White Paper on

"Primitives and Patterns for Parallel Graph Computation"

Technical POC:

Prof. John R. Gilbert
Computer Science Department
5109 Harold Frank Hall
University of California
Santa Barbara, CA 93106-5110
Telephone: 805-448-6438
gilbert@cs.ucsb.edu

Administrative POC:

Mr. Ned Nash
Computer Science Department
2112 Harold Frank Hall
University of California
Santa Barbara, CA 93106-5110
Telephone: 805-893-5283
ned@cs.ucsb.edu

May 19, 2010

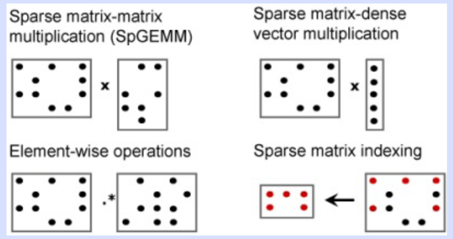
Primitives and Patterns for Parallel Graph Computation

UCSB

STATUS QUO

Computation with discrete structures does not scale on multicore and future manycore processors

NEW INSIGHTS



Sparse matrix-matrix multiplication (SpGEMM)
Sparse matrix-dense vector multiplication
Element-wise operations
Sparse matrix indexing

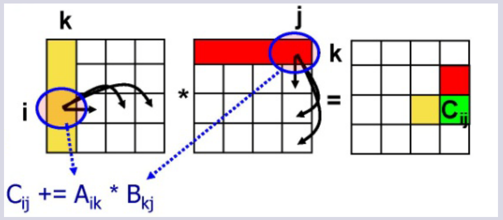
- **Algebraic graph primitives aggregate elementary operations to permit optimization**
- **2D sparse array decomposition can scale beyond 10^3 cores**

MAIN ACHIEVEMENT:

A library of concise, flexible, efficient tools for graph operations

Enables algebraic, visitor, and map/reduce patterns to interoperate cleanly for both edge-based and traversal-based algorithms

Discrete structure computations achieve parallel performance and manycore scaling by use of medium-granularity primitives that encapsulate load balancing, scheduling, and latency issues



HOW IT WORKS:

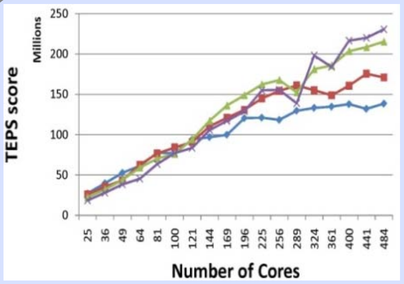
2D compressed sparse block data decomposition supports user-defined edge/vertex/property types and operations

Common framework integrates edge-based, algebraic, and traversal-based patterns with user-defined elementwise unary, binary, n-ary operations

ASSUMPTIONS AND LIMITATIONS:

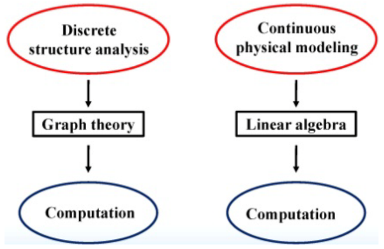
Targets only global address space and shared memory architectures

QUANTITATIVE IMPACT



- **Performance of discrete structure computations scales up as core count increases**
- **Standard primitives relieve programmer of parallel coding burden**

END-OF-PROJECT GOAL



Computational Graph Theory is fully functional as middleware for modern Discrete Structure Analysis

Expressive and flexible middleware enables scalable manycore combinatorial computing.

1. Executive Summary

Computation with combinatorial and discrete structures – graphs, networks, strings, partial orders, etc. – has become ubiquitous in many areas of data analysis and scientific modeling. However, the field of high-performance combinatorial computing is in its infancy, and computations with discrete structures do not scale well on multicore and future manycore processors.

In the mature field of numerical computing, by contrast, programmers possess standard algorithmic primitives, high-performance software libraries, powerful rapid-prototyping tools, and a deep understanding of effective mappings of problems to high-performance computer architectures.
A key challenge of the manycore revolution is to replicate these achievements for combinatorial computing.

We propose to design and implement a library called "Graph BLAS," consisting of concise, flexible, efficient tools for graph operations that can serve as "middleware" for modern combinatorial computing and discrete structure analysis.

We will build on our experience with novel sparse array-based algebraic graph primitives, which aggregate elementary operations to permit optimization and to encapsulate issues of load-balancing, scheduling, and latency tolerance. In addition, our system will support visitor-based traversal primitives and map/reduce primitives in a common framework. We will use an underlying two-dimensional compressed sparse block representation, supporting user-defined edge/vertex/property types and elementary operations, which can allow scaling to thousands of cores. The Graph BLAS will combine the performance and power of our algebraic primitives and 2D sparse block data structures with the expressivity and generality of fully general labeled graphs and visitor/traversal primitives in a coherent framework.

The impact of this work will be that standard primitives can relieve the multicore programmer of much of the burden of parallel coding, while still enabling performance to scale with increasing core count. As the manycore revolution gains momentum, more and more programmers will face the need to use parallelism in combinatorial computations, and this impact will become more and more significant.

2. Innovative Claims

Computation with large combinatorial structures – graphs, strings, partial orders, etc. – has become fundamental in many areas of data analysis and scientific modeling. The field of high-performance combinatorial computing, however, is in its infancy. By way of contrast, in numerical computing we possess standard algorithmic primitives, high-performance software libraries, powerful rapid-prototyping tools, and a deep understanding of effective mappings of problems to high-performance computer architectures.

Linear algebra has played a crucial (one might almost say magical) role as "middleware" between continuous physical models to be computed and the simple arithmetic operations implemented by actual computer hardware. From a mathematical point of view, graph theory occupies a similar position as middleware between models of discrete structures and actual computers. However, for the most part computational graph theory still lacks the above-mentioned standard primitives, software libraries, rapid-prototyping tools, and connections to machine architecture.

We propose a research plan leading to the development of tools and primitives for graph computation that play roles analogous to those of the BLAS (standard primitives), LAPACK and ScaLAPACK (high-performance libraries for key algorithms), and even Matlab (interactive systems for data analysis, exploration, and algorithmic prototyping). The novel features of our proposal include:

- We build on our experience with algebraic primitives for graph algorithms, which aggregate elementary operations to a level that allows optimization and load-balancing internal to the primitive.

- By replacing earlier semiring-operation APIs with higher-level user-specified properties and operators, we achieve the performance benefits of the algebraic computational patterns with a more intuitive and flexible interface.

- We propose a common framework that coherently integrates algebraic, visitor, and map-reduce patterns in a concise library of primitives that will interoperate cleanly for both edge-based and traversal-based algorithms.

- Our implementation strategy is based on an efficient parallel sparse array infrastructure, which uses a 2D compressed sparse block data representation for efficiency and scalability.

- We design and exploit kernels that are efficient in the "hypersparse" realm, where performance is limited by cache and memory bandwidth rather than raw operation count.

- We benchmark and tune our library on a range of graph algorithms and applications.

- Our design focuses on the capabilities of the manycore chips of the future, while achieving performance on today's multicore chips and multisocket shared-memory systems.

Our three-year project plan is centered on the design, implementation, evaluation, and application of a new library called "Graph BLAS", or GBLAS.

3. Detailed Technical Rationale, Approach, and Constructive Plan

What would it mean to complete the analogy between linear algebra as middleware for continuous physical modeling, and graph theory as middleware for analysis of discrete structures? This asks us to reflect on the steps that evolved numerical linear algebra from mathematical linear algebra ("numerical" here means "computational"), and to similarly develop a computational graph theory.

Among the questions we must ask are: What is a good set of primitive representations and operations for computational graph theory? How can such primitives be implemented so that both functionality and performance are portable across a range of modern computers? How should such primitives be presented to the user as APIs, programming patterns, languages, interactive environments? How should we evaluate these tools from the points of view of performance, expressivity, and productivity? Here we propose specific directions to explore for the answers to these questions.

3.1. Primitives

Primitives should supply a common notation to express computations; should have broad scope but fit into a concise framework; should encourage programming at the appropriate level of abstraction and granularity; should scale seamlessly from laptop to supercomputer; and should hide architecture-specific details from their users. We plan to blend a selection of primitives drawn from a number of different sources into a coherent whole for this purpose. An important part of our research will be to continually evaluate the completeness of our chosen primitive set; we expect its exact contents to evolve to a final state over the course of the project.

3.1.1. Sparse array-based primitives

A sparse graph can be represented by a sparse array of edges (indexed by head and tail vertices), which in turn can be considered as a sparse matrix data structure for the graph's adjacency matrix. Linear algebraic operations on this matrix correspond to some kinds of graph operations; for example, multiplying a sparse matrix by a dense vector systematically explores all the neighbors of each vertex. We have achieved [BulGil2008a, BulGil2008b, BFFGL2009] good performance and scalability on parallel graph algorithms by using novel two-dimensional "compressed sparse block" data structures with sequential "hypersparse" array kernels.

The most powerful algebraic primitives include: SpGEMM (sparse matrix-matrix multiplication); SpAdd (sparse matrix-matrix addition); SpMV (sparse matrix-dense vector multiplication); SpRef (selection of a subarray of a sparse matrix); SpAsgn (assignment to a subarray of a sparse matrix) and SpCat (concatenation of sparse arrays), all with the possibility of substituting different objects for numerical array elements and different operations for scalar addition and multiplication. (In fact, these all can be implemented in terms of SpGEMM, though it's not clear this is the most effective way to proceed.) SpGEMM has not been studied extensively by the numerical linear algebra community, although sequential SpGEMM algorithms appear in Matlab [GilMolSch1992, Dav2007] and as far back as Gustavson [Gus1978].

An example of the use of SpGEMM is our algebraic implementation of the SSCA#2 Betweenness Centrality benchmark [BMGSKMK2006]. The inner loop of BC is essentially a breadth-first search from multiple starting vertices. (The number of starting vertices is a tunable parameter.) If A is the transpose of the graph's adjacency matrix, and B is an indicator matrix with a column for each start vertex, then B = A*B corresponds to one step of parallel breadth-first search. The two-dimensional block decomposition of the sparse arrays invisibly encapsulates three levels of parallelism in this algorithm. First, decomposing the columns of B corresponds to doing multiple simultaneous breadth-first searches in parallel. Second, decomposing the rows of B and the columns of A corresponds to parallelism across the frontier vertices of each breadth-first search. Third, decomposing the rows of A corresponds to parallelism across the incident edges of individual high-degree frontier vertices.

Merely identifying the primitive does not solve the problems of load balancing, synchronization, and latency, of course; but it encapsulates them in the implementation of the primitive and thus hides them from the user. We have created an efficient distributed-memory implementation of SpGEMM in our distributed-memory Combinatorial BLAS library. Using this library for a sparse array-based computation of the Betweenness Centrality benchmark yields good scalability up to many hundreds of cores on the TACC Ranger cluster, where it achieves a record of over 300 MTEPS (million edges traversed per second) on an RMAT power-law graph with 8 million vertices [Bul2010].

Gilbert et al. [GilReiSha2007] and Kepner et al. [KepGil2010] have shown that linear algebra over various user-defined semirings (including the familiar (+,*) ring, but also (min,+), (and,or), and others) can be used to implement a wide variety of graph algorithms. However, semiring algebra becomes awkward in some cases, such as Fineman's shortest path tree algorithm [Fin2010]. Here the data dependencies and the structure of the computation are exactly those of the underlying algebraic primitive (SpMV or SpGEMM in this case), but defining the elementary manipulations as semiring scalar operations – while possible – seems unintuitive.

Thus we propose to use the computational patterns of the sparse linear algebraic operations as primitives, rather than defining semiring matrix algebra operations as primitives themselves. This has several advantages: it will simplify unification of the algebraic primitives with other classes of primitives as described below; it will allow arbitrary representations of individual edge and vertex properties; and it will allow arbitrary user-specified operations on individual vertices, edges, and pairs to be invoked within the matrix-algebraic pattern. This approach will also retain the advantages of the purely algebraic primitives in terms of data parallelism, level of abstraction, and aggregation of elementary operations.

3.1.2. Visitor primitives

While algebraic primitives map well to data-parallel edge-based algorithms like breadth-first search, they map less well to priority-directed traversal algorithms like weighted single-source shortest paths or labeled subgraph isomorphism. The visitor-based search patterns of the Boost Graph Library [SieLeeLum2001] and its relatives PBGL [GreLum2005] and MTGL [BHKK2007] are powerful ways to express algorithms that are based (as are many classical sequential algorithms) on depth-first search or cost-based priority searches. A challenge here is that visitor patterns can require fine-grained reasoning about synchronization and race conditions, and may be too fine-grained for a primitive to optimize across a large aggregation of operations. Indeed, depth-first search (for example) is inherently difficult to parallelize; a problem like strongly connected components that is solved sequentially by DFS [Tar1972] is often best solved in parallel by different approaches [FleHenPin2000]. However, the power and utility of visitor-based patterns is undeniable in some situations, so we propose to include at least BfsVisitor (for breadth-first search), DfsVisitor (for depth-first search), and UcsVisitor (for uniform cost search) among our standard primitives.

3.1.3. Map/Reduce primitives
The Map/Reduce (or Hadoop) framework [DeaGhe2008, Whi2009] is powerful for scanning, summarizing, and classifying data, especially in a distributed-storage "cloud" setting. Cohen [Coh2009] has suggested that MapReduce could be appropriate for some edge-based graph computations like cluster coefficient (or triangle counting), but warns that "the prospect of the entire graph traversing the cloud fabric for each MapReduce job is disturbing." With its focus on loosely-coupled distributed systems, and its lack of a straightforward user-visible cost model, MapReduce seems initially somewhat outside the scope of this project. However, we do plan to include Map and Reduce patterns among our primitives for two reasons: First, MapReduce has proved to be a simple framework for many computations, and will probably sometimes be the most natural way to express even tightly-coupled computations. Second, relatively seamless scaling from small (local) to large (distributed) computations is certainly an advantage for any library of computational primitives.

3.1.4. Other primitives
An important part of our project will be to identify a set of lower-level supporting primitives that is as concise and coherent as possible while still being sufficiently expressive for our needs. This will undoubted include PrefixScan [Ble1989] (including segemented scans, and again with user-defined elements and operations); some versions of Set; and some kind of parallel PriorityQueue. We expect also to define a general ExpandFrontier graph primitive, for use in algorithms like delta-stepping shortest path search [MeySan2003] in which a relaxed-priority parallel search replaces the strict priority search of a classical sequential algorithm.

3.2. Integration of primitives
All the primitives – algebraic, visitor-based, and otherwise – will be built on top of a common sparse graph representation based on the compressed sparse block (CSB) data structure of Buluc et al. [BFFGL2009], which decomposes a graph not by vertices but by groups of edges. Representation of individual vertices and edges (including labels, weights, or other properties) is user-defined. The user interacts with the graph primitives by specifying (in sequential code) operations on single elements (vertex, edge, or label) or on pairs or sets of elements (corresponding to scalar semiring ops, scan ops, etc.). We will also strive for a simple interface between our graph primitives and existing (or future) numerical sparse matrix primitives, since numerical matrix computation is useful in graph algorithms such as spectral clustering or importance ranking, and graph computation is useful in numerical sparse matrix applications as well.

3.3. Library design
We plan to package the graph primitives in a "Graph BLAS" library of fairly simple design, building on our experience with the CBLAS library [Bul2010]. Unlike CBLAS or PBGL [GreLum2005], we will at least initially specify a single underlying graph data structure, namely CSB, for performance reasons: we have both analytical and experimental evidence [BulGil2008a, BFFGL2009, Bul2010] that CSB-based algorithms scale better with increasing processor count than vertex-based algorithms, in the range of processor counts that will be available on single-socket systems in the next several years. However, our library will support fully general user-defined element types, vertex and edge labels, and elementwise operations (unlike CBLAS, which assumed an underlying algebraic semiring). Our goal is to produce a fairly spare library API that can be easily adapted or extended to different contexts.

3.4. Target architecture

We will target single-address-space architectures (unlike CBLAS, PBGL, and Star-P [GilReiSha2008a]). We expect that distributed-memory multinode architectures will remain of interest for some very large-scale graph computations, and we hope that this work will form the basis for such distributed-memory systems (whether programmed by message passing or by PGAS), but that is not our focus in this project. We foresee continuing and indeed increasing interest in single-address-space large graph computation, partly due to the impact of manycore chips. Our work will initially be aimed at future manycore single-socket systems; we will however also implement and benchmark our codes on shared-memory multisocket systems.

3.5. Performance evaluation, benchmarking, and tuning

Performance is an essential goal of all parallel programming (otherwise why bother!). We will measure parallel performance in a number of kernel and benchmark computations, and we will also seek opportunities to collaborate with application scientists to use our libraries in real codes. Some kernel computations are listed below.

Computation	**Type**
Degree distribution	Edge-based
Breadth-first search	Edge-based or traversal-based
Betweenness centrality, SSCA#2	BFS-based or traversal-based
Cluster coefficient, triangles	Edge-based
Single-source shortest paths, weighted	Traversal-based
Connected components	Edge-based or traversal-based
Connected components in dynamic graph	Edge-based or traversal-based
Subgraph isomorphism, with labels	Traversal-based
Minimum spanning tree	Edge-based
Strongly connected components, dmperm	Traversal-based
Depth-first search	Traversal-based, difficult
Markov clustering	BFS-based
Spectral clustering, ordering, partitioning	Numerical
Bottleneck, cutset, vulnerability analysis	Traversal-based
Dynamic variants of all of the above	Complex

In tuning a primitive’s performance, a challenge is how to obtain statistics on bandwidth bottlenecks, cache behavior, etc. While developing tools to measure and analyze such statistics is not part of our proposed project, we hope to collaborate closely with performance analysis tool developers; their tools can help our primitives, and our primitives can be good test cases for their tools.

3.6. Further directions

We plan several directions for further exploration, some in future work. We plan early on to develop interactive Matlab and Python interfaces, which will be useful not only for our own debugging and performance evaluation but as the basis for future high-productivity parallel graph programming environments. We plan eventually to extend our sparse graph infrastructure to higher-dimensional sparse arrays, for tensor computation in data analysis [KolBad2009] and applications to computations on multigraphs and time series of graphs. We also plan eventually to develop autotuning technology for our graph primitives, along the lines of those for numerical kernels [FriJoh1998, VudDemYel2005, WOVSYD2009].

4. Comparison with Other Ongoing Research

4.1. Frameworks for parallel graph computation

The Parallel Boost Graph Library (PBGL) [GreLum2005] is a parallel library for distributed memory computing on graphs. It is a significant step towards facilitating rapid development of high performance applications that use distributed graphs as their main data structure. Like the sequential Boost Graph Library [SieLeeLum2001], it has a dual focus on efficiency and flexibility. It relies heavily on generic programming through C++ templates.

Lumsdaine et al. [LGHB2007] observed poor scaling of PBGL for some large graph problems. We believe that the scalability of PBGL is limited due to two main factors. The graph is distributed by vertices instead of edges, which corresponds to a one-dimensional partitioning in the sparse matrix world. We have shown [BulGil2008a] that this approach does not scale to large numbers of cores. We also believe that the visitor paradigm is sometimes too low-level for scalability, because it makes the computation data driven and obstructs opportunities for optimization. Nonetheless, the visitor paradigm can be quite powerful, and as described in Section 3 we plan to integrate parts of it into our primitive library.

The MultiThreaded Graph Library (MTGL) [BHKK2007] was originally designed for development of graph applications on massively multithreaded machines, namely Cray MTA-2 and XMT. It was later extended to run on mainstream shared-memory architectures [BBMW2009]. MTGL is a significant step towards an extendible and generic parallel graph library. It will certainly be interesting to quantify the abstraction penalty it pays due to its generality. As of now, only preliminary performance results are published for MTGL.

The Graph Algorithm and Pattern Discovery Toolbox (GAPDT, later renamed KDT) [GilReiSha2008b] provides both combinatorial and numerical tools to manipulate large graphs interactively. KDT runs sequentially on Matlab or in parallel on Star-P [ShaGil2005], a parallel dialect of Matlab. Although KDT focuses on algorithms, the underlying sparse matrix infrastructure also exposes linear algebraic kernels. KDT, like PBGL, targets distributed-memory machines. Differently from PBGL, it uses operations on distributed sparse matrices for parallelism. KDT provides an interactive environment instead of compiled code, which makes it unique among the frameworks surveyed here. Like PBGL, KDT's main weakness is limited scalability due to its one-dimensional distribution of sparse arrays.

The Small-world Network Analysis and Partitioning (SNAP) framework [BadMad2008] contains algorithms and kernels for exploring large-scale graphs. SNAP is a collection of different algorithms and building blocks that are optimized for small-world networks. It combines shared-memory thread level parallelism with state-of-the-art algorithm engineering for high performance. The graph data can be represented in a variety of different formats depending on the characteristics of the algorithm that operates on it. SNAP's performance and scalability are high for the reported algorithms, but a head-to-head performance comparison with PBGL and KDT is not available.

The Combinatorial BLAS (CBLAS) [Bul2010] is a C++ library for graph computation on large distributed memory machines. Like KDT, its primitives are based on sparse arrays and semiring operations, but the underlying sparse data structures, element types, and semiring operations are quite flexible and can be defined by the user. CBLAS supplies a 2D-partitioned distributed sparse array data structure, as well as implementations of some of the most useful semirings ((+,*); (min,+); etc.). CBLAS has been shown to scale well to over 1000 cores on high-performance distributed memory clusters. Its main disadvantages are: (1) since it uses a basic MPI infrastructure (for portability) it cannot take advantage of flexible shared-memory operations; (2) it supplies only algebraic primitives, not traversal-based primitives; (3) the user must cast elementary operations as semiring operators, which can sometimes be unintuitive.

Our proposed project combines the performance and power of the CBLAS's algebraic primitives and 2D sparse block data structures with the expressivity and generality of fully general labeled graphs and visitor/traversal primitives in a coherent framework.

4.2 . Frameworks for parallel sparse array and sparse matrix computation
We briefly mention some other work on parallel sparse arrays, much of which is directed at numerical sparse matrix computation rather than graph computation. Many libraries exist for solving sparse linear system and eigenvalue problems; some, like Trilinos [Her2005], include significant combinatorial capabilities. The Sparse BLAS [DufHerPoz2002] is a standard API for numerical matrix- and vector-level primitives; its focus is infrastructure for iterative linear system solvers, and therefore it does not include such primitives as SpGEMM and SpRef. Global Arrays [NieHarLit1996, NieCar1999] is a parallel dense and sparse array library that uses a one-sided communication infrastructure portable to message-passing, NUMA, and shared-memory machines. Star-P [GilReiSha2007, GilReiSha2008a] and pMatlab [KepTra2002] are parallel dialects of Matlab that run on distributed-memory message-passing machines; both include parallel sparse distributed array infrastructures.

4.3. Parallel graph algorithms
Finally, we briefly mention some more traditional algorithmic work. Parallel graph algorithms have been studied intensely by the theoretical computer science community since the 1970s; see e.g. Leighton [Lei1991] and Grama et al. [GGKK2003]. However, "computational graph theory," which combines the theoretical and practical considerations of parallel algorithms, very large data, computer architecture, performance analysis, and algorithm engineering, is still an emerging field.

It was recognized early on that efficient parallel graph algorithms could be quite different from sequential ones. Depth-first search, the basis for many efficient sequential graph algorithms, does not expose parallelism well. Shiloach and Vishkin [ShiVis1982] gave a PRAM algorithm for connected components that uses linking and shortcutting, edge operations, instead of traversal; later parallel algorithms and comparisons are described in [Gre1994, BHKK2007]. Though strongly connected components use DFS sequentially, no parallel algorithm is known that is efficient in both work and span; Fleischer et al. [FleHenPin2000] give a parallel algorithm that is effective in many practical settings. Maximal independent set is easy sequentially but more subtle in parallel; Luby's seminal paper [Lub1986] gave a simple randomized algorithm that is widely used (and also a more complicated derandomized version that is less widely used). Breadth-first search has more parallelism than DFS, though taking advantage of it can be subtle [YCHMHC2005, BadMad2006b, BHKK2007]. Priority-based traversal algorithms for problems like single-source weighted shortest paths have been parallelized via various schemes to relax strict priority constraints [MeySan2003, BHKK2007]. Subgraph isomorphism is an NP-hard problem of interest in various applications, and can sometimes be done effectively in practice on labeled graphs [BHKK2007]. Betweenness centrality has been used as a parallel benchmark and therefore studied in many settings [Bra2001, BMGSKMK2006, BadMad2006a, BKMM2007, KepGil2010, Bul2010].

5. Citations

[BKMM2007] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. In A. Bonato and F. R. K. Chung, editors, *WAW*, volume 4863 of *Lecture Notes in Computer Science*, pages 124–137. Springer, 2007.

[BadMad2006a] D. Bader and K. Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In *Proc. 35th Int'l. Conf. on Parallel Processing (ICPP 2006)*, pages 539–550. IEEE Computer Society, Aug. 2006.

[BadMad2006b] D. A. Bader and K. Madduri. Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2. In *Proc. 35th Int'l. Conf. on Parallel Processing (ICPP 2006)*, pages 523–530, IEEE, 2006.

[BadMad2008] D. A. Bader and K. Madduri. SNAP, Small-world Network Analysis and Partitioning: An open-source parallel graph framework for the exploration of large-scale networks. *Proc. IEEE Intl. Symp. Parallel & Distributed Processing (IPDPS)*, 2008.

[BMGSKMK2006] David A. Bader, Kamesh Madduri, John R. Gilbert, Viral Shah, Jeremy Kepner, Theresa Meuse, and Ashok Krishnamurthy. Designing scalable synthetic compact applications for benchmarking high productivity computing systems. *Cyberinfrastructure Technology Watch*, Nov. 2006.

[BBMW2009] B.W. Barrett, J.W. Berry, R. C. Murphy, and K. B.Wheeler. Implementing a portable multi-threaded graph library: The MTGL on qthreads. *Proc. 23rd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2009.

[BHKK2007] J. W. Berry, B. Hendrickson, S. Kahan, and P. Konecny. Software and algorithms for graph queries on multithreaded architectures. In *IPDPS*, pages 1–14. IEEE, 2007.

[Ble1989] G. E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers* 38(11): 1526 – 1538, 1989.

[Bra2001] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[Bul2010] A. Buluc. *Linear Algebraic Primitives for Parallel Computing on Large Graphs*. Ph.D. thesis, University of California, Santa Barbara, 2010.

[BFFGL2009] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-First ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, Calgary, Canada, August 2009.

[BulGil2008a] Aydin Buluç and John R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *37th International Conference on Parallel Processing (ICPP'08)*, pages 503-510, Portland, Oregon, USA, September 2008.

[BulGil2008b] Aydin Buluç and John R. Gilbert. On the representation and multiplication of hypersparse matrices. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008)*, pages 1-11, April 2008.

[BulGilBud2008] Aydin Buluç, John R. Gilbert, and Ceren Budak. Gaussian elimination based algorithms on the GPU. Technical Report UCSB/CS-2008-15, CS Department, University of California, Santa Barbara, November 2008. A revised version entitled “Solving path problems on the GPU” is to appear in *Parallel Computing*.

[Coh2009] J. Cohen. Graph twiddling in a mapreduce world. *IEEE Computing in Science and Engg*., 11(4):29–41, 2009.

[Dav2007] T. A. Davis. *Direct Methods for Sparse Linear Systems.* SIAM, 2006.

[DeaGhe2008] J. Dean and S. Ghemawat. Map Reduce: Simplified data processing on large clusters. *Comm. ACM* 51:1, 107-113, 2008.

[DufHerPoz2002] I. S. Duff, M. A. Heroux, and R. Pozo. An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum. *ACM Transactions on Mathematical Software* 28(2): 239-267, 2002.

[Fin2010] J. Fineman. Fundamental graph algorithms. In J. Kepner and J. Gilbert, editors, *Graph Algorithms in the Language of Linear Algebra*. SIAM, Philadelphia. In press, 2010.

[FleHenPin2000] Lisa K. Fleischer, Bruce Hendrickson, and Ali Pinar. On identifying strongly connected components in parallel. *Parallel and Distributed Processing*, Springer LNCS 1800:505-511, 2000.

[FriJoh1998] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. *Proc. Intl. Conf. Acoustics Speech and Signal Processing*, 1381-1384, IEEE, 1998.

[GilMolSch1992] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Analysis and Applications*, 13: 333-356, 1992.

[GilReiSha2007] John R. Gilbert, Steve Reinhardt, and Viral B. Shah. High performance graph algorithms from parallel sparse matrices. In *Applied Parallel Computing. State of the Art in Scientific Computing. 8th International Workshop, PARA 2006.*, pages 260-269, 2007.

[GilReiSha2008a] J. R. Gilbert, S. Reinhardt, and V. B. Shah. Distributed sparse matrices for very high level languages. *Advances in Computers*, 72, June 2008.

[GilReiSha2008b] John R. Gilbert, Steve Reinhardt, and Viral B. Shah. A unified framework for numerical and combinatorial computing. *Computing in Sciences and Engineering*, 10(2):20-25, Mar/Apr 2008.

[GGKK2003] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing,* second edition, Addison Wesley, 2003.

[GreLum2005] D. Gregor and A. Lumsdaine. The Parallel BGL: A generic library for distributed graph computations. In *Parallel Object-Oriented Scientific Computing (POOSC)*, July 2005.

[Gre1994] J. Greiner. A comparison of parallel algorithms for connected components. *Proc. Sixth ACM Symposium on Parallel Algorithms and Architectures*, pp. 16-25, 1994.

[Gus1978] F. G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software*, 4(3):250–269, 1978.

[HenBer2008] B. Hendrickson and J. Berry. Graph analysis with high-performance computing. *Computing in Science and Engineering* 10:2, pp. 14-19, IEEE 2008.

[Her2005] M. Heroux and 15 coauthors. An overview of the Trilinos project. *ACM Transactions on Mathematical Software* 31(3): 397-423, 2005.

[KepGil2010] J. Kepner and J. Gilbert, editors, *Graph Algorithms in the Language of Linear Algebra*. SIAM, Philadelphia. In press, 2010.

[KepTra2002] J. Kepner and N. Travinin. Parallel Matlab: The next generation. *Proc. HPEC Workshop*, MIT Lincoln Laboratory, 2002.

[KolBad2009] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. SIAM Rev. 51(3): 455-500, 2009.

[Lei1991] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1991.

[Lub1986] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Computing* 15(4):1036-1055, 1986.

[LGHB2007] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry. Challenges in parallel graph processing. *Parallel Processing Letters* vol. 17, no. 1, pp. 5-20, 2007.

[MeySan2003] U. Meyer and P. Sanders. Delta-stepping: A parallelizable shortest path algorithm. *J. Algorithms* 49(1): 114-152, 2003.

[NieCar1999] J. Nieplocha and B. Carpenter. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. *Parallel and Distributed Computing*, Springer LNCS 1586: 533-546, 1999.

[NieHarLit1996] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global Arrays: A nonuniform memory access programming model for high-performance computers. *J. Supercomputing*, 10:197-220, 1996.

[ShaGil2005] Viral Shah and John R. Gilbert. Sparse matrices in Matlab*P: Design and implementation. In *High Performance Computing (HiPC 2004)*, pages 144-155, 2005.

[ShiVis1982] Y. Shiloach and U. Vishkin. An O(log n) parallel connectivity algorithm. *J. Algs.*, 3(1):57–67, 1982.

[SieLeeLum2001] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library User Guide and Reference Manual*. Addison-Wesley Professional, 2001.

[Tar1972] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2): 146–160, 1972.

[VudDemYel2005] R. Vuduc, J. W. Demmel, and K. A. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics*: Conference Series, 16(1):521+, 2005.

[Whi2009] T. White. *Hadoop: the Definitive Guide*. O'Reilly Media, Inc., 2009.

[WOVSYD2009] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Parallel Computing*, 35(3):178–194, 2009.

[YCHMHC2005] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and U. Catalyurek. A scalable distributed parallel breadth-first search algorithm on Bluegene/L. In *SC ’05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, page 25, 2005.