivity [2]. The basic idea of the force-directed layout is physical simulation, where vertexes are modeled as objects with mechanical springs and electrical repulsion among them. The edges tend to have uniform length because of the mechanical spring force, and vertexes that are not connected tend to be drawn further apart due to the electrical repulsion.

The force-directed layout calculation is an iterative process. In each iteration, the algorithm calculates the new position for each vertex based on its current position, the total spring force from its neighbor vertexes and the total electrical repulsion from its unconnected vertexes. That is, for each iteration, we need to calculate the force (either spring force or repulsion) between any two vertexes. A basic implementation is that each GPU thread calculates the force for a vertex, through scanning the vertex list and calculating the force during the scan. While the basic implementation takes advantage of the thread parallelism of the GPU, it incurs excessive memory accesses.

We improve the memory performance of the basic implementation with two hardware features of GPU, i.e., coalesced accesses and shared memory. In CUDA, T GPU threads are grouped into a *warp* ($T = 32$ in current CUDA). If the threads in a warp access consecutive memory addresses, these accesses are coalesced into a single request such that the bandwidth utilization is improved. The shared memory is a piece of fast on-chip memory for storing the frequently accessed data. Combining these two features, a warp first reads T vertexes into the shared memory, and then each thread in the warp calculates the partial forces on the T vertexes. Next, this calculation repeats until the vertex list is exhausted. With the coalesced access and the shared memory, the number of memory requests is significantly reduced.

The spectral layout [1] is based on the calculation of the eigenvector of the adjacency matrix of the graph. We implement the Lanczos algorithm for the eigenvector calculation [7] with CUDA BLAS library.

The tree layout is to show a rooted tree-like formation for a graph. It is suitable for a tree-like graph. We use breadth first traversal (BFS) to generate the tree layout. The GPU-based BFS is performed in k iterations. Initially, the input set includes s only, where s is a root vertex defined by the user. In each iteration, we span one hop from the input set of vertexes in order to get all the neighbor vertexes within one hop. We use an array *flag* to indicate whether a vertex is firstly accessed in the kth iteration. Initially, only the flag for s is set to be zero, and other flags are -1. At the ith iteration, we get the neighbor list of the vertex whose flag equals to $(i - 1)$. This is implemented using a map primitive [4]. A map is similar to a database scan, with a CUDA feature *coalesced access* memory optimizations for bandwidth utilization. Next, we set the flag for each vertex in the neighbor list: if the flag is -1, it is set to be i; otherwise, the flag does not change. The iteration ends when no flag is set within the iteration. Given the BFS result, we can calculate the position of each vertex in the display region, by considering the tree height and the fanout [3].

3 Case Studies

We evaluate GViewer on a commodity machine with 2GB RAM, one NVDIA Quadro 5000 GPU and one Intel quad-core W3565 CPU. The operating system is Windows 7. We extract an undirected graph from DBLP (http://dblp.uni-trier.de/xml/) for demonstration: each author as a vertex, and two connected vertexes meaning co-authorship between the two corresponding authors. The co-authorship represents the relationship between any two authors of the same paper. The extracted graph consists of 820 thousand vertices and 5.7 million edges.

We present the major result, including the comparison between the CPU- and the GPU-based implementation, and community discovery.

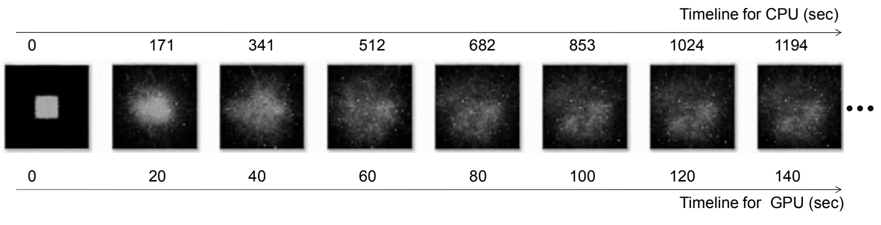

Fig. 1. Side-by-side comparison between the GPU- and the CPU-based force-directed layout

GPU vs. CPU-based layouts. We conduct a side-by-side comparison between the CPU and the GPU. Figure 1 shows the screen shots during the process of the CPU- and the GPU-based visualization on the graph with $D = 96$ and $C = 8$ in the force-directed layout. Along the time line, we can imagine the difference on the user experience between the CPU- and the GPU-based visualizations. For example, in order to see the fourth screen shot, the user needs to wait for 512 seconds on the CPU-based visualization, and only needs to wait for 60 seconds on the GPU-based visualization. Note, the FD-layout algorithm takes around one thousand iterations before the layout becomes stable.

Community Discovery. We demonstrate the flow of exploring the graph with a specific author in order to find his/her co-authorship community. We use Jiawei Han as an example of community discovery: (1) As the first step, we select "Jiawei Han" and highlight its neighbors with two hops. The result is omitted here. (2) We drill down from Jiawei with two hops. GViewer visualizes the subgraph with the force directed layout (The figure is omitted due to space constraints). We observed that Jiawei has a large two-hop co-author community. (3) If we set the number of hops to be one, we can easily find that Jiawei's most important coauthors (Figure 2).

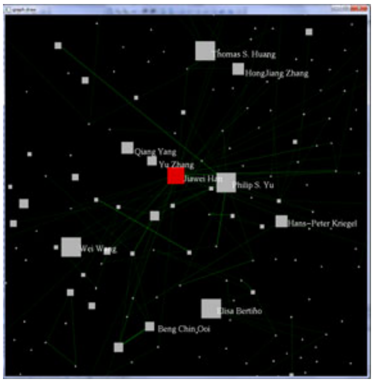

Fig. 2. One hop from "Jiawei Han"

Acknowledgement. This work is supported by an NVIDIA Academic Partnership (2010-2011) and an AcRF Tier-1 grant in Singapore.

References

1. Beckma, B.: Theory of Spectral Graph Layout. Technical report, MSR-TR-94-04 (1994)
2. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. 21(11) (1991)
3. Grivet, S., Auber, D., Domenger, J.-P., Melancon, G.: Bubble tree drawing algorithm. In: International Conference on Computer Vision and Graphics (2004)
4. He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N., Luo, Q., Sander, P.: Relational joins on graphics processors. In: SIGMOD (2008)
5. He, B., Yu, J.X.: High-throughput transaction executions on graphics processors. In: Proc. VLDB Endow., vol. 4, pp. 314–325 (February 2011)
6. Koenig, P.-Y., Zaidi, F., Archambault, D.: Interactive searching and visualization of patterns in attributed graphs. In: Proceedings of Graphics Interface (2010)
7. Parlett, B.N.: The symmetric eigenvalue problem. Prentice-Hall, Inc., Upper Saddle River (1998)
8. Rodrigues Jr., J.F., Tong, H., Traina, A.J.M., Faloutsos, C., Leskovec, J.: Gmine: a system for scalable, interactive graph visualization and mining. In: VLDB (2006)