Irregular Graph Algorithms on Parallel Processing Systems

George M. Slota

Research Interests and Motivations

My thesis work focuses on designing new parallel approaches for analyzing large real-world networks. This area of research specialization lies at the interface of high performance computing and big data analytics. What makes network analysis so interesting is the variety of data that can be expressed using graph abstractions. A graph representation fits naturally for biological networks at various scales, intercommunication in social networks, visual and geospatial relationships, neural activity in the brain, and numerous other sources of data within the scientific realm. The computational difficulties associated with analyzing real-world graphs is widely recognized, and it is correspondingly listed as one of DARPA's 23 toughest mathematical challenges [1]. The high complexity, scale, and variation of graph-structured data poses an immense challenge in the design of techniques to study and derive insight from such data. Therefore, as we build more powerful supercomputers, it is important to understand how we can efficiently solve graph-theoretic problems on modern hardware. A 50$\times$ performance gap is sometimes observed between estimated and real running times due to programming model overhead, poor abstraction and subroutine choices, and sub-optimal data layouts. HPC platforms with manycore accelerators, such as GPUs or the Intel Xeon Phi, pose an entirely different set of challenges for algorithm designers to overcome. My primary research goals focus on tackling these problems through the optimization of graph analytics at all levels of hardware architecture, from thread to core to processor to single-node to multi-node to system-level scale. The fact that graph-structured data is so universal means that this research is useful to a large collection of data-intensive problems within the social and physical sciences.

This abstract will describe some of the graph problems I've worked on throughout my doctoral program, their implementation, and their performance results relative to the prior state-of-the-art.

Subgraph Counting and Minimum Weight Path Finding

Subgraph isomorphism and its variants are fundamental graph analysis methods used to identify latent structure in complex data sets. Subgraph counting is a computationally intensive (NP-complete) problem, with the naïve algorithm runing in $O(n^k)$ time, where n is the number of vertices in the network and k is the number of vertices in the subgraph. The color-coding technique [2] can be used to determine counts of non-induced tree-structured subgraphs in $O(m \cdot 2^k \cdot e^k \cdot \frac{\log 1/\delta}{\epsilon^2})$ time, where m is the number of edges in the graph, and δ and ϵ are confidence and error parameters, respectively. Color-coding can also be applied to the NP-hard problem of finding the minimum-weight path of a given length in a weighted graph [3].

Recently, I introduced a new shared-memory parallel implementation of the color-coding based subgraph counting algorithm called FASCIA [4]. Through abstracting the key dynamic programming phase of color-coding, FASCIA achieved **several orders-of-magnitude speedup and memory reduction** relative to prior art, and this allowed subgraph analysis of graphs larger than previously possible. Figure 1 (left) shows the overall execution time for several input templates on a single 16 total core Sandy Bridge compute node. More recent work builds on the prior FASCIA implementation with distributed-memory parallelization [5], allowing scaling on modest clusters to analyze multi-billion edge networks. Figure 1 (right) demonstrates the performance of distributed FASCIA on two such large networks. FASCIA's framework was also extended to find minimum simple weighted paths for an edge-weighted network [10], an important problem for analyzing biological interaction networks. This implementation was called FASTPATH.

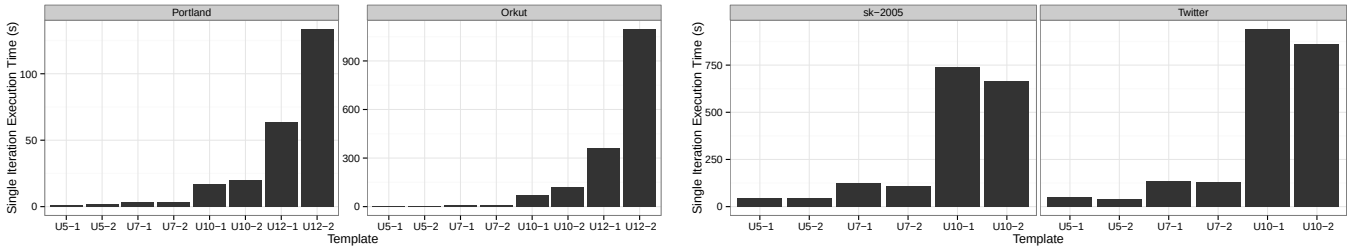

Figure 1: Single iteration execution times for several template sizes running in shared memory on million edge networks (left) and in distributed memory on billion edge networks (right).

Preliminary open-source versions of both FASCIA and FASTPATH are currently available online[1,2]. Future work is underway for expanding FASCIA to to be able to perform subgraph counting in a system-scale distributed environment. Subgraph counting has never before been completed on graphs at this scale, and it can provide novel insight into the structure of massive real-world datasets. Additionally, the color-coding subgraph counting algorithm can essentially be reduced to an ordered and iterative set of matrix-matrix multiplications. This would allow for efficient SIMD vectorization and acceleration on both Xeon Phi coprocessors and GPUs. Since Xeon Phis and GPUs are utilized in a substantial portion of the world's fastest supercomputers, fully exploiting this additional computational power is another avenue for further scaling.

[1] `http://fascia-psu.sourceforge.net/`

[2] `http://sourceforge.net/projects/fastpath-psu/`

Graph Connectivity

Determining the connectivity and component decomposition of a graph is one of the most basic problems in network science. My dissertation work has considered specifically the problems of determining the weakly (WCC) and strongly connected components (SCC) of a directed graph and the connected (CC) and biconnected components (BiCC) of an undirected graph.

I developed the *Multistep Method* for the problems of CC, WCC, and SCC detection in large real-world graphs, such as online social networks and web crawls, using current shared-memory multicore platforms [11]. It utilizes variants of FW-BW [12] and Orzan's coloring methods [13] in subroutines. It minimizes synchronization and avoids use of fine-grained locking. The breadth-first search (BFS) subroutine incorporates several recently-identified optimizations for low-diameter graphs and multicore platforms, and demonstrated a mean traversal rate of 1.4 GTEPS across a suite of test graphs. Multistep is also **faster on average by 1.9× and up to 8.9×** (on the most difficult problem considered) than the state-of-the-art Hong et al. method [14]. Figure 2 (left) gives the speedups relative to Tarjan's algorithm (optimal serial algorithm) for the Hong et al. method and Multistep on a 16 core Sandy Bridge platform with graphs of varying scale up to 2 billion edges. Similar speedups are seen with CC and WCC, which are omitted for brevity.

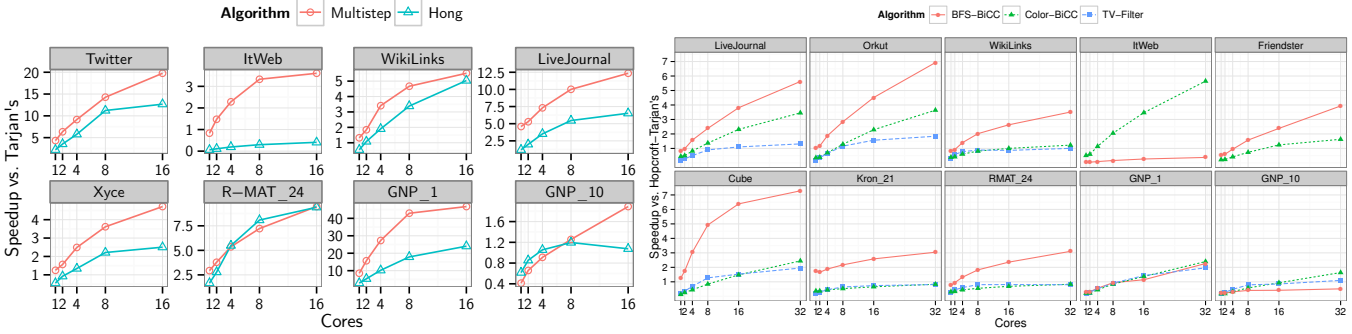

Figure 2: Speedup relative to Tarjan's algorithm for Multistep and Hong et al. (left) and speedup relative to the Hopcroft-Tarjan algorithm for the new BiCC algorithms and TV-Filter on real and synthetic test graphs from various sources [7, 15–18].

Utilizing the Multistep BFS and color propagation subroutines, two new algorithms were proposed for parallel biconnected components decomposition [19]. These algorithms showed considerable speedup to the serial Hopcroft-Tarjan [20] algorithm. A comparison of the BFS-based (BFS-BiCC) and coloring-based (Color-BiCC) algorithms is given by Figure 2 (right) with speedups relative to the Hopcroft-Tarjan serial algorithm. Also included was the state-of-the-art TV-Filter algorithm [21] based on the parallel Tarjan-Vishkin algorithm [22].

Approaches for Portable Manycore Optimizations

As the level of parallelization increases, it become more difficult to create an even distribution of work among processing elements for irregular graphs. As such, a focus of ongoing research is to create a general approach for graph algorithms that is effective on new many-core architectures and heterogeneous environments. The recently developed Kokkos library [23] is an example of a recent paradigm in scientific computing, the development of frameworks for *write-once-run-anywhere* code. Ideally, algorithms implemented in such a framework should be resistant to the current trends in hardware (e.g. increasing parallelism and changing memory hierarchies) by exploiting wide parallelization, minimal synchronization, and localized memory accesses whenever possible.

As part of my dissertation work, I noted that a number of graph algorithms follow a tri-nested loop structure. By performing some general set of optimizations, I demonstrated that the performance of graph algorithms fitting the aforementioned structure could benefit over a baseline parallelization [24]. One critical optimiziation was collapsing the inner two loops to exploit higher parallelizability; a technique called the Manhattan Collapse. Figure 3 gives the performance in GTEPS for SCC decomposition of various graphs on K20 and K40 GPUs, Sandy Bridge CPU (SNB), and Knights Corner Xeon Phi (KNC) using OpenMP parallelization (OMP), a local variant of the Manhattan Collapse (ML), a global variant of the Manhattan Collapse (MG), and baseline parallelization (B). We see that the local Manhattan Collapse performs best for GPU, greatly outperforming CPU on the most difficult test instance (DBpedia) by a factor of 3.25×.

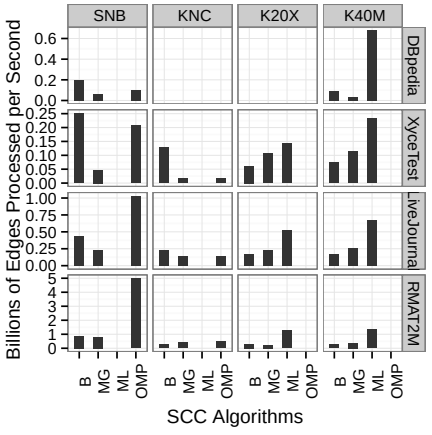

Figure 3: Performance in GTEPS for implementations of Multistep SCC using various parallelization strategies on various architectures with multiple test graphs.

Small-world Graph Partitioning

There have recently emerged several online repositories that host representative real-world graphs with up to billions of vertices and edges and new open-source and commercial distributed graph processing frameworks targetted at analyzing such graphs. These graphs are characterized by a low diameter and skewed vertex degree distributions and are informally referred to as *small-world* or *power-law* graphs. The graph processing frameworks use different I/O formats and programming models, but all of them require an initial vertex and edge partitioning for scalability in a distributed-memory setting.

Due to its fast speed and scalability, high quality results, and simple implementation, the label propagation algorithm for community detection [25] has been utilized for partitioning such large-scale small-world graphs in several studies. As part of my dissertation work, I developed PuLP: Partitioning using Label Propagation [26]. PuLP exploits a weighted label propagation approach with iterative balance and refinement stages to perform multiple constraint (vertices and edges per part) and multiple objective (total edge cut and max edge cut per part) partitioning. All other label propagation work and most well-known partitioners only handle single constraints (either edges or vertices) and single objectives (usually total edge cut or communication volume).

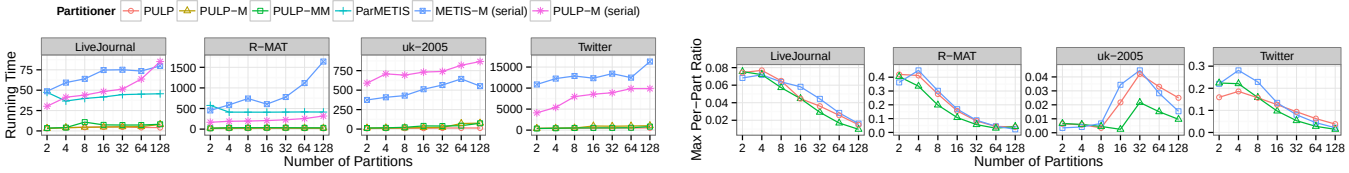

Figure 4: Execution times on various networks for METIS, ParMETIS, and various PuLP variants (single constraint single objective, single objective multi-constraint, multi-objective multi-constraint), and max per-part edge cut for multi-constraint METIS and multi-constraint multi-objective PuLP.

To demonstrate the efficacy of PuLP, a comparison was done to the state-of-the-art multilevel k-way partitioning method in METIS and ParMETIS [27]. The multiple constraint version of both the codes were used. Figure 4 (left) gives the execution time results of PuLP with several configurations in comparison to METIS and ParMETIS and Figure 4 (right) gives the total edge cuts for these partitioners. Over a wide range of partition counts (2-128) and with fixed edge and vertex balance constraints, PuLP is comparable or better than METIS with the most commonly-used quality measures (edge cut). For the secondary objective (maximum edge cut per partition), PuLP gives consistently better quality results. The main advantage of our approach is the relative efficiency improvement: for instance, to partition the 1.8 billion edge Slovakian domain (.sk) crawl [8], **our approach uses 7.5× less memory and is 16× faster than METIS**. PuLP takes less than a minute on a single compute node to generate 128 partitions of this graph, while satisfying both the vertex and edge balance constraints.'

Ongoing work with PuLP includes an optimized distributed-memory version, which **allows partitioning of graphs up to trillions of edges in size in only a few minutes**. No other available partitioner is able to produce quality partitions of graphs of this scale.

Distributed Graph Layout

Ongoing dissertation work is examining PuLP partitioning in the context of a *distributed graph layout* [28]. This layout strategy includes both partitioning as well as intra-part vertex orderings, which can improve computation times through increasing spatial locality of memory accesses. We compare PuLP to METIS and random partitioning and our ordering strategy (DGL) to Reverse Cuthill-McKee (RCM) [29] and random orderings. Figure 5 gives speedups in communication times for the partitioning strategies relative to random (top) and speedups in computation times for the ordering strategies relative to random (bottom) for a distributed PageRank implementation executing on 16 nodes. We observe our strategies to give comparable or better speedups relative to the other approaches. Additionally, the preprocessing times for PuLP and DGL are much quicker than METIS and RCM, respectively. However, the tradeoffs between different partitioning and ordering strategies is complex and not entirely yet understood, so an analysis of the underpinnings determining these tradeoffs is anticipated for future work. It is hoped to develop partitioning (and eventually ordering) strategies tailored to the requirements of various individual distributed graph analytics processing various graph topologies.

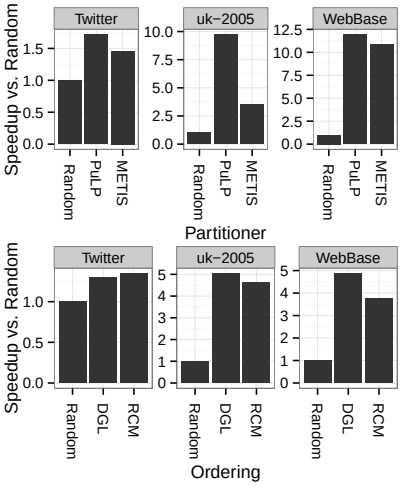

Figure 5: Speedups in PageRank's communication (top) and computation (bottom).

Putting it All Together: Large-Scale Graph Analysis

Using the generalizable techniques and optimizations I gleaned from my prior work in shared-memory algorithm implementation and distributed memory graph layout strategies, I recently implemented several graph analytics for processing at large scale, including algorithms for connectivity measurements, k-core finding, community detection, as well as PageRank and other vertex centrality measures [30]. To test these implementations, I analyzed the largest publicly available real-world graph, the 2012 Web Data Commons hyperlink graph (`http://webdatacommons.org/hyperlinkgraph/`). This analysis produced novel information about the web crawl, including explicit per-vertex centrality measurements and community structure. As shown in Figure 6, we found the community size distribution seemed to follow that of a power law with an apparent heavy tail, which is a similar distribution observed with other connectivity and centrality measurements [31].

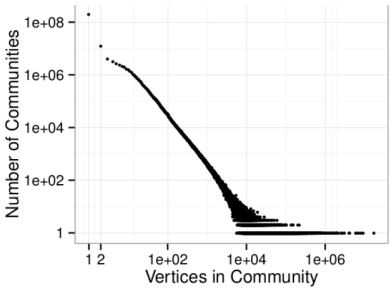

Figure 6: Frequency plot of community structure.

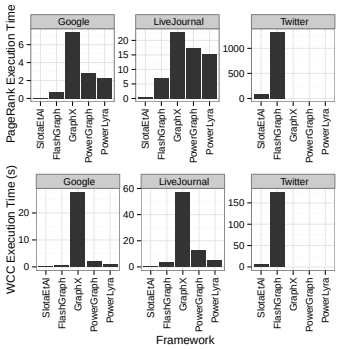

Figure 7: Framework execution time comparison.

I also sought to use this work to provide an informed commentary on ease of algorithm implementation and overall performance for large scale end-to-end graph analytics on high-end (HPC) systems. Using the hyperlink graph, end-to-end execution time for I/O, preprocessing, and all analytics was **under 20 minutes** on 256 nodes of the *Blue Waters* system. Figure 7 gives a comparison of my PageRank (top) and WCC (bottom) implementations to several popular frameworks (GraphX, PowerGraph, PowerLyra, FlashGraph) on smaller graphs in a 16 node test. Smaller-scale testing was necessary as all of the framework except for FlashGraph encountered scalability challenges with larger graphs. The average speedup for my code across all graphs relative to the frameworks is considerable, as a **one to two order-of-magnitude speedup** is noted for both WCC (97×) and PageRank (37×). My implementations are not needlessly complex either; each analytic is implemented in only one to two hundred lines of C code. This goes against an accepted notion in a large part of the graph analytics community that graph processing frameworks are a much better tool for this level of work.

[1]Defense Advanced Research Projects Agency, "Darpa mathematical challenges," Tech. Rep. BAA-07-68, DARPA, 2008.
[2]N. Alon, R. Yuster, and U. Zwick, "Color-coding," *J. ACM*, vol. 42, no. 4, pp. 844–856, 1995.
[3]J. Scott, T. Ideker, R. M. Karp, and R. Sharan, "Efficient algorithms for detecting signaling pathways in protein interaction networks," *Journal of Computational Biology*, vol. 13, no. 2, pp. 133–144, 2006.
[4]G. M. Slota and K. Madduri, "Fast approximate subgraph counting and enumeration," in *Proc. 42nd Int'l. Conf. on Parallel Processing (ICPP)*, pp. 210–219, 2013.
[5]G. M. Slota and K. Madduri, "Complex network analysis using parallel approximate motif counting," in *28th IEEE International Parallel and Distributed Processing Symposium (IPDPS14)*, 2014.
[6]Network Dynamics and Simulation and Science Laboratory, "Synthetic data products for societal infrastructures and proto-populations: Data set 1.0," Tech. Rep. NDSSL-TR-06-006, Virginia Polytechnic Institute and State University, 2006.
[7]"Stanford large network dataset collection." `http://snap.stanford.edu/data/index.html`, last accessed July 2014.
[8]P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "UbiCrawler: A scalable fully distributed web crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.
[9]M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in Twitter: The million follower fallacy," in *Proc. Int'l. Conf. on Weblogs and Social Media (ICWSM)*, 2010.
[10]G. M. Slota and K. Madduri, "Parallel color-coding," *to appear in Parallel Computing, Systems & Applications*, 2015.
[11]G. M. Slota, S. Rajamanickam, and K. Madduri, "BFS and Coloring-based parallel algorithms for strongly connected components and related problems," in *Proc. Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, 2014.
[12]L. Fleischer, B. Hendrickson, and A. Pinar, "On identifying strongly connected components in parallel," in *Parallel and Distributed Processing*, vol. 1800 of *LNCS*, pp. 505–511, Springer Berlin Heidelberg, 2000.
[13]S. Orzan, *On Distributed Verification and Verified Distribution*. PhD thesis, Vrije Universiteit, 2004.
[14]S. Hong, N. C. Rodia, and K. Olukotun, "On fast parallel detection of strongly connected components (SCC) in small-world graphs," in *Proc. Supercomputing*, 2013.
[15]T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–25, 2011.
[16]J. Kunegis, "KONECT - the Koblenz network collection." `konect.uni-koblenz.de`, last accessed July 2014.
[17]K. Madduri and D. A. Bader, "GTgraph: A suite of synthetic graph generators." `http://www.cse.psu.edu/~madduri/software/GTgraph/`, last accessed Aug 25, 2014.
[18]D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, "Graph partitioning and graph clustering, 10th DIMACS implementation challenge workshop," *Contemporary Mathematics*, vol. 588, 2013.
[19]G. M. Slota and K. Madduri, "Simple parallel biconnectivity algorithms for multicore platforms," in *IEEE International Conference on High Performance Computing*, 2014.
[20]J. Hopcroft and R. Tarjan, "Efficient algorithms for graph manipulation," *CACM*, vol. 16, no. 6, pp. 374–378, 1973.
[21]G. Cong and D. A. Bader, "An experimental study of parallel biconnected components algorithms on symmetric multiprocessors (SMPs)," in *Proc. Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, 2005.
[22]R. E. Tarjan and U. Vishkin, "An efficient parallel biconnectivity algorithm," *SIAM Journal on Computing*, vol. 14, no. 4, pp. 862–874, 1985.
[23]H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, 2014.
[24]G. M. Slota, S. Rajamanickam, and K. Madduri, "High-performance graph analytics on manycore processors," in *International Parallel & Distributed Processing Symposium (IPDPS)*, 2015.
[25]U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, no. 3, 036106, 2007.
[26]G. M. Slota, K. Madduri, and S. Rajamanickam, "PULP: Scalable multi-objective multi-constraint partitioning for small-world networks," in *IEEE International Conference on Big Data*, 2014.
[27]G. Karypis and V. Kumar, "MeTis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. version 5.1.0." `http://glaros.dtc.umn.edu/gkhome/metis/metis/download`, last accessed July 2014.
[28]G. M. Slota, K. Madduri, and S. Rajamanickam, "Distributed graph layout for scalable small-world network analysis," in *under review*, 2015.
[29]E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 1969 24th Nat'l. Conf.*, ACM '69, (New York, NY, USA), pp. 157–172, ACM, 1969.
[30]G. M. Slota, S. Rajamanickam, and K. Madduri, "Supercomputing for web graph analytics," 2015.
[31]R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer, "Graph structure in the web — revisited: A trick of the heavy tail," in *WWW'14*, 2014.