AN INTERACTIVE ENVIRONMENT TO MANIPULATE LARGE GRAPHS

John R. Gilbert[†*], *Viral Shah*[†]

UC Santa Barbara
Dept. of Computer Science

Steven Reinhardt

Silicon Graphics Inc

ABSTRACT

Interactive environments such as MATLAB and STAR-P have made numerical computing tremendously accessible to engineers and scientists. They allow people who are not well–versed in the art of numerical computing to nonetheless reap the benefits of numerical computing. The same is not true in general for combinatorial computing. Often, many interesting problems require a mix of numerical and combinatorial computing. Tools developed for numerical computing – such as sparse matrix algorithms – can also be used to develop a comprehensive infrastructure for graph algorithms. We describe the current status of our effort to build a comprehensive infrastructure for operations on large graphs in an interactive parallel environment such as STAR-P.

***Index Terms*—** Combinatorial scientific computing, Large graphs, Pattern discovery, Interactive

1. INTRODUCTION

High performance applications increasingly combine numerical and combinatorial algorithms. Past research on high performance computation has focused mainly on numerical algorithms, and we have a rich variety of tools for high performance numerical computing. On the other hand, few tools exist for large–scale combinatorial computing.

Our goal is to build a general set of tools to allow scientists and engineers develop applications using modern numerical and combinatorial tools with as little effort as possible. Sparse matrix computations allow structured representation of irregular data structures, decompositions, and irregular access patterns in parallel applications.

Sparse matrices are a convenient way to represent graphs. Since sparse matrices are first class citizens in MATLAB and many of its parallel dialects [1], it is natural to use the duality between sparse matrices and graphs to develop a rich infrastructure for numerical and combinatorial computing.

Graphs are a general purpose abstraction useful in a wide variety of disciplines. In the signal processing domain, graphs find use in sensor network topology, sensor localization, knowledge discovery and tracking. They are also useful in acoustics, speech and signal processing applications.

[*]This author's work was partially supported by Silicon Graphics Inc.

[†]These authors' work was partially supported by the Air Force Research Laboratories under agreement number AFRL F30602-02-1-0181 and by the Department of Energy under contract number DE-FG02-04ER25632.

2. GRAPH ALGORITHMS AND PATTERN DISCOVERY TOOLBOX

Our toolbox is designed from the outset to run interactively with terascale graphs via STAR-P. All the components we provide in our toolbox are scalable to tens/hundreds of processors. High performance and interactivity are the salient features of our toolbox. This is also why we set out to write a new toolbox rather than use existing ones such as Ed Scheinerman's MATGRAPH, or a library such as BOOST.

Our graph algorithms and pattern discovery toolbox contains routines for graph structure queries, graph algorithms, graph partitioning and clustering, graph generators, graph visualization, and other utilities. We will discuss each of these in some detail in the next section.

Sparse matrices are used to store the adjacency relations in graphs. We may augment this basic data structure to include other ancillary data. This allows all sparse matrix operations in MATLAB to be used for graph operations.

2.1. Graph queries and operations

These are some basic operations to query common graph parameters. All these operations are extremely fast.

- *Simple graph queries*:
  Our toolbox provides a number of routines to query simple graph properties. For instance, `nverts` gives the number of vertices in a graph. Vertices with no edges are also acceptable, and counted by `nverts`. `nedges` is the number of edges in a graph. This is the number of nonzeros in the corresponding sparse matrix for a directed graph. In the case of an undirected graph, each edge gets counted twice, and hence the number of edges is half the number of nonzeros. `nlayers` gives the number of layers of multiple edges in a multigraph. `degree` returns the number of edges incident on a vertex in an undirected graph. For a directed graph,

`indegree` gives the number of directed edges incident on a vertex and `outdegree` gives the number of directed edges leaving a vertex. `degreehist` can be used to plot a histogram of vertex degrees. This is useful to get a rough idea about the distribution of edges in the graph. For instance, a graph may have vertex degrees from different distributions. Our graph generators allow the construction of several commonly used graphs.

- *Neighbor queries*:
  The routine `neighbors` returns the adjacencies of a vertex or vertex set, whereas `commonnbrs` returns the common adjacencies of two vertices. `reach` returns the reachable vertices from initial vertex(es). Repeatedly using `reach` traverses the graph in a breadth–first search order. This operation is implemented as sparse matrix multiplication. Several bfs searches from different starting points may also be computed simultaneously using sparse matrix–matrix multiplication [2].

- *Simple graph operations*:
  `grdistance` returns the distance in edges from one set of vertices to another. `grperm` permutes a graph. `eccentricity` returns the eccentricity of a vertex (max distance to another). `grintersect` returns the intersection of edges of two graphs. The result is a graph containing edges that exist in both graphs. This is the same as element wise multiplication of the sparse matrices representing these graphs. `grunion` gives the union of edges of two graphs. This is the same as the sum of the sparse matrices representing these graphs. The difference between two graphs can be computed with `grdiff`, which results in a graph consisting of edges that exist in one graph but not the other.

- *Sparse matrix multiplication*:
  Matrix multiplication is an important primitive for a number of signal processing applications. STAR-P provides an efficient implementation of multiplication of large sparse matrices (graphs) in parallel [2]. We use sparse matrix multiplication heavily, for instance in neighbor queries, reachability and visualization. However, we believe that it is an important operation by itself and may find use in many other applications in signal processing.

2.2. Graph algorithms and global structure of graphs

We provide some efficient implementations of parallel graph algorithms:

- `toposort` - Compute the topological ordering of a directed graph. The result is a permutation vector `perm` such that the reordering of the graph $G(perm, perm)$ is lower triangular [3]. If no such permutation exists, the empty set `[]` is returned.

- `bfstree` - Compute the breadth-first spanning tree of the graph. Often the purpose of breadth–first search is to compute a breadth first search tree, which contains node–parent relationships. We implement this using sparse matrix–vector multiplication.

- `components` - Find the connected components of an undirected graph. We use the PRAM algorithm described by Shiloach and Vishkin [4]. The output labels each vertex in the graph by the component it belongs to.

- `mis` - Find a maximal independent set in the graph. We use a probabilistic algorithm described by Luby [5]. An independent set is a set of vertices such that no two of the vertices are neighbors. If no more vertices can be added to this set without violating the first condition, then such an independent set is called a maximal independent set. This is useful in many cases, and we have used it successfully for clustering in version 1.1 of the SSCA #2 graph analysis benchmark [6, 7].

- `betcentral` - Find the betweeness centrality [8] metric for vertices in a graph. Betweenness centrality is commonly used in social network analysis. It may be useful in analyzing vulnerabilities in a large sensor network.

2.3. Graph partitioning and clustering

These are some parallel graph partitioning algorithms that we provide currently in our toolbox. We plan to signficantly expand the clustering capabilities in the near future, based on our work on version 1.1 of the SSCA #2 graph analysis benchmark [7].

We believe that graph partitioning and clustering algorithms provide basic building blocks to design algorithms for the operation of large sensor networks. Especially, we have tools such as `geopart` which can scale effortlessly to a network with hundreds of thousands of nodes.

- `geopart` - Partition a graph using geometric partitioning [9]. This method works especially well for graphs that arise out of discretizations of partial differential equations. Geometric partitioning scales very well to large problems – but its limitation is that it requires co–ordinates for graph vertices. In many cases, graph vertices may not be available. The problem of finding useful graph embeddings is by itself an interesting research problem.

- `specpart` - Use a spectral method for partitioning. We currently use `eigs` to implement our spectral par-

titioning. However, this is not practical for extremely large graphs.

- `specdice` - Use spectral partitioning recursively to get a multi–way partition.
- `cutsize` - Find or count edges cut by a partition. In general, a good graph partitioning algorithm produces a smaller edge cut.

2.4. Graph generators

Researchers often need a variety of graphs to model their applications. We provide a few graph generators, and plan to keep adding more in the future.

- `powergraph` - Generate a directed graph with a power–law degree distribution.
- `powergraphsym` - Generate an undirected graph with power–law degree distribution.
- `grid5` - A 2D square 5-point mesh.
- `grid9` - A 2D square 9-point mesh.
- `grid3d` - A 3D cubical mesh.
- `grid3dt` - A 3D tetrahedral mesh.
- `clique` - Generate a clique.
- `ssca2graph` - Generate the graph for ver 1.1 of the SSCA#2 graph analysis benchmark.
- `rmat` - A general purpose graph generator that may be used to generate realistic graphs [10].

2.5. Visualization and graphics

Visualization is a key part of any interactive environment. We provide some visualization tools for parallel graphs. We are also working on methods for computing meaningful graph embeddings. Figure 1 shows an embedding of the SSCA #2 graph in 3 space. We are working on generalizing such techniques and making them available for users of our toolbox.

`spyy` is a routine in our toolbox which plots the sparsity pattern into buckets. For an example, see Figure 2. This routine is a good example for visualization of large datasets. It does the entire computation on the parallel computer and only transfers the image to the frontend for viewing.

3. CONCLUSION

We have used our graph toolbox successfully to implement several versions of the SSCA # 2 graph analysis benchmark. We have manipulated extremely large graphs while working with this benchmark – for instance, we ran an older version of the benchmark on a graph with 134 million vertices. We have also manipulated graphs with 400 millon vertices and 4 billion edges. We have also used this toolbox to implement the Algebraic Multigrid Method (AMG) in parallel. Apart from this, we are using our graph algorithms and pattern discovery toolbox in conjunction with the numerical computing facilities in STAR-P to speed up an application in computational ecology.

Fig. 1. Example visualization: Rendered SSCA#2 v1.1 graph

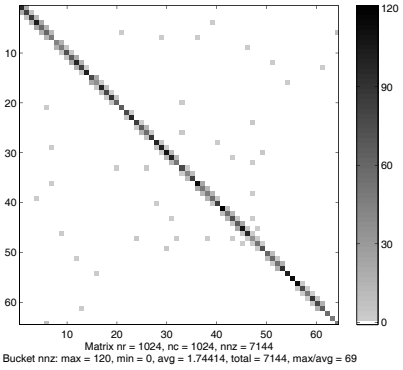

Fig. 2. Example visualization: spyy plot

4. REFERENCES

[1] Viral Shah and John R. Gilbert, "Sparse matrices in Matlab*P: Design and implementation.," in *HiPC*, Luc Bougé and Viktor K. Prasanna, Eds. 2004, vol. 3296 of *Lecture Notes in Computer Science*, pp. 144–155, Springer.

[2] Christopher Robertson, "Sparse parallel matrix multiplication," *M.S. Project, Department of Computer Science, UCSB*, 2005.

[3] Iain S. Duff and J. K. Reid, "Algorithm 529: Permutations to block triangular form [f1].," *ACM Trans. Math. Softw.*, vol. 4, no. 2, pp. 189–192, 1978.

[4] Yossi Shiloach and Uzi Vishkin, "An o(log n) parallel connectivity algorithm.," *J. Algorithms*, vol. 3, no. 1, pp. 57–67, 1982.

[5] Michael Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM J. Comput.*, vol. 15, no. 4, pp. 1036–1053, 1986.

[6] D. A. Bader, J. R. Gilbert, J. Kepner, D. Koester, E. Loh, K. Madduri, B. Mann, and T. Meuse, "HPCS SSCA #2, graph analysis," 2006.

[7] Steven Reinhardt, John R. Gilbert, and Viral Shah, "High performance parallel graph algorithms," in *Workshop on state of the art in scientific and parallel computing (In review)*, 2006.

[8] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology, 25(2):163–177*, 2001.

[9] John. R. Gilbert, Gary L. Miller, and Shang-Hua Teng, "Geometric mesh partitioning: Implementation and experiments," *SIAM Journal on Scientific Computing*, vol. 19, no. 6, pp. 2091–2110, 1998.

[10] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, and Christos Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication.," in *PKDD*, Alípio Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, Eds. 2005, vol. 3721 of *Lecture Notes in Computer Science*, pp. 133–145, Springer.