Speeding Up Network Layout and Centrality Measures for Social Computing Goals

Puneet Sharma[1,2], Udayan Khurana[1,2], Ben Shneiderman[1,2], Max Scharrenbroich[3], and John Locke[1]

[1] Computer Science Department
[2] Human-Computer Interaction Lab
[3] Department of Mathematics
University of Maryland
College Park MD 20740
{puneet,udayan,ben}@cs.umd.edu,
{john.b.locke,max.franz.s}@gmail.com

Abstract. This paper presents strategies for speeding up calculation of graph metrics and layout by exploiting the parallel architecture of modern day Graphics Processing Units (GPU), specifically *Compute Unified Device Architecture (CUDA)* by *Nvidia*. Graph centrality metrics like *Eigenvector, Betweenness, Page Rank* and layout algorithms like *Fruchterman-Rheingold* are essential components of *Social Network Analysis (SNA)*. With the growth in adoption of SNA in different domains and increasing availability of huge networked datasets for analysis, social network analysts require faster tools that are also scalable. Our results, using NodeXL, show up to 802 times speedup for a Fruchterman-Rheingold graph layout and up to 17,972 times speedup for Eigenvector centrality metric calculations on a 240 core CUDA-capable GPU.

Keywords: Social Computing, Social Network Analysis, CUDA.

1 Introduction

The enormous growth of social media (e.g. email, threaded discussions, wikis, blogs, microblogs), social networking (e.g. Facebook, LinkedIn, Orkut), and user generated content (e.g. Flickr, YouTube, Digg, ePinions) presents attractive opportunities for social, cultural, and behavioral analysts with many motivations. For the first time in history, analysts have access to rapidly changing news stories, national political movements, or emerging trends in every social niche. While temporal trend analysis answers many questions, network analysis is more helpful in discovering relationships among people, sub-communities, organizations, terms, concepts, and even emotional states.

Many tools have been developed to conduct Social Network Analysis (SNA), but our effort is tied to NodeXL[1] [3], a network data analysis and visualization [8] plug-in for Microsoft Excel 2007 that provides a powerful and simple means to graph data

[1] NodeXL: Network Overview, Discovery and Exploration for Excel
http://nodexl.codeplex.com/

J. Salerno et al. (Eds.): SBP 2011, LNCS 6589, pp. 244–251, 2011.

contained in a spreadsheet. Data may be imported from an external tool and formatted to NodeXL specifications, or imported directly from the tool using one of the supported mechanisms (such as importing an email, Twitter, Flickr, YouTube, or other network). The program maps out vertices and edges using a variety of layout algorithms, and calculates important metrics on the data to identify nodes and relationships of interest.

While NodeXL is capable of visualizing a vast amount of data, it may not be feasible to run its complex computation algorithms on larger datasets (like those found in Stanford's SNAP library[2]) using hardware that is typical for a casual desktop user. And we believe that in order for data visualization tools to reach their full potential, it is important that they are accessible and fast for users using average desktop hardware.

Nvidia's CUDA[3] technology provides a means to employ the dozens of processing cores present on modern video cards to perform computations typically meant for a CPU. We believe that CUDA is appropriate to improve the algorithms in NodeXL due to the abundance of Graphical Processing Units (GPUs) on mid-range desktops, as well as the parallelizable nature of data visualization. Using this technology, we hope to allow users to visualize and analyze previously computationally infeasible datasets.

This paper has two contributions. The first contribution is to provide the strategies and directions to GPU-parallelize the existing layout algorithms and centralities measures on commodity hardware. And the second contribution to apply these ideas to widely available and accepted visualization tool, NodeXL. To our knowledge, ours is the first attempt to GPU-parallelize the existing layout algorithms and centrality measures on commodity hardware. In our modified NodeXL implementation, we targeted two computationally expensive data visualization procedures: the Fruchterman-Rheingold [1] force-directed layout algorithm, and the Eigenvector centrality node metric.

2 Computational Complexity of SNA Algorithms

Social Network Analysis consists of three major areas - Network Visualization, Centrality Metric Calculation and Cluster detection. Popular techniques for visualization use force-directed algorithms to calculate a suitable layout for nodes and edges. Computationally, these algorithms are similar to n-body simulations done in physics. The two most popular layout algorithms are Harel-Koren [2] and Fruchterman-Rheingold [1]. The computational complexity of the latter can roughly be expressed as $O(k(V^2+E))$ and the memory complexity as $O(E+V)$, where k is the number of iterations needed before convergence, V is the number of vertices and E is the number of edges in the graph. Centrality algorithms calculate the relative importance of each node with respect to rest of the graph. Eigenvector centrality [6] measures the importance of a node by the measure of its connectivity to other "important" nodes in the graph. The process of finding it is similar to belief propagation and the algorithm is iterative with the time complexity O(k.E).

[2] SNAP: Stanford Network Analysis Platform `http://snap.stanford.edu/index.html`

[3] What is CUDA? `http://www.nvidia.com/object/what_is_cuda_new.html`

With the increase in the number of vertices and edges, the computation becomes super-linearly expensive and nowadays analysis of a very highly dense graph is a reasonable task. Therefore, speedup and scalability are key challenges to SNA.

2.1 Approaches for Speedup and Scalability

To achieve greater speedup, two approaches that immediately come to mind are faster algorithms with better implementation, and secondly, more hardware to compute in parallel. If we believed that the best algorithms (possible, or the best known) are already in place, the second area is worth investigating. However, it is the nature of the algorithm which determines that whether it can be executed effectively in a parallel environment or not. If the graph can be easily partitioned into separate workable sets, then the algorithm is suitable for parallel computing, otherwise not. This is because the communication between nodes in such an environment is minimal, and so all the required data must be present locally. We divide the set of algorithms used in SNA into three different categories based on feasibility of graph partitioning:

- **Easy Partitioning** - Algorithm requires only the information about locally connected nodes (neighbors) e.g. Vertex Degree Centrality, Eigenvector centrality
- **Relatively Hard Partitioning** - Algorithm requires global graph knowledge, but local knowledge is dominant and efficient approximations can be made. e.g. Fruchterman-Rheingold
- **Hard Partitioning** - Data partitioning is not an option. Global graph knowledge is required at each node. e.g. Betweenness centrality which requires calculation of All Pairs of Shortest Paths.

Algorithms that fall under the first category can be made to scale indefinitely using a distributed systems like Hadoop [7] and GPGPUs. For the second category, there is a possibility of a working approximation algorithm, whereas nothing can be said about the third category of algorithms.

3 Layout Speedup

3.1 Layout Enhancement

In NodeXL layout algorithms, the Fruchterman-Rheingold algorithm was chosen as a candidate for enhancement due to its popularity as an undirected graph layout algorithm in a wide range of visualization scenarios, as well as the parallelizable nature of its algorithm. The algorithm works by placing vertices randomly, then independently calculating the attractive and repulsive forces between nodes based on the connections between them specified in the Excel workbook. The total kinetic energy of the network is also summed up during the calculations to determine whether the algorithm has reached a threshold at which additional iterations of the algorithm are not required.

The part of the algorithm that computes the repulsive forces is $O(N^2)$, indicating that it could greatly benefit from a performance enhancement (the rest of the algorithm is $O(|V|)$ or $O(|E|)$, where v and e are the vertices and edges of the graph,

respectively). The algorithm is presented in pseudo-code and computationally intensive repulsive calculation portion of the algorithm is italicized in a longer version of this paper [9].

Due to the multiple nested loops contained in this algorithm, and the fact that the calculations of the forces on the nodes are not dependent on other nodes, we implemented the calculation of these forces in parallel using CUDA. In particular, the vector V was distributed across GPU processing cores. The resulting algorithm, **Super Fruchterman-Rheingold**, was then fit into the NodeXL framework as a selectable layout to allow users to recruit their GPU in the force calculation process.

3.2 Results for Layout Speedup

We ran our modified NodeXL on the computer with CPU (3 GHz, Intel(R) Core(TM) 2 Duo) and GPU (GeForce GTX 285, 1476 MHz, 240 cores). This machine is located in the Human Computer Interaction Lab at the University of Maryland, and is commonly used by researchers to visualize massive network graphs. The results of our layout algorithm are shown in the table below. Each of the graph instances listed may be found in the Stanford SNAP library:

Table 1. Results of layout speedup testing

Graph Instance Name	#Nodes	#Edges	Super F-R run time (ms)	Simple F-R run time (ms)	Speedup
CA-AstroPh	18,772	396,160	1,062.4	84,434.2	**79x**
cit-HepPh	34,546	421,578	1,078.0	343,643.0	**318x**
soc-Epinions1	75,879	508,837	1,890.5	1,520,924.6	**804x**
soc-Slashdot0811	77,360	905,468	2,515.5	1,578,283.1	**625x**
soc-Slashdot0902	82,168	948,464	2,671.7	1,780,947.2	**666x**

The results of parallelizing this algorithm were actually better than we had expected. The speedup was highly variable depending on the graph tested, but the algorithm appears to perform better with larger datasets, most likely due to the overhead of spawning additional threads. Interestingly, the algorithm also tends to perform better when there are many vertices to process, but a fewer number of edges.

This behavior is the result of reaping the benefits of many processing cores with a large number of vertices, while the calculation of the repulsive forces *at each node* still occurs in a sequential fashion. A graph with myriad nodes but no edges would take a trivial amount of time to execute on each processor, while a fully connected graph with few vertices would still require much sequential processing time. Regardless, the Super Fruchterman-Rheingold algorithm performs admirably even in

the worst case of our results, providing a 79x speedup in the *CA-AstroPh* set with a 1:21 vertex to edge ratio. The ability for a user to visualize their data within two seconds when it originally took 25 minutes is truly putting a wider set of data visualization into the hands of typical end users.

4 Metric Speedup

4.1 Metric Enhancement

The purpose of data visualization is to discover attributes and nodes in a given dataset that are interesting or anomalous. Such metrics to determine interesting data include *Betweenness centrality, Closeness centrality, Degree centrality,* and *Eigenvector centrality*. Each metric has its merits in identifying interesting nodes, but we chose to enhance Eigenvector centrality, as it is frequently used to determine the "important" nodes in a network. Eigenvector centrality rates vertices based on their connections to other vertices which are deemed important (connecting many nodes or those that are "gatekeepers" to many nodes). By the Perron-Frobenius theorem we know that for a real valued square matrix (like an adjacency matrix) there exists a largest unique eigenvalue that is paired with an eigenvector having all positive entries.

4.2 Power Method

An efficient method for finding the largest eigenvalue/eigenvector pair is the power iteration or power method. The power method is an iterative calculation that involves performing a matrix-vector multiply over and over until the change in the iterate (vector) falls below a user supplied threshold. We outline the power method algorithm with the pseudo-code in Section 4.2 in [9].

4.3 Power Method on a GPU

For our implementation we divided the computation between the host and the GPU. A kernel for matrix-vector multiply is launched on the GPU, followed by a device-to-host copy of the resulting vector for computation of the vector norm, then a host-to-device copy of the vector norm and finally a kernel for vector normalization is launched on the GPU. Figure 4 in [9] outlines the algorithm flow to allow execution on the GPU.

4.4 Results for Eigenvector Centrality Speedup

We tested our GPU algorithm on eleven graph instances of up to 19k nodes and 1.7m edges. As a comparison, we used the serial version of the algorithm implemented in NodeXL with C#. Also, we used the same machine used in section 3.2. Table 2 shows the speedups we achieved by running the algorithm on the GPU.

Speedups ranged between 102x and 17,972x. We investigated relationship between the problem instances and the speedups by examining how problem size, measured by the number of edges, affected the speedup.

There is an increase in the speed up as the size of the graph increases. We believe that this is due to the effects of scale on the serial CPU implementation for graphs with bigger sizes, whereas the overhead of dividing the data for a CUDA-based calculation was more of a detriment for smaller datasets.

Table 2. Tabular view of the speedups

Graph Name	#Nodes	#Edges	Time - GPU(sec)	Time - CPU (sec)	Speedup
Movie Reviews	2,625	99,999	0.187	23.0	**123x**
CalTech Facebook	769	33,312	0.015	1.6	**102x**
Oklahoma Facebook	17,425	1,785,056	0.547	9,828.0	**17972x**
Georgetown Facebook	9,414	851,278	0.187	1,320.0	**7040x**
UNC FB Network	18,163	1,533,602	1.874	13,492.0	**7196x**
Princeton FB Network	6,596	586,640	0.094	495.0	**5280x**
Saket Protein	5,492	40,332	0.109	107.0	**798x**
SPI - Human.pin	8,776	35,820	0.078	263.0	**3366x**
Wiki Vote	7,115	103,689	0.266	137.0	**516x**
Gnutella P2P	10,874	39,994	0.172	681.7	**3966x**
AstroPh Collab	18,772	396,160	0.344	2,218.7	**6454x**

5 Future Work

One facet of this work that we hope to discover more about in the future is the scalability of the CUDA Super Fruchterman-Rheingold layout algorithm. We were fortunate to have large datasets available from the SNAP library, but given the 666x speedup obtained in the largest dataset tested, which contains 82,163 nodes, we have yet to determine the limits of the algorithm. It is clear that given the restricted amount of memory on today's GPUs (about 1 GB on high-end cards) that eventually memory will become a bottleneck, and we have yet to determine where that limit occurs. There are obviously interesting networks that contain millions of nodes, and we hope to be able to provide significant speedup for those situations as well. Unfortunately, the Windows Presentation Foundation code which paints a graph on the visualization window proved to be the weakest link in the visualization of multi-million node

graphs. On large enough graphs, the WPF module will generate an out-of-memory error or simply not terminate. Overcoming this WPF shortcoming would be the first step towards finding the bounds of the Super Fruchterman-Rheingold algorithm.

Additionally, there are other algorithms within NodeXL that could benefit from GPU-based parallelization. The Harel-Koren Fast Multiscale algorithm [2] is another heavily-used layout algorithm in data visualization that is currently implemented sequentially. Finally, the speedup achieved using the eigenvector centrality metric also bring other vertex metrics to our attention. In particular, *Closeness* and *Betweenness* centrality are two measures that are used frequently in data visualization, and have yet to be implemented using CUDA.

6 Conclusions

NodeXL is a popular data visualization tool used frequently by analysts on typical desktop hardware. However, datasets can be thousands or millions of nodes in size, and a typical desktop CPU does not contain enough processing cores to parallelize the various algorithms contained in NodeXL sufficiently to execute in a reasonable timeframe. Nvidia's CUDA technology provides an intuitive computing interface for users to use the dozens of processing elements on a typical desktop GPU, and we applied the technology to two algorithms in NodeXL to show the benefit that the application can gain from these enhancements.

The results obtained from running these enhancements were impressive, as we observed a 79x - 804x speedup in our implemented layout algorithm. Further, the 102x - 17,972x speedup gained by parallelizing the Eigenvector centrality metric across GPU cores indicates that when a CUDA-capable GPU is available, it should be used to perform centrality metrics. Such improvement in these execution times brings the visualization of data previously infeasible on a standard desktop into the hands of anyone with a CUDA-capable machine, which includes low-to-middle end GPUs today. While we have by no means maxed out the parallelization possibilities in the NodeXL codebase, we hope to have provided enough preliminary results to demonstrate the advantage that CUDA technology can bring to data visualization.

Acknowledgements

We would like to acknowledge Dr. Amitabh Varshney and Dr. Alan Sussman for their guidance through the course of this project. We are grateful to Microsoft External Research for funding the NodeXL project at University of Maryland.

References

1. Fruchterman, T., Reingold, E.: Graph Drawing by Force-directed Placement. Software – Practice And Experience 21(11), 1129–1164 (1991)
2. Harel, D., Koren, Y.: A Fast Multi-Scale Algorithm for Drawing Large Graphs. Journal of Graph Algorithms and Applications 6(3), 179–202 (2002)

3. Hansen, D., Shneiderman, B., Smith, M.: Analyzing Social Media Networks with NodeXL: Insights from a Connected World. Morgan Kaufmann, San Francisco
4. Perer, A., Shneiderman, B.: Integrating Statistics and Visualization: Case Studies of Gaining Clarity During Exploratory Data Analysis. In: ACM SIGCHI Conference on Human Factors in Computing Systems, pp. 265–274 (2008)
5. Harish, P., Narayanan, P.J.: Accelerating large graph algorithms on the GPU using CUDA. In: Proc. 14th International Conference on High Performance Computing, pp. 197–208 (2007)
6. Centrality – Wikipedia, http://en.wikipedia.org/wiki/Centrality
7. Hadoop, http://wiki.apache.org/hadoop/
8. Social Network Analysis, http://en.wikipedia.org/wiki/Social_network#Social_network_analysis
9. Sharma, P., Khurana, U., Shneiderman, B., Scharrenbroich, M., Locke, J.: Speeding up Network Layout and Centrality Measures with NodeXL and the Nvidia CUDA Technology. Technical report, Human Computer Interaction Lab, University of Maryland College Park (2010)