Book of Abstracts of the Sixth SIAM Workshop on Combinatorial Scientific Computing

Bora Uçar

CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing

École Normale Supérieure de Lyon

July 21-23, 2014

Book of abstracts
Edited by: Bora Uçar, CNRS and LIP ENS Lyon, France

CSC14: the Sixth SIAM Workshop on Combinatorial Scientific Computing

The Sixth SIAM Workshop on Combinatorial Scientific Computing, CSC14, was organized at the Ecole Normale Supérieure de Lyon, France on 21st to 23rd July, 2014. This two and a half day event marked the sixth in a series that started ten years ago in San Francisco, USA.

The CSC14 Workshop's focus was on combinatorial mathematics and algorithms in high performance computing, broadly interpreted. The workshop featured three invited talks, 27 contributed talks and eight poster presentations. All three invited talks were focused on two interesting fields of research specifically: randomized algorithms for numerical linear algebra and network analysis. The contributed talks and the posters targeted modeling, analysis, bisection, clustering, and partitioning of graphs, applied in the context of networks, sparse matrix factorizations, iterative solvers, fast multi-pole methods, automatic differentiation, high-performance computing, and linear programming.

The workshop was held at the premises of the LIP laboratory of ENS Lyon and was generously supported by the LABEX MILYON (ANR-10-LABX-0070, Université de Lyon, within the program ``Investissements d'Avenir'' ANR-11-IDEX-0007 operated by the French National Research Agency), and by SIAM.

Program Committee

Organizing committee co-chairs

Paul Hovland, Argonne National Laboratory, USA
Bora Uçar, CNRS and ENS Lyon, France

Local organization

Evelyne Blesle, INRIA and ENS Lyon, France
Bora Uçar, CNRS and ENS Lyon, France

Organizing committee

Sanjukta Bhowmick, University of Nebraska Omaha, USA
Rob Bisseling, Utrecht University, the Netherlands
Erik Boman, Sandia National Laboratories, USA
Martin Bücker, University of Jena, Germany
Ümit V. Çatalyürek, The Ohio State University, USA
Assefaw Gebremedhin, Purdue University, USA
John Gilbert, University of California, Santa Barbara, USA
Laura Grigori, INRIA, France
Jean-Yves L'Excellent, INRIA, France
Sven Leyffer, Argonne National Laboratory, USA
X. Sherry Li, Lawrence Berkeley National Laboratory, USA
Lek-Heng Lim, University of Chicago, USA
Padma Raghavan, Pennsylvania State University, USA
Jennifer Scott, Rutherford Appleton Laboratory, UK
Blair Sullivan, North Carolina State University, USA
Miroslav Tůma, Academy of Sciences, Czech Republic

Steering Committee

Patrick R. Amestoy, INPT(ENSEEIHT)-IRIT, France
Rob Bisseling, Utrecht University, the Netherlands
Bruce Hendrickson, Sandia National Laboratories, USA
Paul Hovland, Argonne National Laboratory, USA
Alex Pothen, Purdue University, USA

Table of Contents

Invited Talks

Contributed Talks

Poster Presentations

ANALYTICAL AND ALGORITHMIC CHALLENGES IN NETWORK ANALYSIS

TAMARA G. KOLDA[†]

Large-scale graphs and networks are used to model interactions in a variety of contexts. We discuss the modeling choices in a graph representation of data, and the types of questions that are typically answered by studying networks. We then focus on measuring specific characteristics (e.g., triangles and 4-vertex patterns) of graphs and building generative models. In the undirected case, we discuss the Block Two-level Erdös-Rényi (BTER) model that reproduces a given degree distribution and clustering coefficient profile. We discuss various implementations of available methods and how they fit into the larger theme of graph processing. We conclude by discussing challenges such as how to characterize time-evolving networks.

[†]Sandia National Laboratories

RANDOMIZED ALGORITHMS IN NUMERICAL LINEAR ALGEBRA (RANDNLA) AND APPLICATIONS IN DATA ANALYSIS

PETROS DRINEAS[†]

The introduction of randomization in the design and analysis of algorithms for matrix computations (such as matrix multiplication, least-squares regression, the Singular Value Decomposition (SVD), etc.) over the last decade provided a new paradigm and a complementary perspective to traditional numerical linear algebra approaches. These novel approaches were motivated by technological developments in many areas of scientific research that permit the automatic generation of large data sets, which are often modeled as matrices.

In this talk we will outline how such approaches can be used to approximate problems ranging from matrix multiplication and the Singular Value Decomposition (SVD) of matrices to the Column Subset Selection Problem and the CX decomposition. Application of the proposed algorithms to data analysis tasks in population genetics will also be discussed.

[†]Rensselaer Polytechnic Institute

GRAPHS AND LINEAR ALGEBRA: THE NEW RANDOMIZED KACZMARZ LINEAR SOLVER

SIVAN TOLEDO[†]

Algorithms that solve symmetric diagonally-dominant linear systems in nearly linear time have been proposed in the theoretical-computer science literature for about a decade now. In this talk, I will describe a promising member of this class of algorithms, called the Randomized Combinatorial Kaczmarz, which was proposed recently by Kelner, Orecchia, Sidford, and Zhu [1]. This algorithm is relatively simple to understand and implement. I will describe the algorithm, relate it to older combinatorial solvers that were based on preconditioning, and I will discuss remaining challenges in this exciting area.

REFERENCES

[1] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu, *A simple, combinatorial algorithm for solving SDD systems in nearly-linear time*, CoRR,abs/1301.6628, 2013.

[†]Tel-Aviv University

Fast Implementation of the Minimum Local Fill Ordering Heuristic

Esmond G. Ng and Barry W. Peyton

Abstract

It is well known that sparse matrix factorizations suffer fill. That is, some of the zero entries in a sparse matrix will become nonzero during factorization. To reduce factorization time and storage, it is important to arrange the computation so that the amount of fill is kept small. It is also well known that the amount of fill is often influenced greatly by how the rows and columns of the sparse matrix are permuted (or ordered). We focus on the Cholesky factorization of a sparse symmetric positive definite matrix, in which case the fill depends solely on the sparsity of the given matrix and on the choice of the permutation (or ordering).

We use the following notation. Let A be an $n \times n$ symmetric positive definite matrix and P be an $n \times n$ permutation matrix. Let m be the number of nonzero entries in the lower triangle of A. We denote the factorization of PAP^T by LL^T, where L is lower triangular. We use L_{*k} to denote the k-th column of L. The number of nonzero entries in a vector v is represented by $\mathrm{nnz}(v)$.

It is well known that finding an ordering to minimize fill in sparse Cholesky factorization is an NP-complete problem [7]. Thus, we rely on heuristics. Two greedy, bottom-up heuristics are the minimum degree (MD) algorithm and the minimum local fill (MF) algorithm. The MD algorithm, introduced by Tinney and Walker [5], is probably the best-known and most widely-used greedy heuristic. It reduces fill by finding a permutation P so that $\mathrm{nnz}(L_{*k})$ is minimized locally at step k of the factorization. A great deal of work has been done to reduce the runtime of the MD algorithm. The MF algorithm, also introduced by Tinney and Walker [5], is not as well known or widely used as MD. In the MF algorithm, the permutation is chosen so that the number of zero entries in the reduced matrix that become nonzero is as small as possible at each step of the factorization.

There are a number of reasons why the MF algorithm has not been as popular as the MD algorithm. The metric $\mathrm{nnz}(L_{*k})$ is easy and inexpensive to compute. By contrast, the metric required by the MF algorithm is more difficult and expensive to compute. Consequently, the general experience has been that the MF algorithm requires far greater time than the MD algorithm. For example, Rothberg and Eisenstat [4] report that "while many of the enhancements described above for minimum degree are applicable to minimum local fill (particularly supernodes), runtimes are still prohibitive". Also, Ng and Raghavan [3] report that their implementation of MF was on average slower than MD by "two orders of magnitude".

Another reason for the lack of popularity of the MF algorithm is the belief that MF orderings are often just marginally better than MD orderings [1]. It has been shown, however, that MF orderings are often considerably better than MD orderings. For example, in an early version of [3], Ng and Raghavan reported that their MF orderings, on average, resulted in 9% less fill and 21% fewer operations than their MD orderings. On a different set of test matrices, Rothberg and Eisenstat [4] similarly reported that their MF orderings, on average, resulted in 16% less fill and 31% fewer operations than their MD orderings. Consequently, a truly efficient way to compute MF orderings would prove valuable, and that is the focus of our talk.

The reason for the high runtimes of standard implementations of MF is that whenever a column's fill count may have changed, it is set to zero and recomputed from scratch. In [6], Wing and Huang described an elegant way to *update* the deficiencies rather than recomputing them from scratch. Their updating scheme was mentioned a few times in the circuit simulation literature, but it apparently was not widely used and it certainly was not adopted by the sparse matrix community.

We will describe in this talk our recent work on the Wing-Huang updating scheme. In particular, we will show that the worst-case time complexity of the MF algorithm with Wing-Huang updates is the *same* as that of the MD algorithm, namely $O(n^2m)$. We will also demonstrate that techniques for reducing the runtime of the MD algorithm, such as mass elimination and indistinguishable nodes, are equally applicable in the efficient implementation of the MF algorithm with Wing-Huang updates. It is particularly important that we can adapt the Wing-Huang updating technique so that it can be used efficiently when quotient graphs are used to represent the elimination graphs.

Results from our preliminary implementation of the MF algorithm with Wing-Huang updates are encouraging. Over a collection of 48 sparse matrices from the Florida Sparse Matrix Collection, our MF algorithm with Wing-Huang updates is just 4.6 times more expensive than the minimum degree (MMD) algorithm with multiple eliminations [2] on average. Our MF orderings, on the average, produce 17% less fill and require 31% fewer operations than the MMD algorithm. On one large test matrix (3dtube), MF produces 29% less fill and requires 55% fewer operations.

In the future, we hope to look into ways to further reduce runtimes for our implementation of the MF algorithm using Wing-Huang updating.

References

[1] Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, NY, 2nd edition, 1989.

[2] Joseph W.H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11(2):141–153, 1985.

[3] Esmond G. Ng and Padma Raghavan. Performance of greedy ordering heuristics for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(4):902–914, 1999.

[4] Edward Rothberg and Stanley C. Eisenstat. Node selection strategies for bottom-up sparse matrix orderings. *SIAM J. Matrix Anal. Appl.*, 19(3):682–695, 1998.

[5] W.F. Tinney and J.W. Walker. Direct solution of sparse network equations by optimally ordered triangular factorization. In *Proc. IEEE*, volume 55, pages 1801–1809, 1967.

[6] O. Wing and J. Huang. SCAP - a sparse matrix circuit analysis program. In *Proceedings - IEEE International Symposium on Circuits and Systems*, pages 213–215, 1975.

[7] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2(1):77–79, 1981.

IMPROVING COARSENING FOR MULTILEVEL PARTITIONING OF COMPLEX NETWORKS

Roland Glantz, Henning Meyerhenke, Peter Sanders, and Christian Schulz

Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT)

Introduction

Complex networks such as web graphs or social networks have become a research focus [1]. Such networks have many low-degree nodes and few high-degree nodes. They also have a small diameter, so that the whole network is discovered within a few hops. Various emerging applications produce massive complex networks whose analysis would benefit greatly from parallel processing. Parallel graph algorithms, in turn, often require a suitable network partition, motivating graph partitioning (GP).

Given a graph $G = (V, E)$ with (optional) edge weight function ω and a number of blocks $k > 0$, the GP problem asks for a partition of V into blocks $V_1, \ldots, V_k$ such that no block is larger than $(1 + \varepsilon) \cdot \lceil \frac{|V|}{k} \rceil$, where $\varepsilon \geq 0$ is the allowed imbalance. When GP is used for parallel processing, each processing element (PE) usually receives one block, and edges running between two blocks model communication between PEs. The most widely used objective function is the *edge cut*, the total weight of the edges between different blocks. To model the communication cost of parallel iterative graph algorithms, the *maximum communication volume* (MCV) can be more accurate [4]. MCV considers the worst communication volume taken over all blocks V_p $(1 \leq p \leq k)$ and thus penalizes imbalanced communication: $MCV(V_1, \ldots, V_k) := \max_p \sum_{v \in V_p} |\{V_i \mid \exists \{u, v\} \in E \text{ with } u \in V_i \neq V_p\}|$.

For solving optimization tasks such as GP on large networks, multilevel methods (consisting of recursive coarsening, initial partitioning, successive prolongation and local improvement) are preferred in practice. Partitioning static meshes this way is fairly mature. Yet, the structure of complex networks challenges current tools. One key issue for most multilevel graph partitioners is coarsening.

Here we present two independent improvements to coarsening. The first one uses the established framework of contracting edges computed as matching. Yet, it defines a new *edge rating* which indicates with non-local information how much sense it makes to contract an edge and thus guides the matching algorithm. The second approach uses cluster-based coarsening and contracts larger sets of nodes into a supernode, yielding fewer levels.

New Coarsening Approaches

Conductance-based Edge Rating. Let the terms *cut* and *cut-set* refer to a 2-partition $(C, \overline{C})$ of a graph and to the set $S(C)$ of edges running between C and $\overline{C}$, respectively. The graph clustering measure *conductance* [5] relates the size (or weight) of the cut-set to the volumes of C and $\overline{C}$. More precisely, $\text{cond}(G) := \min_{C \subset V} \frac{|S(C)|}{\min\{\text{vol}(C), \text{vol}(\overline{C})\}}$, where the volume $\text{vol}(X)$ of a set X sums over the (weighted) degrees of the nodes in X.

An edge rating in a multilevel graph partitioner should yield a low rating for an edge e if e is likely to be contained in the cut-set of a "good" cut. In our approach a good cut is one that has low conductance and is thus at least moderately balanced. A loose connection between conductance and MCV can be established via isoperimetric graph partitioning [3]. Our approach to coarsen a graph with a new edge rating is as follows. (i) Generate a collection $\mathcal{C}$ of moderately balanced cuts of G with a low conductance value. (ii) Define a measure $\text{Cond}(\cdot)$ such that $\text{Cond}(e)$ is low [high] if e is [not] contained in the cut-set of a cut in $\mathcal{C}$ with low conductance. (iii) Use the new edge rating $\text{ex\_cond}(\{u, v\}) = \omega(\{u, v\})\,\text{Cond}(\{u, v\})/(c(u)c(v))$ as weights for an approximate maximum weight matching algorithm $\mathcal{A}$, where $c(x)$ refers to the weight of node x. The higher $\text{ex\_cond}(e)$, the higher the chances for e to be contracted. (iv) Run $\mathcal{A}$ and contract the edges returned in the matching.

We arrive at a collection $\mathcal{C}$ of $|V|-1$ moderately balanced cuts of G by (i) computing connectivity-based "contrast" values for the edges of G, (ii) computing a minimum spanning tree T^m of G w. r. t. these values, and (iii) letting $\mathcal{C}$ consist of G's fundamental cuts w. r. t. T^m. We want the contrast value $\gamma(e)$ of an edge e to be high if e is part of "many" connections via shortest paths in G. Based on a collection $\mathcal{T}$ of rooted spanning trees of G, this means that (i) e is contained in many trees from $\mathcal{T}$ and (ii) e is not involved in small cuts that separate a small subgraph of G from G's "main body". We achieve this by setting $\gamma(\{u, v\}) = \min\{n_{\mathcal{T}}(u, v), n_{\mathcal{T}}(v, u)\}$, where $n_{\mathcal{T}}(u, v)$ denotes the number of trees in $\mathcal{T}$ containing e such that u is closer to the tree's root than v. $\text{Cond}(\cdot)$ is finally

defined such that $\text{Cond}(e)$ is low [high] if e is [not] contained in the cut-set of a cut in $\mathcal{C}$ with low conductance: $\text{Cond}(e) = \min_{C \in \mathcal{C}, e \in S(C)}(\text{cond}(C))$.

Cluster-based Coarsening. As an alternative approach to coarsening networks with a highly irregular structure, we propose a more aggressive coarsening algorithm that contracts size-constrained clusterings computed by a label propagation algorithm (LPA). LPA was originally proposed by Raghavan *et al.* [7] for graph clustering. It is a fast, near-linear time algorithm that locally optimizes the number of edges cut. Initially, each node is in its own cluster/block. In each of the subsequent rounds, the nodes of the graph are traversed in a random order. When a node v is visited, it is moved to the block that has the strongest connection to v (with some tie-breaking mechanism). The original process is repeated until convergence, each round takes $\mathcal{O}(n + m)$ time. Here, we perform at most ℓ iterations of the algorithm, where ℓ is a tuning parameter, and stop the algorithm if less then 5% of the nodes changed its cluster during one round. Hence, we do not face the occasional instabilities of the original algorithm. Most importantly, we adapt LPA such that clusters cannot grow beyond a certain size. This is done to respect the imbalance criterion of GP.

We integrate further algorithmic extensions such as modified iterations over the node set within LPA, ensemble clusterings, and iterated multilevel schemes. They are described in more detail in the corresponding full paper.

To compute a graph hierarchy, the clustering is contracted by replacing each cluster with a single node, and the process is repeated recursively until the graph is small.

Here we aim at partitioning for low edge cuts with this method. The intuition for achieving this goal is that a good clustering contains only few edges between clusters.

Implementation and Experimental Results

Experimental results have been obtained by implementing our new methods within the framework of the state-of-the-art graph partitioner KaHIP [9].

Conductance-based Edge Rating. KaHIP contains a reference implementation of the edge rating ex_alg(·), which yielded the best quality for complex networks so far [8]. In addition to our new edge rating ex_cond(·), we have integrated a greedy postprocessing step that trades in small edge cuts for small MCVs into KaHIP. Our experiments show that greedy MCV postprocessing alone improves the partitions of our complex network benchmark set in terms of MCV by about 11% with a comparable running time for both ex_alg(·) and ex_cond(·). Additional bipartitioning experiments (MCV postprocessing included) show that, compared to ex_alg(·), the fastest variant of our new edge rating further improves the MCVs by 10.3%, at the expense of an increase in running time by a factor of 1.8. Altogether, compared to previous work on partitioning complex networks with state-of-the-art methods [8], the total reduction of MCV amounts to 20.4%.

Cluster-based Coarsening. For the second set of experiments, KaHIP uses the hierarchy computed by cluster-based coarsening and its own initial partitioning as well as existing local search algorithms for refinement on each level, respectively. Some algorithm configurations also use the size-constrained LPA as local search procedure. We compare against the established tools kMetis, hMetis, and Scotch, all in graph partitioning mode.

Depending on the algorithm's configuration, we are able to compute the best solutions in terms of edge cut or partitions that are comparable to the best competitor in terms of quality, hMetis, while being nearly an order of magnitude faster on average. The fastest configuration partitions a web graph with 3.3 billion edges using a single machine in about ten minutes while cutting less than half of the edges than the fastest competitor, kMetis.

Accompanying Publications. Details can be found in the respective papers [2, 6] and their full arXiv versions.

References

[1] L. F. Costa, O. N. Oliveira Jr, G. Travieso, F. A. Rodrigues, P. R. V. Boas, L. Antiqueira, M. P. Viana, and L. E. C. Rocha. Analyzing and Modeling Real-World Phenomena with Complex Networks: A Survey of Applications. *Adv. in Physics*, 60(3):329–412, 2011.

[2] R. Glantz, H. Meyerhenke, and C. Schulz. Tree-based coarsening and partitioning of complex networks. In *Proc. 13th Intl. Symp. on Experimental Algorithms (SEA 2014)*, pages 364–375, 2014.

[3] L. Grady and E. L. Schwartz. Isoperimetric graph partitioning for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(3):469–475, 2006.

[4] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000.

[5] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, 2004.

[6] H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning complex networks via size-constrained clustering. In *Proc. 13th Intl. Symp. on Experimental Algorithms (SEA 2014)*, pages 351–363, 2014.

[7] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[8] I. Safro, P. Sanders, and C. Schulz. Advanced coarsening schemes for graph partitioning. In *Proc. 11th Int. Symp. on Experimental Algorithms*, pages 369–380. Springer, 2012.

[9] P. Sanders and C. Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, LNCS, pages 164–175. Springer, 2013.

Partitioning hypergraphs for multiple communication metrics

Mehmet Deveci [1], Kamer Kaya [1], Bora Uçar [2], Ümit V. Çatalyürek [1]
[1] Dept. Biomedical Informatics, The Ohio State University
[2] CNRS and LIP, ENS Lyon, France

The datasets in many fields of science and engineering are growing rapidly with the recent advances that enable generating and storing big data. These datasets usually have irregular patterns, and there exist complex interactions between their elements. These interactions make it difficult to develop simple and efficient load-balancing mechanisms for the data analysis kernels, forcing researchers to study more complex partitioning methods. Thus, a good partitioning of the data, which is necessary to obtain a scalable performance, a shorter makespan, a minimized energy usage, and a better resource utilization, is more important and harder to obtain.

In scientific computing, computational tasks are usually modeled using connectivity-based topological models, such as graphs and hypergraphs [1, 2, 3]. These models transform the problem at hand to a balanced vertex partitioning problem. The balance restriction on part weights in conventional partitioning corresponds to the load balance in a parallel environment, and the minimization objective for a given metric relates to the minimization of the communication cost among the processing units. A good partitioning should minimize the inter-processor communication while distributing the computation load evenly to the processors. In this work, we study these connectivity-based models and methods, specifically hypergraph models and methods, designed to partition the communicating tasks for an efficient parallelization.

Most of the existing state-of-the-art hypergraph partitioners aim to minimize the total communication volume while balancing the load for efficient parallelization [1, 4, 5]. However, the other communication metrics, such as the total number of messages, the maximum amount of data transferred by a single processor, or combinations of multiple metrics, are equally, if not more, important. For example, the latency-based metrics, which model the communication by using the number of messages sent/received throughout the execution, become more and more important as the number of processors increases [6]. Ideal partitions yield perfect computational load balance and minimize the communication requirements by minimizing the communication overhead. On the other hand, most of the existing hypergraph partitioning methods aim to minimize only the traditional total communication volume metric, with the hope that it improves the other communication metrics as a side effect.

In this work, we argue that the general hypergraph model used by the state-of-art hypergraph partitioners is not sufficient to model and capture other communication metrics than the total volume of communication. We propose a directed hypergraph model that can simultaneously capture multiple communication metrics. Given an application, our main objective is to partition the tasks evenly among processing units and to minimize the communication overhead by minimizing several communication cost metrics by using the proposed directed hypergraph model. Previous studies addressing multiple metrics [7, 8] with the traditional hypergraph model work in two phases where the phases are concerned with disjoint subsets of communication metrics. Generally, the first phase tries to obtain a proper partitioning of data for which the total communication volume is reduced. Starting from this partitioning, the second phase tries to optimize another communication met-

ric. Even though such two-phase approaches allow the use of state-of-the-art techniques in one or both phases, since the solutions sought in one phase are oblivious of the metric used in the other, the search can stuck in a local optima that cannot be improved in the other phase. Instead, we present a novel approach to treat the minimization of multiple communication metrics as a multi-objective minimization problem which is solved in a single phase with the help of the proposed directed hypergraph model. Addressing all the metrics in a single-phase allows a trade-off between the cost associated with one metric and the cost associated with another one. Inherently, the standard hypergraph model cannot see the communication metrics that are defined per-processor basis. Therefore, the balance on the communication loads of the processors cannot be modeled and formulated in a natural way. Furthermore, since almost all the state-of-the-art partitioners use iterative-improvement-based heuristics for the refinement, a single-phase approach increases the explored search space by avoiding local optima for a single metric.

We have materialized our approach in UMPa [9], which is a multi-level partitioner employing a directed hypergraph model and novel K-way refinement heuristics, since balancing per-processor communication cost metrics requires a global view of the partition. We present methods for minimizing the maximum communication volume, the total and maximum number of messages per processor, as well as the traditional total communication volume in a generalized framework that simultaneously minimizes multiple prioritized communication overhead metrics at the same time. Compared to the state-of-art hypergraph partitioners, we show on a large number of problem instances that UMPa produces much better partitions in terms of several communication metrics with 128, 256, 512, and 1024 processing units; UMPa reduces the maximum communication volume, the total number of messages, and the maximum number of messages sent by a single part up to %85, %45, and %43, respectively.

References

[1] Ü. V. Çatalyürek, C. Aykanat, Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 673–693.

[2] B. Hendrickson, T. G. Kolda, Graph partitioning models for parallel computing, Parallel Computing 26 (2000) 1519–1534.

[3] Ü. V. Çatalyürek, C. Aykanat, A fine-grain hypergraph model for 2D decomposition of sparse matrices, in: Proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA, 2001.

[4] B. Vastenhouw, R. H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix-vector multiplication, SIAM Review 47 (1) (2005) 67–95.

[5] K. Devine, E. Boman, R. Heaphy, R. Bisseling, Ü. V. Çatalyürek, Parallel hypergraph partitioning for scientific computing, in: Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2006.

[6] K. Kaya, B. Uçar, U. V. Çatalyürek, On analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication, in: Proc. of the 10th Int'l Conf. on Parallel Processing and Applied Mathematics (PPAM), 2013.

[7] B. Uçar, C. Aykanat, Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies, SIAM J. Sci. Comput. 25 (2004) 1837–1859.

[8] R. H. Bisseling, W. Meesen, Communication balancing in parallel sparse matrix-vector multiplication, Electronic Transactions on Numerical Analysis 21 (2005) 47–65.

[9] Ü. V. Çatalyürek, M. Deveci, K. Kaya, B. Uçar, UMPa: A multi-objective, multi-level partitioner for communication minimization, in: 10th DIMACS Implementation Challenge Workshop: Graph Partitioning and Graph Clustering, 2012, pp. 53–66, published in Contemporary Mathematics, Vol. 588, 2013.

Comparison of One-Phase and Two-Phase Approaches for Replicated Hypergraph Partitioning

R. Oguz Selvitopi[a], Volkan Yazici[c], Ata Turk[b], Cevdet Aykanat[a]

[a] *Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey*
[b] *Yahoo! Labs, Barcelona, Spain*
[c] *Department of Computer Science, Ozyegin University, Istanbul 34794, Turkey*

Hypergraphs find their applications in wide range of domains that include VLSI circuit design, scientific computing [1], information retrieval and database systems. They are successfully adopted and used in these domains to model problems with different types of relations. These relations can broadly be categorized as directed or undirected according to the requirements of the application being modeled. In undirected relations, the relation among data items is equally shared, whereas in directed relations, there exists an input/output relation among data items being modeled. In information retrieval and database systems, replication is a useful method to provide fault tolerance, enhance parallelization and improve processing performance. In this study, we target vertex replication to further improve the objective of hypergraph partitioning for hypergraphs that employ undirected relations. The applications that benefit from hypergraph models and replication can utilize the methods and techniques that are proposed for replicating vertices of undirected hypergraphs.

There are two possible approaches to solve the replicated hypergraph partitioning problem: one-phase and two-phase. In [2], we propose a one-phase approach where replication of vertices in the hypergraph is performed during the partitioning process. This is achieved during the uncoarsening phase of the multilevel methodology by proposing a novel iterative-improvement-based heuristic which extends the Fiduccia-Mattheyses (FM) algorithm [3] by also allowing vertex replication and unreplication. In [4], we describe a two-phase approach in which replication of vertices is performed after partitioning is completed. The replication phase of this two-phase approach utilizes a unified approach of coarsening and integer linear programming (ILP) schemes. The one-phase approach has the possibility of generating high quality solutions since it achieves replication during the partitioning process and it considers unreplication of vertices as well. The advantages of using two-phase approach are the flexibility of using any of the hypergraph partitioning tools available and being able to work on the partitions that already contain replicated vertices.

In the one-phase approach, to perform replication of vertices during the partitioning process, the FM heuristic is extended to allow replication and unreplication of vertices as well as move operations. This extended heuristic has the same linear complexity as the original FM heuristic and is able to replicate vertices in a two-way partition by new vertex states and gain definitions. This heuristic is later utilized in a recursive bipartitioning framework to enable K-way replicated partitioning of the hypergraph and it supports the two widely used cut-net and connectivity cutsize metrics. It is integrated into the successful hypergraph partitioning tool PaToH as the refinement algorithm in the uncoarsening phase.

The two-phase approach utilizes Dulmage-Mendelsohn [5] decomposition to find replication

sets for each part by only considering boundary vertices. The replication sets are bounded by a maximum replication capacity, and they are arranged in such a way that the imbalance of the given original partition is not disturbed, or even is sometimes improved. The balancing constraint is enforced by proposing a part-oriented method to determine the amount of replication to be performed on each part in a particular order.

We compare the mentioned one-phase and two-phase approaches for achieving vertex replication in undirected hypergraphs. We present the results of the quality of the partitions on hypergraphs from different domains with varying setups.

References

[1] O. Selvitopi, M. Ozdal, C. Aykanat, A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies, Parallel and Distributed Systems, IEEE Transactions on PP (99) (2014) 1–1. doi:http://dx.doi.org/10.1109/TPDS.2014.2311804.
URL http://www.computer.org/csdl/trans/td/preprint/06766662-abs.html

[2] R. Oguz Selvitopi, A. Turk, C. Aykanat, Replicated partitioning for undirected hypergraphs, J. Parallel Distrib. Comput. 72 (4) (2012) 547–563. doi:10.1016/j.jpdc.2012.01.004.
URL http://dx.doi.org/10.1016/j.jpdc.2012.01.004

[3] C. M. Fiduccia, R. M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the 19th Design Automation Conference, DAC '82, IEEE Press, Piscataway, NJ, USA, 1982, pp. 175–181.
URL http://portal.acm.org/citation.cfm?id=800263.809204

[4] V. Yazici, C. Aykanat, Constrained min-cut replication for k-way hypergraph partitioning, INFORMS Journal on Computing, 10.1287/ijoc.2013.0567 0 (0) (0) null. arXiv:http://pubsonline.informs.org/doi/pdf/10.1287/ijoc.2013.0567, doi:10.1287/ijoc.2013.0567.
URL http://pubsonline.informs.org/doi/abs/10.1287/ijoc.2013.0567

[5] A. Pothen, C.-J. Fan, Computing the block triangular form of a sparse matrix, ACM Trans. Math. Softw. 16 (4) (1990) 303–324. doi:10.1145/98267.98287.
URL http://doi.acm.org/10.1145/98267.98287

An Efficient Automatic Differentiation Algorithm for Hessians: Working With *Live* Variables*

Mu Wang Assefaw Gebremedhin Alex Pothen

Introduction. Gower and Mello [3] recently introduced a graph model for computing Hessians using Automatic Differentiation (AD) [1, 2]. In the model, which is based solely on Reverse Mode AD, the computational graph of the input function is augmented with minimal information—additional edges corresponding precisely with nonlinear interactions—and the symmetry available in Hessian computation is exploited. Using the model, they developed an algorithm, called `Edge_Pushing`, where edges representing nonlinear interactions are 'created' and their contributions to descendants are 'pushed' as the algorithm proceeds. The approach represents an important progress in AD for Hessians, but it unfortunately has several shortcomings. First, the authors' derivation of the algorithm is rather complicated and hard to understand. Second, their implementation in `ADOL-C` relies on an indexing assumption not necessarily supported by `ADOL-C` so that their code gives incorrect results in some cases. In this work, we provide a new, intuitive derivation of the `Edge_Pushing` algorithm from a completely different perspective and a robust implementation (built on top of `ADOL-C`) that works correctly in all cases. At the heart of our approach lies this: we identify an invariant in the first order *incremental reverse mode* of AD, which we arrive at by taking a *data-flow* perspective. We obtain the Hessian algorithm by extending the invariant to second order. Additionally, we incorporate *preaccumulation* in the Hessian algorithm to further enhance performance.

Reverse Mode AD. In data-flow analysis in compiler theory, a variable is said to be *live* if it holds a value that *might be* read in the future. We find a similar notion useful in our context. Since in reverse mode AD all information about the execution sequence of the objective function is recorded on an evaluation trace, we can in fact work with a more restricted definition for a live variable. In particular, we say a variable is live if it holds a value that *will be* read in the future. And we call the set made up of all live variables at each step of the execution sequence a *live variable set* in that sequence.

Following the notations of Griewank and Walther [1], the first order incremental reverse mode of AD can be written using a sequence of Single Assignment Code (SAC) as:

Algorithm: First Order Incremental Reverse Mode (FOIRM)	
Initialization: $\bar{v}_l = 1.0$, $\bar{v}_{1-n} = \cdots = \bar{v}_0 = \bar{v}_1 = \cdots = \bar{v}_{l-1} = 0$	
for $i = l, \cdots, 1$ **do**	
for all $v_j \prec v_i$ **do**	($\prec$ denotes precedence)
$\bar{v}_j += \frac{\partial \varphi_i}{\partial v_j} \bar{v}_i$	

We observe the following invariant in this mode of computing adjoints:

Observation 1 *The set of adjoints computed in each step i of the FOIRM algorithm involve partial derivatives with respect to only the current live variable set, not the entire set of variables.*

Hessian Algorithm. We extend the invariant formulated in Observation 1 to second order derivatives. Let S denote the set of live variables in a given step. Then, the first order derivatives (adjoints), denoted in the code in FOIRM by $\bar{v}$, for each $v \in S$, can be viewed as a mapping $a : S \to \mathbb{R}$, $a(v) = \bar{v}$. Analogously, the second order derivatives (Hessian) can be viewed as a symmetric mapping $h : S \times S \to \mathbb{R}$, $h(v, u) = h(u, v)$. Our target algorithm is then precisely a prescription of how S, $a(S)$ and $h(S, S)$ should be changed as the associated SAC is processed such that the invariant is maintained.

Let $\hat{S}$, $\hat{a}(\hat{S})$ and $\hat{h}(\hat{S}, \hat{S})$ denote the live variable set, the adjoint mapping, and the Hessian mapping, respectively, after a SAC $v_i = \varphi_i(v_j)_{v_j \prec v_i}$ is processed. Because the sequence proceeds in the reverse order,

*A full report on this work is being submitted elsewhere. Authors' emails: {wang970, agebreme, apothen}@purdue.edu

we have $\hat{S} = \{S \setminus \{v_i\}\} \cup \{v_j | v_j \prec v_i\}$. Considering the adjoints equation, and noting that $\frac{\partial \varphi_i}{\partial v_j} = 0$ when $v_j \not\prec v_i$, and $a(v_j) = 0$ when $v_j \notin S$:

$$\forall v_j \in \hat{S}, \hat{a}(v_j) = a(v_j) + \frac{\partial \varphi_i}{\partial v_j} a(v_i).$$

For the second order rule, noting that $h(v_j, v_k) = 0$ when $v_j \notin S$ or $v_k \notin S$, and applying the chain rule of calculus, analogous to the adjoint case, we have $\forall v_j, v_k \in \hat{S}$:

$$\begin{aligned}\hat{h}(v_j, v_k) = h(v_j, v_k) + \frac{\partial \varphi_i}{\partial v_j} h(v_i, v_k) + \frac{\partial \varphi_i}{\partial v_k} h(v_i, v_j) + \frac{\partial \varphi_i}{\partial v_j} \frac{\partial \varphi_i}{\partial v_k} h(v_i, v_i) \\ + a(v_i) \frac{\partial^2 \varphi_i}{\partial v_j \partial v_k}. \end{aligned} \quad (1)$$

Equation (1) corresponds to the `Edge_Pushing` algorithm of [3], in which the last three terms on the first line represent the pushing part, and the sole term in the second line represents the creating part in the component-wise form of their algorithm.

Implementation and Evaluation. We implemented this data flow-based Hessian algorithm in `ADOL-C`. We observe that in order to take advantage of the symmetry available in Hessian computation, the result variables in the SAC sequence need to have monotonic indices. However, the location scheme for variables currently used in `ADOL-C` does not satisfy this property, which is one reason why the Gower-Mello implementation of the `Edge_Pushing` algorithm fails. We implemented a fix in `ADOL-C` where we appropriately translate indices of variables before starting the reverse Hessian algorithm.

To further improve efficiency, we incorporate a statement-level *preaccumulation* technique to the Hessian algorithm. Preaccumulation splits the reverse Hessian algorithm into a local and a global level. In the local level, each SAC is processed to compute the first and second order derivatives of local functions defined by assign-statements in the execution path. In the global level, the derivatives of each local function is accumulated to compute the entire Hessian of the objective function.

The table below shows sample results comparing the runtime (sec.) of the new approach (`EPwithPreacc` and `EPwithoutPreacc`) with two related approaches: (i) a full Hessian algorithm in which sparsity is *not* exploited (`Full-Hessian`) and (ii) two compression-based sparse Hessian algorithms involving sparsity structure detection, graph coloring, compressed evaluation and recovery (`SparseHess-direct` and `SparseHess-indirect`). Results are shown for synthetic test functions from [5] and mesh optimization problems in the FeasNewt benchmark [4]. Details will be discussed in the upcoming full report.

	Synthetic				**Mesh Optimization**		
Matrix order n:	10,000	10,000	10,000	10,000	2,598	11,597	39,579
Number of nonzeros:	19,999	59,985	44,997	59,985	46,488	253,029	828,129
`Full-Hessian`	31.78	573.16	28.91	33.83	129.36	> 2 hours	> 2 hours
`SparseHess-direct`†	0.04	0.30	0.12	16.05	5.17	37.35	129.35
`SparseHess-indirect`†	0.20	0.31	0.33	25.72	4.14	28.94	111.97
`EPwithoutPreacc`	0.05	0.27	0.12	0.12	0.53	3.63	12.80
`EPwithPreacc`	0.06	0.23	0.08	0.10	0.48	3.27	11.10

† The times are a total of the four steps, whose contributions vary greatly. As an example, the breakdown for the largest mesh optimization problem (nnz=828,129) is:

	Pattern	**Coloring**	**Compressed H.**	**Recovery**
`SparseHess-direct`	54.1%	1.02%	44.8%	0.03%
`SparseHess-indirec`	61.1%	0.96%	24.8%	13.1%

References

[1] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* 2nd Edition, SIAM, 2008.

[2] U. Naumann. *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation.* SIAM, 2012.

[3] R. M. Gower and M. P. Mello. *A New Framework for The Computation of Hessians.* Optimization Methods and Software, Volume 27, Issue 2, pp 251–273, 2012.

[4] T. S. Munson and P. D. Hovland. *The FeasNewt Benchmark.* IEEE International Symposium on Workload Characterization (IISWC), 2005.

[5] L. Luksan, C. Matonoha and J. Vlcek: *Sparse Test Problems for Unconstrained Optimization.* Tech. Rep. V-1064, ICS AS CR, January 2010.

Hierarchical seeding for efficient sparsity pattern recovery in automatic differentiation

Joris Gillis*[1] and Moritz Diehl[2]

[1] KU Leuven, Electrical Engineering Department (ESAT-STADIUS), Kasteelpark Arenberg 10, 3001 Leuven, Belgium
[2]Freiburg Univ, Department of Microsystems Engineering (IMTEK), G.-Koehler-Allee 102, 79110 Freiburg, Germany

Obtaining the Jacobian $J = \frac{\partial f}{\partial x}$ of a vector valued function $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ is central to applications such as gradient-based constrained optimization and sensitivity analysis. When the function is not a black box, but instead an interpretable algorithm with run-time T, automatic differentiation offers a mechanistic way to derive two algorithms from the original $f(x)$ that accurately evaluate the following sensitivities:

Forward sensitivity	$\mathrm{AD}_f^{\mathrm{fwd}}(x, s^{\mathrm{fwd}}) = J(x)s^{\mathrm{fwd}}, \quad s^{\mathrm{fwd}} \in \mathbb{R}^n$
Adjoint/reverse sensitivity	$\mathrm{AD}_f^{\mathrm{adj}}(x, s^{\mathrm{adj}}) = J^T(x)s^{\mathrm{adj}}, \quad s^{\mathrm{adj}} \in \mathbb{R}^m,$

with the run-time of either of the algorithms AD_f a small multiple of T.
A straightforward approach to recover all Jacobian entries is to seed with columns of an identity matrix. In this way, the forward and reverse sensitivities correspond directly to respectively columns and rows of the sought-after Jacobian. For $m \ll n$, the obvious choice is to use m adjoint sensitivities, while in the $n \ll m$ case, using n is cheapest. With this strategy, the cost for a total Jacobian is in the order of $\min(n, m)T$.

If one knows the sparsity of J beforehand, the number of required sensitivities can potentially be drastically reduced. For example, when $n = m$ and J is known to be diagonal, a single sensitivity evaluation with seed $[1, 1, \ldots]^T$ suffices. More generally, a coloring of the *column intersection graph* of the sparsity pattern of J provides a small set of seeds usable to obtain the full Jacobian. We denote such coloring as $col(J)$ and use an existing distance-2 unidirectional algorithm[2].
The potentially dramatic speed-up requires first the sparsity pattern to be obtained. We will assume for the remainder of this work that we can derive the following bitvector-valued dependency functions[3] from the original algorithm f:

Forward dependency	$\mathrm{dep}_f^{\mathrm{fwd}}(d^{\mathrm{fwd}}) \in \mathbb{B}^m, \quad d^{\mathrm{fwd}} \in \mathbb{B}^n$
Adjoint/reverse dependency	$\mathrm{dep}_f^{\mathrm{adj}}(d^{\mathrm{adj}}) \in \mathbb{B}^n, \quad d^{\mathrm{adj}} \in \mathbb{B}^m,$

with $\mathbb{B}$ the Boolean set $\{0, 1\}$. A zero in the dependency function output means that any seed s with sparsity as in the input d, when supplied to the corresponding sensitivity function, would result in a zero sensitivity output in that same location.
A straightforward technique to recover the full sparsity pattern is to seed the dependency functions with slices of a unit matrix. The run-time τ of the dependency functions is typically orders of magnitude smaller than T. However, for large sparse matrices, the sparsity calculation run-time $\tau\min(n, m)$ could dominate the calculation of the Jacobian. In this work, we propose a hierarchical bitvector-based technique to recover the sparsity pattern faster for highly sparse cases, as would be the case in e.g. multiple-shooting based optimal control problem transcriptions.

*joris.gillis@esat.kuleuven.be; Joris Gillis is a Doctoral Fellow of the Fund for Scientific Research – Flanders (F.W.O.) in Belgium.

The coloring of a sparse Jacobian allows to recover more information from a single sensitivity sweep. A crucial observation is that it can do exactly the same for dependency sweeps. The proposed algorithm starts with obtaining the sparsity pattern in a coarse resolution, performing a coloring of this coarse resolution, and hence potentially reducing the number of fine-grained dependency sweeps needed to obtain a fine-grained image of the sparsity. The algorithm performs this refinement in a recursive way until the full sparsity is recovered:

Input : $\sigma \in \mathbb{N}, \sigma > 1$ subdivision factor
Input : Dimensions n and m of Jacobian
Init : $(N, M) \leftarrow (n, m)$; $r \leftarrow [1]$; `/* Initialize with a scalar */`
while $N > 1$ *and* $M > 1$ **do**
  fwd $\leftarrow$ col(r); adj $\leftarrow$ col(r^T) ; `/* Coloring of the coarse pattern */`
  if *adj is cheaper* **then** seed $\leftarrow$ adj ; $(N, M, n, m, r) \leftarrow (M, N, m, n, r^T)$; mode $\leftarrow$ 'adj';
  else seed $\leftarrow$ fwd; mode $\leftarrow$ 'fwd';
  $(\nu, \mu) \leftarrow$ dimensions of r; $(N, M) \leftarrow (\lceil N/\sigma \rceil, \lceil M/\sigma \rceil)$;
  $S \leftarrow$ block matrix with ν-by-μ empty cells of shape $n/(N\nu)$-by-$m/(M\mu)$;
  foreach $s \in$ *seed* **do**
    $d \leftarrow \text{block\_dep}\left(\text{mode}; s \otimes \mathbf{1}^{m/(M\mu)} \otimes v^M\right)$; `/* Block sparsity seeding */`
    $d \leftarrow \max\left((\mathbf{1}^{n/N} \otimes h^N)d, 1\right)$; `/* Block sparsity aggregate */`
    foreach j *in nonzero locations of* s **do**
      foreach i *in nonzero locations of column* j *of* r **do**
        $S_{i,j} \leftarrow$ rows $ni/(N\nu)$ to $n(i+1)/(N\nu)$ of d; `/* Store result */`
      end
    end
  end
  if *mode* = *'adj'* **then** $S \leftarrow S^T$; $(N, M, n, m) \leftarrow (M, N, m, n)$;
  $r \leftarrow S$;
end
Output: Jacobian sparsity r,

with $\otimes$ the Kronecker product, $\mathbf{1}^n$ a unit matrix of dimension n, v^n a column vector of dimension n with all entries 1, and h^n its transpose. block_dep splits up its bitmatrix argument into columns, feeds these to $\text{dep}_f^{\text{fwd}}$ or $\text{dep}_f^{\text{adj}}$ depending on the mode, and lumps the results back together to form a new bitmatrix. For ease of presentation, the above algorithm is restricted for n and m integer powers of σ. The extension for general dimensions, together with a variant for star-coloring for symmetric Jacobians, was implemented in the `CasADi` framework[1]. In that framework, 64 dependency sweeps are evaluated concurrently and hence a subdivision factor of $\sigma = 64$ was chosen.

The asymptotic run-time is a factor $\sigma/(\sigma - 1)$ worse than the straightforward approach for a fully dense Jacobian (i.e. worst-case). However, for a block-diagonal n-by-n matrix with a blocksize σ, the run-time is $\tau\sigma \log_\sigma(n)$, amounting to a change in complexity from $O(n)$ to $O(log(n))$.
The following table lists run-time results for block-diagonal matrices with blocksize 4-by-4 and shows a clear benefit for the proposed algorithm in practice:

	τ	Run-time, straightforward approach	Run-time, proposed algorithm
$n = 256$	0.11ms	0.6ms (6τ)	0.9ms (8τ)
$n = 16384$	328ms	84.0s (256τ)	1.02s (3τ).

[1] Andersson, J., Åkesson, J., and Diehl, M. CasADi – A symbolic package for automatic differentiation and optimal control. In *Recent Advances in Algorithmic Differentiation* (Berlin, 2012), S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds., Lecture Notes in Computational Science and Engineering, Springer.

[2] Gebremedhin, A. H., Manne, F., and Pothen, A. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review 47* (2005), 629–705.

[3] Giering, R., and Kaminski, T. Automatic sparsity detection implemented as a source-to-source transformation. In *Lecture Notes in Computer Science*, vol. 3994. Springer Berlin Heidelberg, 2006, pp. 591–598.

Compressed threshold pivoting for sparse symmetric indefinite systems

Jonathan Hogg and Jennifer Scott
Scientific Computing Department
STFC Rutherford Appleton Laboratory
Harwell Oxford, OX11 0QX, England.

Abstract

We are interested in the efficient and stable factorization of large sparse symmetric indefinite matrices of full rank. In this talk, we propose two new pivoting strategies that are designed to significantly reduce the amount of communication required when selecting pivots.

Most algorithms for the factorization of large sparse symmetric indefinite matrices employ supernodes, that is, a set of consecutive columns having the same (or similar) sparsity pattern in the factor. By storing only those rows that contain nonzeros, each supernode may be held as a dense $n \times p$ trapezoidal matrix. At each stage, a search is made for a pivot from the $p \times p$ leading block. If a candidate pivot is found to be unsuitable, its elimination is delayed to a later supernode, with a guarantee that all pivots will be eliminated in the final supernode. Such delayed pivots generate additional floating-point operations and storage requirements. Good scalings and orderings can reduce the number of delayed pivots but not remove the need for testing pivots for numerical stability.

With the advent of manycore processors and the growing gap between the speed of communication and computation, many algorithms need to be rewritten to reflect the changing balance in resource. As pivoting decisions must be taken in a serial fashion, they are highly sensitive to the latency and speed of any communication or bandwidth costs incurred. With current algorithms that take into account the entire candidate pivot column below the diagonal, all threads working on a supernode must endure stop-start parallelism for every column of the supernode.

We seek to develop effective pivoting strategies that significantly reduce the amount of communication required by compressing the necessary data into a much smaller matrix that is then used to select pivots. A provably stable algorithm and a heuristic algorithm are presented; we refer to these algorithms as *compressed threshold pivoting* algorithms. The heuristic algorithm is faster than the provably stable alternative and it more accurately approximates the behaviour of traditional threshold partial pivoting in terms of modifications to the pivot sequence. While it can demonstrably fail to control the growth factor for some pathological examples, in practice, provided it is combined with appropriate scaling and ordering, it achieves numerical robustness even on the most difficult practical problems. Further details are given in [1].

This work was funded by EPSRC grant EP/I013067/1.

[1] J. D. Hogg and J. A. Scott. Compressed threshold pivoting for sparse symmetric indefinite systems. Technical Report RAL-TR-2013-P-007, Rutherford Appleton Laboratory, 2013.

Maxflow, min-cuts and multisectors of graphs

Cleve Ashcraft * Iain Duff †

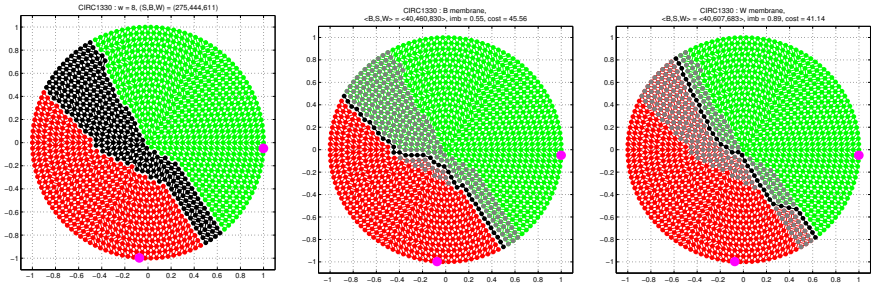
July 7, 2014

Abstract

The vertex bisection problem can be addressed with matching and network flow techniques. Pothen and Fan [2] and Ashcraft and Liu [1] showed how matching, the Dulmage-Mendelsohn decomposition, and maxflow allowed one to find one or more minimal weight vertex separators chosen from a subset of vertices that form a wide separator.

For unit weight bipartite graphs, we can use matching and Dulmage-Mendelsohn. When the vertices do not have constant weight, or when the subgraph is not bipartite, we must solve maxflow over a network to find minimal weight separators.

Here are the mechanics for vertex bisection. The set of candidate vertices form a wide separator, shown as black vertices in the leftmost figure below. There are two subdomains, shown as red and green. We associate a source node with the red subdomain and a sink node with the green subdomain and use these together with the vertices of the wide separator to construct a network on which we will run our maxflow algorithms.



Wide separator Membrane B Membrane W

Each vertex in the wide separator is identified with two nodes of the network that are joined by an arc in the network that has finite capacity. Other arcs, representing edges between wide separator vertices, or connecting the source and sink to vertices,

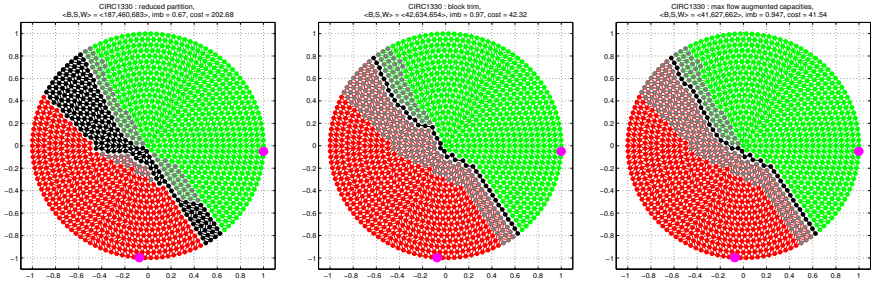
*Livermore Software Technology Corporation, 7374 Las Positas Road, Livermore, CA 94550. cleve@lstc.com

†STFC Rutherford Appleton Laboratory, Harwell Oxford, OX11 0QX UK and CERFACS, Toulouse, France. iain.duff@stfc.ac.uk

have infinite capacity. When we have found a maxflow through the network, any arc in a min-cut must have finite flow and will correspond to a vertex in the wide separator.

We conduct a search from the source to find a min-cut (middle figure above), and also from the sink (right figure above) to find a possibly different min-cut. Both induce a minimal weight vertex separator chosen from vertices in the wide separator. Each cut induces a partition, and we can choose the better of the two based on the partition with better balance.

When the two min-cuts are identical the partition is unique. When the min-cuts are not identical, they trim the wide separator and we call these sets "membrane separators" each associated with a single subdomain. These membrane separators are minimal. If we move vertices from the wide separators to the domains, we have a reduced partition, where the new wide separator is smaller (or no larger) than the original. The left plot below shows the reduced wide separator where the vertices in grey will move into the two subdomains.



Reduced partition Block trimming Augmented capacities

We cannot use our maxflow algorithm again, there is no new information to be given by the two membrane separators. But we can use other algorithms. Here are two examples.

- Block trimming creates a minimal separator. Its run time is linear in the width of the wide separator.
- We augment the capacities to take into account the balance of the partition. Using these augmented capacities we solve maxflow and generate a second, further reduced partition. On the right we see that the reduced partition is actually minimal.

The issues become more interesting and the solutions less satisfactory when we consider three or more subdomains, where the separator is a multisector, not a bisector.

- Maxflow generates a membrane bisector around each subdomain.
- The union of the membrane bisectors may be a strict subset of the multisector, i.e., there may be vertices in the multisector adjacent to no domain.
- Maxflow followed by block trimming produces a minimal multisector.
- Maxflow with augmented capacities is also a good alternative, given a proper definition of augmenting for balance.

This is work in progress.

References

[1] Cleve Ashcraft and Joseph W. H. Liu. Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement. *SIAM Journal on Matrix Analysis and Applications*, 19(2):325–354, 1998.

[2] A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16:303–324, 1990.

Abstract for CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing

Nested dissection with balanced halo

Astrid Casadei[1,3], Pierre Ramet[1,3], and Jean Roman[1,2]

[1]INRIA Bordeaux Sud-Ouest & CNRS (LaBRI UMR 5800)
[2]Bordeaux Institute of Technology (IPB)
[3]Bordeaux University

Nested Dissection (ND) has been introduced by A. George in 1973 [2] and is a well-known and very popular heuristic for sparse matrix ordering to reduce both fill-in and operation count during Cholesky factorization. This method is based on graph partitioning and the basic idea is to build a "small separator C" associated with the original matrix in order to split the remaining vertices in two parts A and B of "almost equal sizes". The vertices of the separator C are ordered with the largest indices, and then, the same method is applied recursively on the two subgraphs induced by A and B. Good separators can be built for classes of graphs occurring in finite element problems based on meshes which are special cases of bounded density graphs or more generally of overlap graphs. In d-dimension, such n-node graphs have separators whose size grows as $\mathcal{O}(n^{(d-1)/d})$. In this presentation, we focus on the cases $d = 2$ and $d = 3$ which correspond to the most interesting practical cases for numerical scientific applications. ND has been implemented by graph partitioners such as MeTiS or Scotch[6].

Moreover, ND is based on a divide and conquer approach and is also very well suited to maximize the number of independent computation tasks for parallel implementations of direct solvers. Then, by using the block data structure induced by the partition of separators in the original graph, very efficient parallel block solvers have been designed and implemented according to supernodal or multifrontal approaches. To name a few, one can cite MUMPS, PaStiX and SuperLU. However, if we examine precisely the complexity analysis for the estimation of asymptotic bounds for fill-in or operation count when using ND ordering[5], we can notice that the size of the halo of the separated subgraphs (set of external vertices adjacent to the subgraphs and previously ordered) play a crucial role in the asymptotic behavior achieved. The minimization of the halo is in fact never considered in the context of standard graph partitioning and therefore in sparse direct factorization studies.

In this presentation, we will focus on hybrid solvers combining direct and iterative methods and based on domain decomposition and Schur complement approaches. The goal is to provide robustness similar to sparse direct solvers, but memory usage more similar to preconditioned iterative solvers. Several sparse solvers like HIPS, MaPHyS, PDSLIN and ShyLU implement different versions of this hybridification principle.

In this context, the computational cost associated to each subdomain for which a sparse direct elimination based on ND ordering is carried out, as well as the computational cost of the iterative part of the hybrid solver, critically depend on the halo size of the subdomains. However, to our knowledge, there does not exist a domain decomposition tool leading to a good balancing of both the internal node set size and the halo node size. Standard partitioning techniques, even by using k-way partitioning approach, which intends to construct directly a domain decomposition of a graph in k sets of independent vertices[4], do not lead in general to good results for the two coupled criteria, and for general irregular graphs coming from real-life scientific applications.

For this purpose, we revisit the original algorithm introduced by Lipton, Rose and Tarjan [5] in 1979 which performed the recursion for nested dissection in a different manner: at each level, we apply recursively the method to the subgraphs induced by $A \cup C$ on one hand, and $B \cup C$ on the other hand. In these subgraphs, vertices already ordered (and belonging to previous separators) are the halo vertices. The partition of these subgraphs will be performed with three objectives: balancing of the two new parts A' and B', balancing of the halo vertices in these parts A' and B' and minimizing the size of the separator C'.

We implement this strategy in the SCOTCH partitioner. SCOTCH strategy is based on the multilevel method[3] which consists in three main steps: the (sub)graph is coarsened multiple times until it becomes small enough, then an algorithm called greedy graph growing is applied on the coarsest graph to find a good separator, and finally the graph is uncoarsened, projecting at each level the coarse separator on a finer graph and refining it using the Fiduccia-Mattheyses algorithm[1].

We have adapted the multilevel framework of Scotch in order to take into account the halo vertices from orginal to coarsest graph. Moreover, we have worked on two variants of greedy graph growing. The first one is called *double greedy graph growing* (DG). Its principle is to pick two seed vertices as far as possible among the halo, and to make parts A and B grow from them, with attention paid to keep halo balanced among the growing parts. The second approach, called *halo-first greedy graph growing* (HF), works in a first stage on the sole halo graph, finding a separator of it. Once it is done, it defines the two halo parts A_h and B_h as two sets of seeds and make these sets grow in the whole graph to build A and B. Finally, we have also changed the Fiduccia-Mattheyses refinement algorithm (FM) in order to preserve the good balancing in the finer graphs. Our algorithms will be explained more deeply during the presentation.

We made tests on a pool of 30 graphs from 140,000 to over 10 millions vertices. We measured both halo and domain interior imbalance. On 16 domains, our algorithms achieve an average gain of 39% on the halo imbalance, while not degrading interior imbalance. We increased the number of domains up to 512 on our biggest graphs and still got very good gains, in particular with HF. More detailed results will be given in the presentation.

References

[1] C.M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Conference on Design Automation*, pages 175–181, June 1982.

[2] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[3] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, pages 28–28, 1995.

[4] G. Karypis, V. Kumar, and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[5] R. J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2), 1979.

[6] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, volume 1067 of *Lecture Notes in Computer Science*, pages 493–498. Springer, 1996.

Adaptive FMM for fractal sets

key words: *adaptive fast multipole method, fractal set, fractal dimension, linear complexity*

Hadi Pouransari and Eric Darve

In the realm of scientific computing there are variety of situations where calculation of pairwise interaction among N points is of interest. Consider N particles (e.g., N masses) are located at positions $\{x_i\}$ in some metric space $\mathcal{M}$, and the net contribution of these particles at some observation point y is calculated by a sum of the form:

$$f(y) = \sum_{i=1}^{N} K(x_i, y)\sigma_i \tag{1}$$

where K is some $\mathcal{M} \times \mathcal{M} \to \mathbb{R}$ function called kernel, and σ_i is the intensity of the i'th particle.

This well-studied problem has broad application in various fields such as molecular dynamics, fluid dynamics, celestial mechanics, and plasma physics. More broadly, the problem of computing N^2 interactions among N points or N variables appears in the boundary element methods, problems involving radial basis functions or in probability theory to describe dense covariance matrices.

The fast multipole formulation introduced by Greengard and Rokhlin approximates a matrix-vector multiplication of the above form with desired accuracy in $\mathcal{O}(N)$ time. Several works have extended the algorithm by studying different kernels, analyzing the approximation error, introducing parallel implementation techniques, etc. [1]

The adaptive FMM refers to the case where the particle distribution, and the corresponding hierarchical tree, are not uniform. The adaptive FMM and various aspects of its parallel implementation on different machines are an ongoing topic of research. [2]

It is known that the adaptive FMM algorithm maintains the $\mathcal{O}(N)$ complexity irrespective of the point distribution [3]. This requires a modification to the original FMM. We will present a new proof for the linear complexity of the adaptive FMM for any distribution of the points. This also will make it apparent what modifications to the original FMM are required to ensure $\mathcal{O}(N)$ complexity for general particle distributions.

Previous works have limited their analysis to very specific point distributions. The key point essentially is the manner in which points are distributed, in a non-uniform adaptive setting, as N goes to infinity. In the uniform case, the issue of increasing N presents no particular difficulty. We can simply increase the density of points uniformly, and study how accuracy and parameters in the FMM are adjusted as a function of N. However, the non-uniform case is more difficult. One essential point is describing the process of adding points so that $N \to \infty$. The adaptive test cases considered by most previous works fall broadly into the following categories:

1. A small number of subregions are picked (e.g., n spheres) and points are progressively added to each subregion by distributing them with some smooth distribution (e.g., uniform, Gaussian, etc.) inside each region. Then the diameter and distance between regions are varied. [2]

2. Manifolds are considered, that is surfaces or lines. Then points are added on these manifolds again using a randomly uniform distribution.

3. Points are chosen such that they accumulate at some location (e.g., $x_i = 1/i^2$).

Complex non-adaptive cases have also been considered, but in those particular cases N was fixed.

All these cases represent only a small set of possible situations. There are many more ways to create non-uniform distribution of points. We focused on the third case, in which points accumulate.

However we extended this situation to points that essentially accumulate at an infinite number of locations. This naturally leads to fractal sets.

There are several practical problems involving fractal sets. Notoriously models of the universe, and antennas with fractal geometries that take advantage of the space-filling properties of fractal curves. In our numerical benchmarks we have considered the generalized Cantor sets that are constructed based on a recursive definition. The points x_i are generated by going through k iterations of this recursive process. As $k \to \infty$, N goes to infinity in a well-defined manner.

Fractal sets are often characterized in terms of their dimension, for example the fractal dimension, box-counting dimension, or Hausdorff dimension. We studied how parameters in the FMM such as the optimum total number of levels or the maximum number of points per leaf cells can be optimized as a function of the dimension of the set. We considered dimensions ranging continuously from 1 to 3, and exponential dependence of cost on the dimension is presented. Other details of the distribution appear to be less important. Our analysis is based both on mathematical bounds and estimates, as well as numerical benchmarks and investigations.

Theoretical estimates for optimal parameters can be found for uniform distributions, while, for a generic adaptive distribution, not much is known. Most implementations, if not all, manually or heuristically tune parameters to get the optimum values of parameters. In order to analyze arbitrary point distributions, we have characterized and categorized different distributions. We organized all possible fractal point distributions in terms of the fractal dimension of the set. We focused on sets for which the box-counting dimension is defined, which is the case for self-similar fractal sets for example. Note that the box-counting dimension cannot be defined for all sets. The Hausdorff dimension always exists but it cannot be directly related to the FMM (because of the oct-tree decomposition of the FMM) so that the Hausdorff dimension is in general not a good parameter to consider when optimizing FMM parameters. We will discuss these technical points in more details. Specifically in our numerical benchmarks, one of our main examples is a triple tensor product of generalized Cantor sets, which provide all range of box-counting dimensions varying continuously from 1 to 3 (in this case box counting is the same as Hausdorff).

We also present a new strategy to build the adaptive tree. We focused on the criterion used to determine whether a cell needs to be further subdivided or not. The original bisection algorithm uses one threshold value for subdivision, which is the maximum number of particles per leaf node. However, we used two threshold values simultaneously, namely, the maximum level of the tree, and the maximum number of particles per leaf nodes. Essentially, by tuning parameters in the dual threshold method we can transform some expensive operations such as M2P (multipole to particle) and P2L (particle to local) to a cheaper operation M2L (multipole to local). Better performance of the proposed scheme is demonstrated.

The aforementioned particle distributions were studied along with a detailed counting of the number of floating point operations. The calculation begins with some standard cases (e.g., uniform, spiral, etc.), and then extends to general fractal sets.

References

[1] William Fong and Eric Darve, The black-box fast multipole method, *Journal of Computational Physics* (2009) 8712–8725.

[2] Lexing Ying, George Biros, Denis Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, *Journal of Computational Physics* (2004) 591-625.

[3] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J White Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations for potential theory, *SIAM Journal on Scientific Computing*, 15(3), 713-735

Multicenter method

Julie Anton * Cleve Ashcraft † Pierre L'Eplattenier ‡

In this abstract, we present a new method, called the multicenter method, that computes efficiently a long range force in a N body problem. Being kernel-independent, it is more general than the well known multipole method. The multicenter method is based upon the idea of defining a subset of sources which we call "centers" and computing a weighted contribution of these centers only. Unlike the multipole method, we have several centers and we compute a polynomial of degree 1 (the number of selected sources depends on the expected accuracy). In this abstract, we will present some of the linear algebra issues raised by the multicenter method : how the centers and the associated weights are defined. We will also present some results on the computation of the electromagnetic field lines which was one of the physical contexts for this work.

1 Define the centers

Let us define a set of n sources $\mathcal{K}$ such that the sources lie inside a ball $B_{\mathbf{c},\mathrm{r}}$ ($\mathbf{c}$ being the center and r the radius of the ball) and a set $\mathcal{T}$ of n target points distributed on a sphere $S_{\mathbf{c},\alpha\mathrm{r}}$ with $\alpha \in \mathbb{R}$, $\alpha > 1$, α is the separation criterion between the sets $\mathcal{K}$ and $\mathcal{T}$. The number of target points should be greater than the number of source points.

The idea is to compute the matrix of the kernels between the source points and some target points far away enough from the source. More precisely, we compute the matrix $A_{\mathcal{T},\mathcal{K}}$ of the kernel between the 2 sets of points $A_{\mathcal{T},\mathcal{K}}(i,j) = k(x_{T_i}, x_{K_j})$ and evaluate its rank in order to determine the leading source points i.e. the centers. When the distance between the sets $\mathcal{K}$ and $\mathcal{T}$ increases, the rank of $A_{\mathcal{T},\mathcal{K}}$ decreases, therefore, we need fewer source points to get a good representation of the entire source set. In order to find those points, we perform a QR factorization with column pivoting of $A_{\mathcal{T},\mathcal{K}}$:

$$A_{\mathcal{T},\mathcal{K}} = Q_{\mathcal{T},\mathcal{T}} \begin{bmatrix} R_{\mathcal{K},\mathcal{K}} \\ 0_{\mathcal{T}\setminus\mathcal{K},\mathcal{K}} \end{bmatrix} \Pi^T_{\mathcal{K},\mathcal{K}} \tag{1}$$

where Q is orthogonal, R is upper triangular and Π is a permutation matrix such that :

$$|r_{1,1}| \geq |r_{2,2}| \geq \cdots \geq |r_{n_K,n_K}| \text{ and } \forall i\, |r_{i,i}| \geq ||R_{i:j,j}||_2\, j = i+1, \ldots, n_K \tag{2}$$

Let us define a low-rank threshold ϵ, the rank of $A_{\mathcal{T},\mathcal{K}}$ is given by :

$$r(\epsilon) = min(r \in \mathbb{N} : \|R_{r,r:n_K}\|_2 < \epsilon\, max(\|R_{i,i:n_K}\|_2)_{i=1,\ldots,n_K}) \tag{3}$$

The leading r columns of $A_{\mathcal{T},\mathcal{K}}\Pi_{\mathcal{K},\mathcal{K}}$ approximate $A_{\mathcal{T},\mathcal{K}}$ to an accuracy $O(\sigma_{r+1(A)})$. Let's define the matrix restricted to the leading columns r, $A_{\mathcal{T},\mathcal{C}}$: $A_{\mathcal{T},\mathcal{K}}\Pi_{\mathcal{K},\mathcal{K}} = [A_{\mathcal{T},\mathcal{C}} A_{\mathcal{T},\mathcal{K}\setminus\mathcal{C}}]$. The r first points of $\mathcal{K}$ define the r centers.

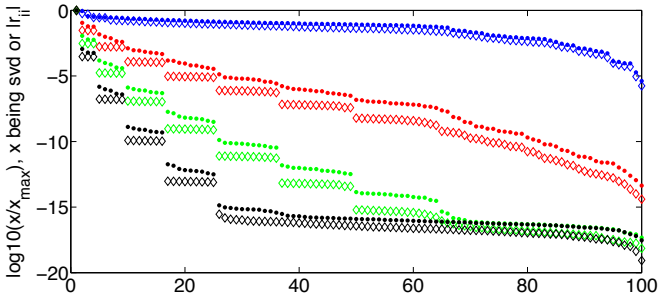


Figure 1: Singular values of $A_{\mathcal{T},\mathcal{K}}$

As a first simple example, we consider a set of source points $\mathcal{K}$ homogeneously distributed inside a ball and a set of target points $\mathcal{T}$ homogeneously distributed on a sphere such that $R_{sphere} = \alpha R_{ball}$ ($\alpha = 1, 10, 100, 1000$ for the blue lines, the red ones, the green ones and the black ones respectively). Figure (1) shows the singular values of the kernel matrix $A_{\mathcal{T},\mathcal{K}}$ (represented by ♦) as well as the values $|r_{ii}|$ resulting from the RRQR of $A_{\mathcal{T},\mathcal{K}}$ (represented by .), see equation (1).
The singular values are well separated as long as the set of points $\mathcal{T}$ is far enough from the source points set. Depending on the tolerance ϵ, we obtain either 1, 4, 9, 16... centers. The values $|r_{ii}|$ provided by (1) follow the same behaviour than the singular values except from the clustering which is not as good.

*Livermore Software Technology Corporation, 7374 Las Positas Road, Livermore, CA 94550. julie@lstc.com
†Livermore Software Technology Corporation, 7374 Las Positas Road, Livermore, CA 94550. cleve@lstc.com
‡Livermore Software Technology Corporation, 7374 Las Positas Road, Livermore, CA 94550. pierre@lstc.com

2 Define the weights

As seen before, the rank of $A_{\mathcal{T},\mathcal{K}}$ allows to define a subset of the source points : the centers. Once the centers are defined, the correlation between them and the other source points are defined by a "barycentric matrix". In order to minimize the error, the barycentric $B_{\mathcal{C},\mathcal{K}}$ matrix is defined as follows :

$$A_{\mathcal{T},\mathcal{K}}\Pi_{\mathcal{K},\mathcal{K}} = \begin{bmatrix} Q_{\mathcal{T},\mathcal{C}} & Q_{\mathcal{T},\mathcal{K}\setminus\mathcal{C}} \end{bmatrix} \begin{bmatrix} R_{\mathcal{C},\mathcal{C}} & R_{\mathcal{C},\mathcal{K}\setminus\mathcal{C}} \\ 0_{\mathcal{K}\setminus\mathcal{C},\mathcal{C}} & R_{\mathcal{K}\setminus\mathcal{C},\mathcal{K}\setminus\mathcal{C}} \end{bmatrix} \tag{4}$$

$$\approx \begin{bmatrix} Q_{\mathcal{T},\mathcal{C}}R_{\mathcal{C},\mathcal{C}} & Q_{\mathcal{T},\mathcal{C}}R_{\mathcal{C},\mathcal{K}\setminus\mathcal{C}} \end{bmatrix} \tag{5}$$

$$= Q_{\mathcal{T},\mathcal{C}}R_{\mathcal{C},\mathcal{C}} \begin{bmatrix} I_{\mathcal{C},\mathcal{C}} & R_{\mathcal{C},\mathcal{C}}^{-1}R_{\mathcal{C},\mathcal{K}\setminus\mathcal{C}} \end{bmatrix} \tag{6}$$

$$= A_{\mathcal{T},\mathcal{C}}B_{\mathcal{C},\mathcal{K}} \tag{7}$$

where $B_{\mathcal{C},\mathcal{K}} = \begin{bmatrix} I_{\mathcal{C},\mathcal{C}} & \widehat{R}_{\mathcal{C},\mathcal{K}\setminus\mathcal{C}} \end{bmatrix} \equiv \begin{bmatrix} I_{\mathcal{C},\mathcal{C}} & R_{\mathcal{C},\mathcal{C}}^{-1}R_{\mathcal{C},\mathcal{K}\setminus\mathcal{C}} \end{bmatrix}$

By summing up the row entries of $B_{\mathcal{C},\mathcal{K}}$, we obtain the weight associated to each center. If the matrix $A_{\mathcal{T},\mathcal{K}}$ were full rank, then there would be as many centers as source points in set $\mathcal{K}$ and the barycentric matrix $B_{\mathcal{C},\mathcal{K}}$ would be the identity matrix. To compute the resulting long-range force at one point P far away from the sources, we only need to compute the sum of the weighted interactions between P and the centers.

3 Results

In this section, we compare 3 methods : the multipole method, the multicenter method and the direct method which consists in taking into account the contributions of each source. Unlike the multipole and the multicenter methods, the direct method does not do any approximation and, therefore, constitues our reference.

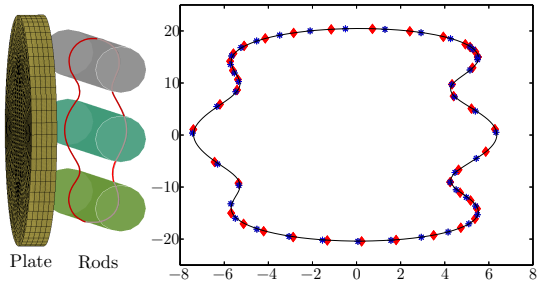


Figure 2: Direct method: −, multipole method: ♦, multicenter method: ∗

Let's consider a conductor formed by 3 rods and a plate. An electric current circulates into the 3 tubes which induces a current in the plate. An electromagnetic field is generated around the 3 tubes. We want to visualize a magnetic field line close to the rods. To do that, we need to solve the magnetic field line equations which requires to compute the magnetic field $\vec{B}$. $\vec{B}$ is the sum of the contributions of each source point which can be computed either directly or by an approximation method (FMM or multicenter). Figure (2) shows one of the magnetic field lines generated around the rods on the left hand side. On the right hand side, only the magnetic field line is represented, the results given by the three methods are placed on top of each other. Both multipole and multicenter methods ensure a relatively good accuracy, the results are comparable to those obtained with the direct method.

For the next problem, we want to study the interaction between two sets of points instead of one set of points and one point only. It is of importance when it comes to build the BEM-FEM system to solve for the electromagnetic fields for example. The BEM system is dense and solved through an iterative method such as GMRES or PCG, therefore using either the FMM or the multicenter method can be useful to accelerate the assembly of the matrix as well as the operations for the matrix-vector product.

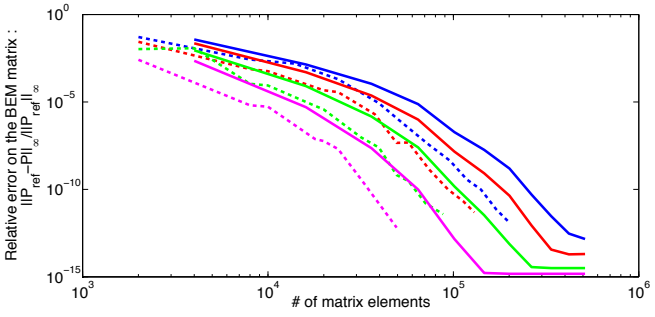


Figure 3: Multicenter versus FMM

Here, we consider 2 cubes of 1000 points each. We want to compute the kernel matrix between those cubes with both methods and compare their cost in terms of matrix-vector product. Figure (3) shows the relative error induced on the kernel matrix against the number of entries in the matrix for different separations (12, 20, 50, 200 for the blues lines, the red ones, the green ones and the pink ones respectively). The dashed lines correspond to the multicenter method whereas the solid lines represent the FMM. For a given distance between the cubes, the multicenter method requires less entries than the FMM to reach the same accuracy on the kernel matrix therefore the low rank representation obtained is better and the operation for the matrix-vector multiply will be more efficient.

In order to improve the efficiency of the matrix-vector product, following the example of the FMM, we intend to add the multilevel aspect to the multicenter method. Work is in progress.

Title: Computing an estimate of Trace(A^{-1}) using matrix sparsification and hierarchical probing.

Jesse Laeuchli and Andreas Stathopoulos

One problem that occurs frequently in numerical linear algebra is the computation of the functional of matrices that are too large to calculate directly. One such functional is the trace of the inverse of A, which occurs frequently in scientific computation. Several approaches have been proposed for this problem before. In the case of small matrices, a factorization approach can solve the problem exactly, but this becomes impractical for many matrices of interest, due to size. Another approach that can be taken is probing. Many matrices exhibit a relationship between the non-zero structure of A^k and A^{-1} after a dropping of some tolerance has been applied. As k increases, the values that must be dropped in A^{-1} for the non-zero structures to match decrease as well. Probing takes advantage of this structure by coloring A^k. By permuting A^k so that nodes that share a color are adjacent, a block diagonal structure consisting of zeros surrounding the nodes sharing the same color is created. The value of these nodes can then be recovered by probing, that is, by creating a probing vector consisting of all ones for nodes sharing the same color, and zeros everywhere else. Because of the block diagonal structure created by the permutation, the value of the diagonals can be recovered using only n vectors, where n is the number of colors used to color A^k. Since the structure of A^{-1} approximates the structure of A^k, if an iterative solver is used, these probing vectors can also be used to recover the trace of A^{-1}.

This approach has two major shortcomings. First, since the non-zero structure of A^{-1} only approximates that of A^k, applying the coloring of A^k to A^{-1} will likely yield a coloring that is not exactly correct for A^{-1}, leading to errors in the computed value of the trace, since the block surrounding the diagonals being probed in A^{-1} will not be all zero. Further, is not clear how large k must be in order to obtain a desired level of accuracy for the trace estimation. However, if after computing a trace approximation with a given k the accuracy of the trace computation is too low, a higher k must be selected, and the approximation recomputed. With classical probing, this means that the results of all the previously preformed solves must be discarded, since the intersection between sets of probing vectors for the two levels of colors is likely to be empty. The other major shortcoming of this method is that for matrices with an associated graph which is highly connected, A^k is likely to become dense very quickly. This means that A^k will contain many colors, which will require too many probing vectors to be practical.

Our research addresses both these issues. First, we attempt to deal with the problem of having to throw out all previously computed probing vector results when proceeding to a higher value of k. This can be addressed by using probing using vectors that span the same space as the original probing vectors, but are subsets of each other. One such basis is the kronecker product of DFTs. Using these matrices as building blocks, it is possible to create a set of probing vectors that work for two different coloring levels, and are nested subsets of each other. The drawback to this method is that in order for this set of probing vectors to be applicable, the generated colors must have two properties. First, they must be hierarchical, that is, if a pair of colors did not share a color at a previous level k, they cannot later share a color at level k+1. Secondly, each color at the k-th level must split into the same number of colors at the k+1th level. In general, two colors independently generated colorings for levels k and k+1, will not have either of these properties.

Since this property does not in general hold for two arbitrary colorings, we modify the colorings created for two different levels, k and k+1, in a post processing stage. This is done in two steps, first by examining where every color block ends for the coloring at level k, and then splitting any blocks that cross that boundary in the k+1th coloring. This ensures that no nodes that had a different color at level k, share a color at the next level. After this, each block of color in the k level coloring is iterated over in order to compute the maximum number of colors that block is split into. Then additional colors are created in the k+1th level coloring by splitting colors apart, until the number of colors each block from the kth level is split into are the same. This approach fixes the problem of not being able to reuse any of the previous probing vectors, but it creates more colors then the minimum needed. If the number of colors is already too large, as in the case of a strongly connected matrix, this algorithm makes the problem worse.

To combat this problem, we apply matrix sparsifcation. While we are still experimenting with which sparsification approach is best, we have developed one method that yields useful results. For each block at the k-th level, we examine all connections between nodes that appear at the k+1th level, and sort them by weight. Each edge is added into our sparse representation of the block until a limit is reached. The resultant coloring is then forced to be hierarchical in the manner previously described. The trace approximation computed using this coloring will not be as good as if the actual A^{k+1} coloring were used, but will be better than the approximation for A^k, and require fewer uses of the solver, since there are fewer probing vectors. Further, as the number of colors that are allowed in each block is increased, the results begin to approximate A^{k+1} better, allowing for control over the tradeoff between sparsity and accuracy.

Parallel distributed-memory simplex for large-scale stochastic LP problems

J. A. J. Hall M. Lubin

19 June 2014

Abstract

Although parallel efficiency using the revised simplex method has not been achieved for general large sparse LP problems, this talk will show how the particular structure of stochastic LP problems gives scope for efficient data parallelism. Issues relating to algorithmic design and data distribution will be discussed. Results obtained on a large cluster and supercomputer using a distributed-memory implementation are presented for stochastic LP problems with up to 10^8 variables and constraints.

Keywords: Simplex method; Parallel computing; Stochastic optimization; Block-angular

ACM Classification: 90C05; 90C15; 68W10

1 Introduction

In this talk, we present a parallel solution procedure based on the revised simplex method for linear programming (LP) problems with a special structure of the form

$$
\begin{array}{lrcccccccccl}
\text{minimize} & \boldsymbol{c}_0^T\boldsymbol{x}_0 & + & \boldsymbol{c}_1^T\boldsymbol{x}_1 & + & \boldsymbol{c}_2^T\boldsymbol{x}_2 & + & \ldots & + & \boldsymbol{c}_N^T\boldsymbol{x}_N & & \\
\text{subject to} & A\boldsymbol{x}_0 & & & & & & & & & = & \boldsymbol{b}_0, \\
& T_1\boldsymbol{x}_0 & + & W_1\boldsymbol{x}_1 & & & & & & & = & \boldsymbol{b}_1, \\
& T_2\boldsymbol{x}_0 & & & + & W_2\boldsymbol{x}_2 & & & & & = & \boldsymbol{b}_2, \\
& \vdots & & & & & & \ddots & & & & \vdots \\
& T_N\boldsymbol{x}_0 & & & & & & & + & W_N\boldsymbol{x}_N & = & \boldsymbol{b}_N, \\
& \boldsymbol{x}_0 \geq \boldsymbol{0}, & \boldsymbol{x}_1 \geq \boldsymbol{0}, & & \boldsymbol{x}_2 \geq \boldsymbol{0}, & \ldots, & \boldsymbol{x}_N \geq \boldsymbol{0}. & & & & &
\end{array}
$$

Very large instances of such problems have been considered to be too big to solve with the simplex method; instead, decomposition approaches based on Benders decomposition or, more recently, interior-point methods are generally used. However, these approaches do not provide optimal basic solutions which allow for the efficient hot-starts required, for example, in a branch-and-bound context, and can provide important sensitivity information. Our approach exploits the dual block-angular structure of these

problems inside the linear algebra of the revised simplex method in a manner suitable for high-performance distributed-memory clusters or supercomputers. While the focus is on stochastic LPs, the work is applicable to all problems with a dual block-angular structure. Our implementation is competitive in serial with highly efficient sparsity-exploiting simplex codes and achieves parallel efficiency when using up to 128 cores and runs up to 100 times faster than the leading open-source serial solver. Additionally, very large problems with hundreds of millions of variables have been successfully solved to optimality.

2 Data parallel linear algebra

The structure of the basis matrix B and matrix N corresponding to the nonbasic variables permits distribution of the data and computation relating to solution of systems of equations involving B and products involving N. Minimal duplicated computation leads to relatively little data transfer being required. The numerical linear algebra is was developed from the COIN-OR utilities and is efficient with respect to the hyper-sparsity present in the problems. Thus the implementation is comparable with world-class open source revised simplex solvers.

3 Test problems, results and conclusions

The principal source of test problems are deterministic LP problems of scenarios resulting from sampling a minimum expected cost stochastic model of wind power generation in the state of Illinois. Increasing the number of scenarios yields ever larger deterministic LP problems, allowing a range of experiments to be performed on two distributed-memory machines: a 320-node cluster of dual quad-core Xeon processors with an InfiniBand QDR interconnect and a Blue Gene/P (BG/P) supercomputer with 40,960 nodes of quad-core 850 MHz PowerPC processors. The nature of the scenario sampling allows a bootstrapping approach to be used to deduce an advanced initial basis, considerably reducing the solution time which would be required otherwise. The largest instance had 463,113,276 variables and 486,899,712 constraints and was solved to optimality: possibly the largest LP ever solved using the simplex method.

This is the largest-scale parallel sparsity-exploiting revised simplex implementation that has been developed to date and the first truly distributed solver. It is built on novel analysis of the linear algebra for dual block-angular LP problems when solved by using the revised simplex method and a novel parallel scheme for applying product-form updates.

This work was awarded the COIN-OR INFORMS 2013 Cup.

Contention Bounds for Combinations of Computation Graphs and Network Topologies

Grey Ballard
Sandia National Laboratories
gmballa@sandia.gov

James Demmel
UC Berkeley
demmel@cs.berkeley.edu

Andrew Gearhart
UC Berkeley
agearh@cs.berkeley.edu

Benjamin Lipshitz*
UC Berkeley
lipshitz@cs.berkeley.edu

Oded Schwartz
Hebrew University†
odedsc@cs.huji.ac.il

Sivan Toledo
Tel-Aviv University
stoledo@tau.ac.il

Good connectivity of the inter-processor network is necessary for efficient parallel algorithms. Insufficient graph-expansion of the network provably slows down specific parallel algorithms that are communication intensive. While parallel algorithms that ignore network topology can suffer from contention along network links, for particular combinations of computations and network topologies, costly network contention may be inevitable, even for optimally designed algorithms. In this paper we obtain novel lower bounds on this *contention cost*.

Most previous communication cost lower bounds for parallel algorithms utilize *per-processor* analysis. That is, the lower bounds establish that some processor must communicate a given amount of data. These include classical matrix multiply, direct and iterative linear algebra algorithms, FFT, Strassen and Strassen-like fast algorithms, graph related algorithms, N-body, sorting, and others (cf. [1, 14, 12, 18, 15, 5, 3, 8, 11, 2, 16, 20, 10, 19]). By considering the network graphs, we introduce communication lower bounds for certain computations and networks that are tighter than those previously known. We translate per-processor bandwidth cost lower bounds to contention cost lower bounds by bounding the communication needs between a subset of processors and the rest of the processors for a given parallel algorithm (defined by a computation graph and work assignment to the processors), and divide by the available bandwidth, namely the words that the network allows to communicate simultaneously between the subset and the rest of the graph.

Contention Lower Bound. Consider a parallel algorithm run on a distributed-memory machine with P processors and connected via network graph G_{Net}. The *per-processor bandwidth cost* W_{proc} is the maximum over processors $1 \le p \le P$ of the number of words sent or received by processor p. Further, the *contention cost* W_{link} is the maximum over edges e of G_{Net} of the number of words communicated along e.

We prove the lower bound using graph expansion analysis. Recall that the small set expansion $h_s(G)$ of a graph $G = (V, E)$ is the minimum normalized number of edges leaving a set of vertices of size at most s. For $s \le |V(G)|/2$, we have

$$h_s(G) = \min_{S \subseteq V(G), |S| \le s} \frac{|E(S, V \setminus S)|}{|E(S)|}$$

where $E(S)$ is the set of edges that have at least one endpoint in vertex subset S and $E(S, V \setminus S)$ is the set of edges with only one endpoint in S. In this note, we provide the contention cost lower bound for regular networks:

THEOREM 1. *Consider a distributed-memory machine with P processors, each with local memory of size M, and a d-regular inter-processor network graph G_{Net}. Given a computation with input and output data size N, and lower bound on the per-processor bandwidth cost $W_{proc} = W_{proc}(P, M, N)$, for all algorithms that distribute the workload so that every processor performs $\Omega(1/P)$ of the computation, and distributing the input and output data such that every processor stores $O(1/P)$ of the data, the contention cost $W_{link} = W_{link}(P, M, N)$ is bounded below by*

$$W_{link}(P, M, N) \ge \max_{t \in T} \frac{W_{proc}(P/t, M \cdot t, N)}{d \cdot t \cdot h_t(G_{Net})}, \text{ where}$$

$$T = \{t : 1 \le t \le P/2, \exists S \subseteq V \text{ s.t. } |S| = t \text{ and } |E(S, V \setminus S)| = \Theta(h_t(G_{Net}) \cdot |E(S)|)\}.$$

PROOF. Partition the P processors into P/t subsets of size $t \in T$ (w.l.o.g., P is divisible by t), where at least one of the subsets s_t is connected to the rest of the graph with at most $d \cdot t \cdot h_t(G_{Net})$ edges. The existence of such a set s_t is guaranteed by the definition of $h_s(G_{Net})$ and T. Then s_t has a total of $M \cdot t$ local memory. By the workload distribution assumption, the processors in s_t perform a fraction $\Omega(t/P)$ of the flops, and by the data distribution assumption, s_t has local access to fraction $O(t/P)$ of the input/output. Hence we can emulate this computation by a parallel machine with P/t processors, each with $M \cdot t$ local memory, and apply the corresponding per-processor lower bound deducing that the processors in s_t require at least $W_{\text{proc}}(P/t, M \cdot t, N)$ words to be sent/received to the processors outside s_t throughout the running of the algorithm. At most $O(d \cdot t \cdot h_t(G_{Net}))$ edges connect s_t to the rest of the graph. Hence at least one edge communicates at least $\Omega\left(\frac{W_{\text{proc}}(P/t, M \cdot t, N)}{d \cdot t \cdot h_t(G_{Net})}\right)$ words. As t is a free parameter, we can pick it to maximize $W_{\text{link}}(P, M, N)$, and the theorem follows. □

Note that the memory-independent contention lower bound, $W_{\text{link}} = W_{\text{link}}(P, N)$, follows.

*Current affiliation: Google Inc.

†This work was done while at UC Berkeley.

Applications. We next demonstrate our bounds for direct dense linear algebra algorithms (including classical matrix multiplication) and fast matrix multiplication algorithms (such as Strassen's algorithm) on D-dimensional tori networks. Table 1 summarizes the contention bounds obtained by plugging in memory-dependent and memory-independent lower bounds for matrix multiplication and other linear algebra computations from [15, 6, 3] into Theorem 1 and using the properties of D-dimensional tori. The D-dimensional torus graph G_{Net} has degree $d = 2D$ and small set expansion guarantee of $h_s(G_{Net}) = \Theta\left(s^{-1/D}\right)$, see [9]. We treat D here as a constant. Table 1 summarizes the bounds.

		Mem. Dep.	Mem. Indep.
Direct Linear Algebra	W_{proc}	$\Omega\left(\frac{n^3}{PM^{1/2}}\right)$	$\Omega\left(\frac{n^2}{P^{2/3}}\right)$
	W_{link}	$\Omega\left(\frac{n^3}{P^{3/2-1/D}M^{1/2}}\right)$	$\Omega\left(\frac{n^2}{P^{1-1/D}}\right)$
Strassen and Strassen -like	W_{proc}	$\Omega\left(\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\omega_0}}\right)$
	W_{link}	$\Omega\left(\frac{n^{\omega_0}}{P^{\omega_0/2-1/D}M^{\omega_0/2-1}}\right)$	$\Omega\left(\frac{n^2}{P^{1-1/D}}\right)$

Table 1: Per-processor bounds (W_{proc}) ([15, 5, 3, 6]) vs. the new contention bounds (W_{link}) on a D-dimensional torus for classical linear algebra and fast matrix multiplication (where ω_0 is the exponent of the computational cost).

Note that of the two contention bounds, the memory-independent one always dominates in these cases:

$$W = \Omega\left(\frac{n^{\omega_0}}{P^{\omega_0/2-1/D}M^{\omega_0/2-1}} + \frac{n^2}{P^{1-1/D}}\right) = \Omega\left(\frac{n^2}{P^{1-1/D}}\right),$$

by the fact that $P \geq P_{min} \geq n^2/M$, where ω_0 is the exponent of the computational cost.

Depending on the dimension of the torus D and number of processors, the tightest bound may be one of the previously known per-processor bounds or the memory-independent contention bound. See Figure 1 for the case of Strassen bounds on torus networks of various dimensions. For example, $D = 3$ is enough for perfect strong scaling of classical matmul but Strassen may need $D = 4$. Recall that perfect strong scaling is when, for a constant problem size, doubling the number of processors halves the runtime. Note that (see Figure 1) a contention-dominated range has a smaller region of perfect strong scaling.

Future Research. In this work, we exclusively address link contention bounds for a subset of direct network topologies (the analysis of tori extends to meshes, and can be extended to hypercubes). We believe results for certain indirect network topologies (e.g. fat trees) should follow, though this requires integrating router nodes into the model.

We focus here on a subset of linear algebraic computations. Our results extend to further computations such as the $O(n^2)$ n-body problem, FFT/sorting and programs that access arrays with affine expressions.

A network may have expansion sufficiently large to preclude the use of our contention bound on a given computation, yet the contention may still dominate the communication cost. This calls for further study on how well computations and networks match each other. Similar questions have been addressed by Leiserson and others [7, 13, 17], and had a large impact on the design of supercomputer networks.

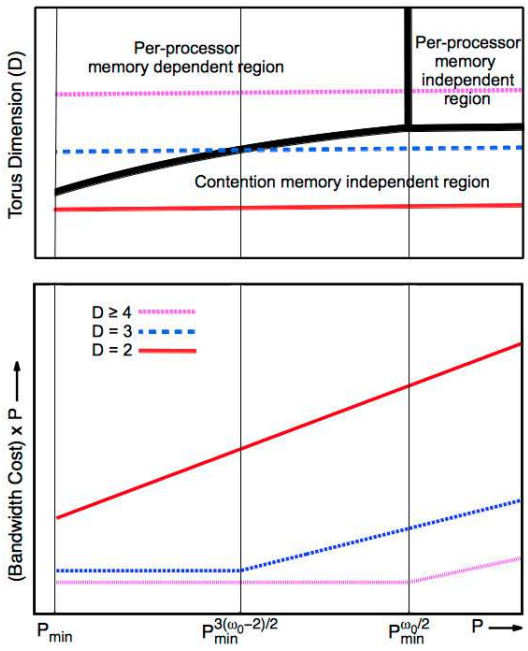


Figure 1: Communication bounds for Strassen's algorithm on D-dim. tori. Both plots share a log-scale x-axis in P. The upper plot illustrates the dominating bound, and is linear on the y-axis. The y-axis of the lower plot is log-scale, and horizontal lines represent perfect strong scaling.

Some parallel algorithms are network aware, and attain the per-processor communication lower bounds, when network graphs allow it (cf. [21] for classical matrix multiplication on 3D torus). Many algorithms are communication optimal when all-to-all connectivity is assumed, but their performance on other topologies has not yet been studied. Are there algorithms that attain the communication lower bounds for any realistic network graph (either by auto tuning, or by network-topology-oblivious tools)?

Acknowledgments. We thank Guy Kindler for pointing us to [9]. Research partially funded by DARPA Award Number HR0011-12-2-0016, the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Nokia, NVIDIA, Oracle, MathWorks and Samsung. Research is also supported by DOE grants DE-SC0004938, DE-SC0005136, DE-SC0003959, DE-SC0008700, AC02-05CH11231, and DE-SC0010200. Research is supported by grant 1045/09 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and grant 2010231 from the US-Israel Bi-National Science Foundation. This research is supported by grant 3-10891 from the Ministry of Science and Technology, Israel. This research was supported in part by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its U.S. Department of Energy Contract No. DE-AC04-94AL85000. Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and does not necessarily reflect the position or the policy of the sponsors.

1. REFERENCES

[1] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theor. Comput. Sci.*, 71:3–28, March 1990.

[2] G. Ballard, A. Buluç, J. Demmel, L. Grigori, B. Lipshitz, O. Schwartz, and S. Toledo. Communication optimal parallel multiplication of sparse random matrices. In *SPAA'13: Proceedings of the 25rd ACM Symposium on Parallelism in Algorithms and Architectures*, 2013.

[3] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 77–79, New York, NY, USA, 2012. ACM.

[4] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In G. Even and D. Rawitz, editors, *Design and Analysis of Algorithms*, volume 7659 of *Lecture Notes in Computer Science*, pages 13–36. Springer Berlin Heidelberg, 2012.

[5] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM*, 59(6):32:1–32:23, Dec. 2012.

[7] P. Bay and G. Bilardi. Deterministic on-line routing on area-universal networks. In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 297–306, 1990.

[8] G. Bilardi, M. Scquizzato, and F. Silvestri. A Lower Bound Technique for Communication on BSP with Application to the FFT. In *Euro-Par 2012 Parallel Processing*, pages 676–687. Springer, 2012.

[9] B. Bollobás and I. Leader. Edge-isoperimetric inequalities in the grid. *Combinatorica*, 11(4):299–314, 1991.

[10] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. Yelick. Communication lower bounds and optimal algorithms for programs that reference arrays - part 1. Technical Report UCB/EECS-2013-61, EECS Department, University of California, Berkeley, 2013.

[11] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick. A communication-optimal n-body algorithm for direct interactions. In *proceedings of the IPDPS*, 2013.

[12] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM J. Computing*, 29(2):416–432, 1999.

[13] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-tress. In *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 241–249, 1985.

[14] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proc. 14th STOC*, pages 326–333, New York, NY, USA, 1981. ACM.

[15] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[16] N. Knight, E. Carson, and J. Demmel. Exploiting data sparsity in parallel matrix powers computations. In *Proceedings of PPAM '13*, Lecture Notes in Computer Science. Springer (to appear), 2013.

[17] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[18] J. P. Michael, M. Penner, and V. K. Prasanna. Optimizing graph algorithms for improved cache performance. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002), Fort Lauderdale, FL*, pages 769–782, 2002.

[19] M. Scquizzato and F. Silvestri. Communication lower bounds for distributed-memory computations. *arXiv preprint arXiv:1307.1805*, 2014. STACS'14.

[20] E. Solomonik, E. Carson, N. Knight, and J. Demmel. Tradeoffs between synchronization, communication, and work in parallel linear algebra computations. Technical Report (Submitted to SPAA'14), University of California, Berkeley, Department of Electrical Engineering and Computer Science, 2013.

[21] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Euro-Par'11: Proceedings of the 17th International European Conference on Parallel and Distributed Computing*. Springer, 2011.

GENERALISED VECTORISATION FOR SPARSE MATRIX–VECTOR MULTIPLICATION

A. N. YZELMAN

This work explores the various ways in which a sparse matrix–vector (SpMV) multiplication may be vectorised. It arrives at a generalised data structure for sparse matrices that supports efficient vectorisation. This novel data structure generalises three earlier well-known data structures for sparse computations: the Blocked CRS format, the (sliced) ELLPACK format, and formats relying on segmented scans. All three formats were first explored in the 1990s, with Blocked CRS being used for CPU-based calculations, while ELLPACK and formats using segmented scans have seen renewed interest within GPU computing.

The new data structure generalises all three formats, and is relevant for sparse computations on modern architectures, since most new hardware supports vectorisation for increasingly wide vector registers. Normally, the use of vectorisation for sparse computations is limited due to bandwidth constraints. In cases where computations are limited by memory latencies instead of memory bandwidth, however, vectorisation can still help performance. Such an effort is made possible by a pair of vector instructions newly introduced with the Intel Xeon Phi instruction set: the gather and the scatter instructions.

The resulting strategy is consistent with the high-level requirements of sparse matrix computations on modern CPU-based hardware [YR14], and still allows for additional optimisation such as segmented scans and bitmasking.

Parallelisation. To illustrate the new approach, the vectorised SpMV is used within the one-dimensional method of Yzelman and Roose [YR14]. This parallel scheme uses a load-balanced row distribution so that the matrix rows of A are split in p contiguous parts, where p is the number of available hardware threads, such that each part contains roughly the same amount of nonzeroes. The output vector is divided in p parts as well, corresponding to the distribution of A. Local parts of A and y are stored separately in memory so to explicitly control data locality, while the input vector is stored in a single contiguous chunk.

Thread-local versions of A are subdivided into relatively small blocks. Blocks are ordered according to the Hilbert curve, while nonzeroes within each block retain a row-major order. This ensures a cache-oblivious traversal that benefits data reuse on the high-level caches, while minimising the amount of memory required for data storage.

The parallelisation is easily prototyped using MulticoreBSP for C. By nature of the 1D row-wise distribution, this parallel implementation of the SpMV multiplication does not require any explicit communication or synchronisation.

Generalised vectorisation. To use vector instructions, l elements must be loaded from main memory into vector registers first. Streaming loads, where l contiguous values are loaded into a register from a cache-aligned memory location, are the most efficient in terms of throughput and latency hiding. Streaming loads cannot be used in case of indirect addressing, however; this while indirect addressing is the norm for unstructured sparse computations.

The new *gather* operation provides a middle way: given a vector v and an index array $i = (i_0, \ldots, i_{l-1})$, a gather on v and i loads $(v_{i_0}, \ldots, v_{i_{l-1}})$ into a vector register. The scatter operates on an input register and an index array to perform the inverse of the gather operation.

These gather/scatters are employed within the following vectorised SpMV multiplication kernel, which operates on $p \times q$ blocks of nonzeroes, with $pq = l$:

- **for** each block **do**
- load the relative position (i, j) of the current block within A;

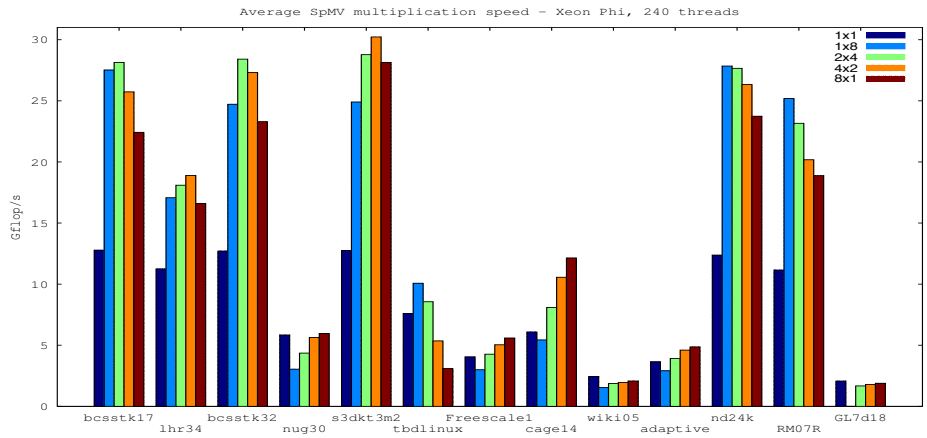


FIGURE 1. The results of the proposed 2D vectorisation of the SpMV multiplication using various matrices on the Intel Xeon Phi architecture.

- stream the l nonzeroes corresponding to this block into a vector register V, stream the row-wise offset array I of size p, and stream the column-wise offset array J of size q;
- gather the (possibly non-disjoint) elements from the output vector y using I into a vector register Y, and gather elements from x using J into a vector register X;
- do a vectorised $Y = Y + V \cdot X$, scatter Y according to I.

When $p > 1$ and $q > 1$, this multiplication scheme is, like Blocked CRS, a 2D vectorisation method. Otherwise, the resulting scheme is 1D and is equivalent to (sliced) ELLPACK (for $q = 1$) or SpMVs using segmented reductions (for $p = 1$). Unlike Blocked CRS and ELLPACK, the indices in I and J need not be contiguous thanks to the use of the gather and scatter primitives. Segmented scans do allow for non-contiguous J but require processing of an additional bitmasking array, which the proposed vectorised scheme does not require. The proposed vectorisation strategy hence generalises all three prior sparse matrix data structures.

The newly proposed method requires fill-in to ensure that the input nonzeroes in the given order fit precisely into successive $p \times q$ blocks; this was also required for Blocked CRS and ELLPACK, but was not required for methods based on segmented reductions. There, instead of filling in explicit zeroes, a bitmasking array in effect notes which nonzeroes are contained in a single block. Such bitmasking, however, was observed to be effective only on specific well-structured matrices; the proposed strategy avoids this issue entirely.

Results. The proposed 2D vectorised method for SpMV multiplication is highly effective on the Intel Xeon Phi. This relatively new architecture relies heavily on the use of many threads as well as the use of wide vector registers, supporting 240 hardware threads ($p = 240$) and storing 8 double-precision floating point values in a single register ($l = 8$). Figure 1 shows of the attained performance of the new SpMV strategy for different matrices and different block sizes, while the 1x1 category corresponds to same parallel SpMV method but without using vectorisation ($l = 1$).

The use of the proposed data structure is compared to the state-of-the-art performance on other architectures as well. Its use with other sparse matrix operations, such as the sparse matrix powers kernel, is also discussed.

REFERENCES

[YR14] A. N. Yzelman and D. Roose, High-level strategies for parallel shared-memory sparse matrix–vector multiplication, IEEE Transactions on Parallel and Distributed Systems, volume 25 (1); pp. 116–125, 2014.

FLANDERS EXASCIENCE LAB, DEPARTMENT OF COMPUTER SCIENCE, KU LEUVEN, CELESTIJNENLAAN 200A - BUS 2402, B-3001 HEVERLEE, BELGIUM., EMAIL: ALBERT-JAN.YZELMAN@CS.KULEUVEN.BE

Efficient Sparse Matrix-Matrix Multiplication on Multicore Architectures*

Adam Lugowski† John R. Gilbert‡

Abstract

We describe a new parallel sparse matrix-matrix multiplication algorithm in shared memory using a quadtree decomposition. Our implementation is nearly as fast as the best sequential method on one core, and scales quite well to multiple cores.

1 Introduction

Sparse matrix-matrix multiplication (or *SpGEMM*) is a key primitive in some graph algorithms (using various semirings) [5] and numeric problems such as algebraic multigrid [9]. Multicore shared memory systems can solve very large problems [10], or can be part of a hybrid shared/distributed memory high-performance architecture.

Two-dimensional decompositions are broadly used in state-of-the-art methods for both dense [11] and sparse [1] [2] matrices. Quadtree matrix decompositions have a long history [8].

We propose a new sparse matrix data structure and the first highly-parallel sparse matrix-matrix multiplication algorithm designed specifically for shared memory.

2 Quadtree Representation

Our basic data structure is a 2D quadtree matrix decomposition. Unlike previous work that continues the quadtree until elements become leaves, we instead only divide a block if its nonzero count is above a threshold. Elements are stored in column-sorted triples form inside leaf blocks. Quadtree subdivisions occur on powers of 2; hence, position in the quadtree implies the high-order bits of row and column indices. This saves memory in the triples. We do not assume a balanced quadtree.

3 Pair-List Matrix Multiplication Algorithm

The algorithm consists of two phases, a *symbolic phase* that generates an execution strategy, and a *computational phase* that carries out that strategy. Each phase is itself a set of parallel tasks. Our algorithm does not schedule these tasks to threads; rather we use a standard scheduling framework such as TBB, Cilk, or OpenMP.

3.1 Symbolic Phase We wish to divide computation of $C = A \times B$ into efficiently composed tasks with sufficient parallelism. The quadtree structure gives a natural decomposition into tasks, but the resulting tree of sparse matrix additions is inefficient. Instead we form a list of additions for every result block, and build the additions into the multiply step. We let C_{own} represent a leaf block in C, and *pairs* the list of pairs of leaf blocks from A and B whose block inner product is C_{own}.

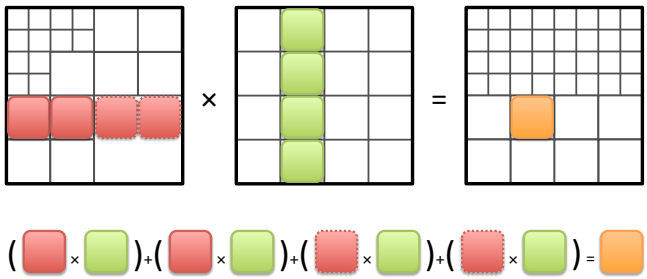


Figure 1: Computation of a result block using a list of pairwise block multiplications.

$$C_{own} = \sum_{i=1}^{|pairs|} A_i \times B_i \tag{3.1}$$

The symbolic phase recursively determines all the C_{own} and corresponding *pairs*.

We begin with $C_{own} \leftarrow C$, and $pairs \leftarrow (A, B)$. If *pairs* only consists of leaf blocks, spawn a compute task with C_{own} and *pairs*. If *pairs* includes both divided blocks and leaf blocks, we temporarily divide the leaves until all blocks in *pairs* are equally divided. This temporary division lets each computational task operate on equal-sized blocks; it persists only until the end of the SpGEMM.

Once the blocks in *pairs* are divided, we divide C_{own} into four children with one quadrant each and recurse, rephrasing divided $C = A \times B$ using (3.1):

$$\begin{aligned} C_1 &= [(A_1, B_1), (A_2, B_3)] \\ C_2 &= [(A_1, B_2), (A_2, B_4)] \\ C_3 &= [(A_3, B_1), (A_4, B_3)] \\ C_4 &= [(A_3, B_2), (A_4, B_4)] \end{aligned} \tag{3.2}$$

For every pair in *pairs*, insert two pairs into each child's *pairs* according to the respective line in (3.2). Each child's *pairs* is twice as long as *pairs*, but totals only 4 sub-blocks to the parent's 8.

3.2 Computational Phase This phase consists of tasks that each compute one block inner product (3.1). Each task is lock-free because it only reads from the blocks in *pairs* and only writes to C_{own}. We extend

*Supported by Contract #618442525-57661 from Intel Corp. and Contract #8-482526701 from the DOE Office of Science.

†CS Dept., UC Santa Barbara, alugowski@cs.ucsb.edu

‡CS Dept., UC Santa Barbara, gilbert@cs.ucsb.edu

Gustavson's sequential algorithm [4] in Algorithm 1.

Our addition to Gustavson is a mechanism that combines columns j from all blocks B_i in *pairs* to present a view of the entire column j from B. We then compute the inner product of column j and all blocks A_i using a "sparse accumulator", or *SPA*. The SPA can be thought of as a dense auxiliary vector, or hash map, that efficiently accumulates sparse updates to a single column of C_{own}.

A and B are accessed differently, so we *organize* their column-sorted triples differently. For constant-time lookup of a particular column i in A, we use a hash map with a $i \rightarrow (offset_i, length_i)$ entry for each non-empty column i. A CSC-like structure is acceptable, but requires O(m) space. We iterate over B's non-empty columns, so generate a list of $(j, offset_j, length_j)$. Both organizers take $O(nnz)$ time to generate. A structure that merges all B_i organizers enables iteration over logical columns that span all B_i.

Algorithm 1 Compute Task's Multi-Leaf Multiply

Require: C_{own} and *pairs*
Ensure: Complete C_{own}
  for all (A_b, B_b) in *pairs* **do**
    organize A_b columns with hash map or CSC
    organize B_b columns into list
  end for
  merge all B organizers into *combined_B_org*
  for all (column j, $PairList_j$) in *combined_B_org* **do**
    $SPA \leftarrow \{\}$
    for all (A_b, B_b) in $PairList_j$ **do**
      for all non-null k in column j in B_b **do**
        accumulate $B_b[k, j] \times A_b[:, k]$ into SPA
      end for
    end for
    copy contents of SPA to $C_{own}[:, j]$
  end for Texte

4 Experiments

We implemented our algorithm in TBB [7] and compared it with the fastest serial and parallel codes available, on a 40-core Intel Nehalem machine. We test by squaring Kronecker product (RMAT) matrices [6] and Erdős-Rényi matrices.

Observe from Table 1 that QuadMat only has a small speed penalty on one core compared to CSparse, but gains with two or more cores.

5 Conclusion

Our algorithm has excellent performance, and has the potential to be extended in several ways. Our next steps include a triple product primitive that does not materialize the entire intermediate product at any one time, and computing $A^T \times B$ with similar complexity to $A \times B$.

Table 1: SpGEMM results on E7-8870 @ 2.40GHz - 40 cores over 4 sockets, 256 GB RAM. Note: CombBLAS is an MPI code that requires a square number of processes.

Squared Matrix		R_{16}	R_{18}	ER_{18}	ER_{20}
Each Input nnz		1.8M	7.6M	8.39M	33.6M
Output nnz		365M	2.96G	268M	1.07G
CSparse [3]	1p	14s	122s	9s	58s
CombBLAS [2]	1p	154s	1597s	64s	248s
	9p	19s	155s	8s	34s
	36p	8s	49s	3s	12s
QuadMat	1p	19s	150s	13s	111s
	2p	10s	87s	8s	66s
	9p	3s	21s	3s	18s
	36p	2s	11s	2s	9s

References

[1] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proc. 21st Symp. on Parallelism in Algorithms and Arch.*, 2009.

[2] A. Buluç and J.R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *Intl. J. High Perf. Computing Appl.*, 25(4):496–509, 2011.

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

[4] F. G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw.*, 4(3):250–269, 1978.

[5] J. Kepner and J. R. Gilbert, editors. *Graph Algorithms in the Language of Linear Algebra*. SIAM, 2011.

[6] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using Kronecker multiplication. In *Proc. 9th Principles and Practice of Knowledge Disc. in Databases*, pages 133–145, 2005.

[7] C. Pheatt. Intel threading building blocks. *J. Comput. Sci. Coll.*, 23(4):298–298, April 2008.

[8] H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, 1984.

[9] Y. Shapira. *Matrix-based Multigrid: Theory and Applications*. Springer, 2003.

[10] J. Shun and G. E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. *SIGPLAN Not.*, 48(8):135–146, February 2013.

[11] R. A. Van De Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997.

Scaling Iterative Solvers by Avoiding Latency Overhead of Parallel Sparse Matrix Vector Multiplication

R. Oguz Selvitopi[a], Mustafa Ozdal[b], Cevdet Aykanat[a]

[a] *Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey*
[b] *Strategic CAD Labs of Intel Corporation, Hillsboro, OR, 97124, US*

Parallel iterative solvers are the most widely used methods for solving sparse linear systems of equations on parallel architectures. There are two basic types of kernels that are repeatedly computed in these solvers: Sparse-matrix vector multiply (SpMV) and linear vector operations. Since linear vector operations are performed on dense vectors, they are regular in nature and are easy to parallelize. Conversely, SpMV operations generally require specific methods and techniques for efficient parallelization due to irregular sparsity pattern of the coefficient matrix. In literature, several partitioning models and methods are proposed for efficient parallel computation of SpMV operations.

In a single iteration of the solver, SpMV operations cause irregular point-to-point (P2P) communication and inner product computations cause regular collective communication. The partitioning techniques proposed in the literature generally aim at reducing communication volume incurred in P2P communications, which loosely relates to latency overhead incurred in parallel SpMV operations. On current large-scale systems, the message latency overhead is at least as important as the message volume overhead, especially in the case of strong scaling in which average message sizes decrease with increasing number of processors. Our preliminary experiments on two large-scale systems (an IBM BlueGene/Q and a Cray XE6) demonstrate that the startup time is as high as transmitting four-to-eight kilobytes of data.

On the contrary to the studies that aim at hiding latency of collective communication operations [1] (by using nonblocking collective primitives and overlapping with computation), we propose a methodology to directly avoid *all* latency overhead associated with P2P messages of SpMV operations. Our methods rely on the observation that in most of the Krylov subspace methods, each SpMV computation is followed by an inner product computation which involves output vector of the SpMV. This introduces a write/read dependency on this vector between SpMV and inner product computational phases.

In [2], we propose a novel computational rearrangement method to resolve the above-mentioned computational dependency between these two computational phases. By doing so, we remove the communication dependencies between these two phases and enable P2P communications of SpMV and collective communications of inner products to be performed in a single communication phase. The computational rearrangement reduces the number of synchronization points for each SpMV and inner product computation pair by one, that is, the proposed scheme requires a single synchronization point in a typical CG implementation. Then, we realize this opportunity to propose a communication rearrangement method to avoid all latency overhead of P2P messages of SpMV operations. This is achieved by embedding P2P communications into collective communication operations. The proposed embedding scheme reduces both the average and the *maximum* number

of messages handled by a single processor to lg K in an iterative solver with K processors, regardless of the coefficient matrix being solved.

The downside, however, is that the embedding scheme causes extra communication volume due to forwarding of certain vector elements. To address this increase in message volume, two iterative-improvement-based algorithms are proposed. The basic idea of these heuristics is to place the processors that exchange high volume of data close to each other so that the store-and-forward scheme required by the embedding method causes less forwarding overhead. This is a preprocessing step as the partitioning itself and the running time of the described faster heuristic is lower than the partitioning time up to 2048 processors.

The mentioned methods and techniques are validated on Conjugate Gradient method. The 1D row-parallel algorithm is used for SpMV. We tested our methods on two large-scale high performance computing systems Cray XE6 and IBM BlueGene/Q up to 2048 processors with 16 test matrices from University of Florida Sparse Matrix Collection. With using proposed computational and communication rearrangement, we show that we obtain superior scalability performance on both architectures. Our findings indicate that the crucial factor to scale an iterative solver is to keep the message latency overhead low, which dominate the message volume overhead at high processor counts.

References

[1] P. Ghysels, W. Vanroose, Hiding global synchronization latency in the preconditioned conjugate gradient algorithm, Parallel Computing (0) (2013) –. doi:http://dx.doi.org/10.1016/j.parco.2013.06.001.
URL http://www.sciencedirect.com/science/article/pii/S0167819113000719

[2] O. Selvitopi, M. Ozdal, C. Aykanat, A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies, Parallel and Distributed Systems, IEEE Transactions on PP (99) (2014) 1–1. doi:http://dx.doi.org/10.1109/TPDS.2014.2311804.
URL http://www.computer.org/csdl/trans/td/preprint/06766662-abs.html

Scalable Large Scale Graph Analytics

Y. Ineichen, C. Bekas, and A. Curioni
IBM Research – Zurich
Computational Sciences
Säumerstrasse 4, 8803 Rüschlikon - Switzerland
{yin,bek,cur}@zurich.ibm.com

In recent years, graph analytics has become one of the most important and ubiquitous tools for a wide variety of research areas and applications. Indeed, modern applications such as ad hoc wireless telecommunication networks, or social networks, have dramatically increased the number of nodes of the involved graphs, which now routinely range in the tens of millions and out-reaching to the billions in notable cases.

We developed novel near linear ($\mathcal{O}(N)$) methods for sparse graphs with N nodes estimating the most important nodes in a graph, the subgraph centralities, and spectrograms, that is the density of eigenvalues of the adjacency matrix of the graph in a certain unit of space.

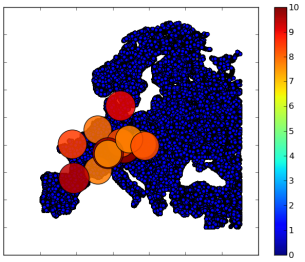


Figure 1: Most central intersections in the European street network.

The method to compute subgraph centralities employs stochastic diagonal estimation [2] and Krylov subspace techniques to drastically reduce the complexity which, using standard methods, is typically $\mathcal{O}(N^3)$. With this technique we can approximate the centralities in a fast, highly scalable and accurate fashion, and thereby open the way for centrality based big data graph analytics that would have been nearly impossible with standard techniques. Subgraph centralities provide a wealth of information in many situations. For example, the subgraph centralities can be used to identify possible bottlenecks in huge networks. Figure 1 visualizes the most central nodes in the European street network[1] (part of the 10th DIMACS challenge [1]) with 51 million nodes. Our efficient parallel implementation only required 800 seconds to compute the centralities on 16 threads.

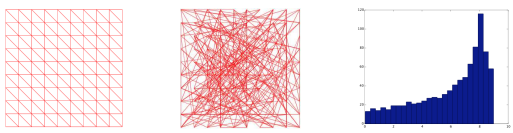


Figure 2: It is almost impossible to determine if the graphs (left and middle column) are the same for a human observer.

In the age of big data it becomes increasingly difficult to compare and visualize graphs. The spectrogram helps to visually interpret and compare data, for example the two leftmost

[1] https://www.cise.ufl.edu/research/sparse/matrices/DIMACS10/europe_osm.html

graphs in Fig. 2. For humans it is impossible to compare the two graphs. In fact, both graphs are exactly the same and one of the graphs can be transformed into the other by a set of simple permutations. The spectrogram shown on the right in Fig. 2 captures the essential characteristic of the graphs immediately and graphically. It transforms complex graphs into a 1-dimensional vector - a simple picture that fosters convenient interpretation.

Spectrograms are powerful in capturing the essential structure of graphs and provide a natural and human readable (low dimensional) representation for comparison. How about comparing graphs that are almost similar (see Figure 3)? Of course, this is a massive dimensionality reduction, however at the same time the shape of the spectrogram yields a tremendous wealth of information.

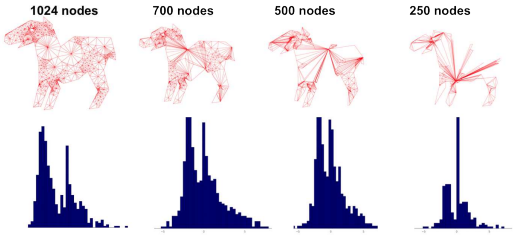


Figure 3: Almost similar graphs and spectrograms

Solving the underlaying eigenvalue problem is getting much harder to master in the era of big data. The cubic complexity of dense methods and the limitation of iterative techniques to look deep into the interior of the spectrum at an acceptable cost, call for a new approach. Our approach starts by estimating $\lambda_{\min}$ and $\lambda_{\max}$ of the adjacency matrix by a few steps of Lanczos, in order to shift and scale the adjacency matrix to have its spectrum in the interval $[-1, 1]$. Next, we divide the range $[-1, 1]$ in the number of requested bins μ, known as inflection points. Subsequently, we estimate the number of eigenvalues below μ, using trace estimation techniques (similar to [3, 2]) of the Fermi-Dirac distribution function, to compute the spectrogram.

In order to tackle arising big data challenges an efficient utilization of available HPC resources is key. Both developed methods exhibit an efficient parallelization on multiple hierarchical levels. For example, computing the spectrogram can be parallelized on three levels: bins and matrix-vector products can be computed independently, and each matrix-vector product can be computed in parallel. The combination of a highly scalable implementation and algorithmic improvements enable us to tackle big data analytics problems that are nearly impossible to solve with standard techniques. A broad spectrum of applications in industrial and societal challenges can profit from fast graph analytics.

Acknowledgments Yves Ineichen and Costas Bekas acknowledge the support from XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323.

References

[1] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Soc., 2013.

[2] C. Bekas, E. Kokiopoulou, and Y. Saad. An estimator for the diagonal of a matrix. *Appl. Numer. Math.*, 57(11-12):1214–1229, Nov. 2007.

[3] M. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

NETWORKIT: AN INTERACTIVE TOOL FOR HIGH-PERFORMANCE NETWORK ANALYSIS

Christian L. Staudt, Aleksejs Sazonovs, Henning Meyerhenke

Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT)

Summary

We introduce NetworKit, an open-source package for high-performance analysis of large complex networks. Complex networks are equally attractive and challenging targets for data mining, and novel algorithmic solutions as well as parallelism are required to handle data sets containing billions of connections [4, 5]. Our goal is to package results of our algorithm engineering efforts and put them into the hands of domain experts.

The package is a hybrid combining the performance of kernels – written in C++ and parallelized with OpenMP – with a convenient interactive interface written in Python. The package supports general multicore platforms and scales from notebooks to workstations to compute servers. In comparison with related software for network analysis, we propose NetworKit as the package which satisfies all of three important criteria: High performance enabled by parallelism, interactive workflows and integration into an ecosystem of tested tools for data analysis and scientific computation. The feature set includes standard network analytics kernels such as connected components, clustering coefficients, community detection, core decomposition, assortativity and centrality. Applying these to massive networks is enabled by efficient algorithm design, parallelism and approximation. Furthermore, the package comes with a collection of graph generators and has basic support for visualization and dynamic networks. With the current release, we aim to present and open up the project to a community of both algorithm engineers and domain experts.

Features

NetworKit has a growing feature set and is built for extensibility. Current features include:

Community detection is the task of identifying groups of nodes which are significantly more densely connected among each other than to the rest of the network. NetworKit includes state-of-the-art heuristics with efficient parallel implementations for partitioning the network into natural modules [7].

Clustering coefficients quantify the tendency of relations in a network to become transitive by looking at the frequency of closed triangles. NetworKit supports both exact calculation and approximation.

Degree distribution and assortativity play an important role in characterizing a network: Complex networks tend to show a heavy tailed degree distribution which follow a power-law with a characteristic exponent [1]. Degree assortativity is the correlation of degrees for connected nodes. NetworKit makes it easy to estimate both.

Components and cores are related concepts for subdividing a network: All nodes in a connected component are reachable from each other. k-cores/k-shells result from successively peeling away nodes of degree k.

Centrality refers to the relative importance of a node within a network. Different ideas of importance are expressed by betweenness, PageRank and eigenvector centrality. Betweenness is approximated with a bounded error to be applicable to large networks.

Standard graph algorithms such as finding independent sets, computing approximate maximum weight matchings, breadth-first and depth-first search or finding shortest paths.

Generative models aim to explain how networks form and evolve specific structural features. NetworKit has efficient generators for basic Erdős-Rényi random graphs, the Barabasi-Albert and Dorogovtsev-Mendes models (which produce power law degree distributions), the Chung-Lu and Havel-Hakimi model (which replicate given degree distributions, the former in expectation, the latter only realizable ones).

Visualization functionality which enables the user to draw smaller networks to the IPython Notebook or files.

Design Goals

NetworKit is designed to stand out in three areas:

Performance Algorithms and data structures are selected and implemented with high performance and parallelism in mind. Some implementations are among the fastest in published research. For example, community detection in a 3 billion edge web graph can be performed on a machine with 16 physical cores and 256 GB of RAM in a matter of minutes.

Interface Networks are as diverse as the series of questions we might ask of them - for example, what is the largest connected component, what are the most central nodes in it and how do they connect to each other? A practical tool for network analysis should therefore avoid restricting the user to fixed and predefined tasks, as most static command line interfaces do. Rather, the aim must be to create convenient and freely combinable functions. In this respect we take inspiration from software like R, MATLAB and Mathematica, as well as a variety of Python packages. An interactive shell, which the Python language provides, meets these requirements. While NetworKit works with the standard Python 3 interpreter, combining it with the IPython Notebook allows us to integrate it into a fully fledged computing environment for scientific workflows [6]. It is also straightforward to set up and control a remote server for heavy computations.

Integration As a Python module, NetworKit enables seamless integration with Python libraries for scientific computing and data analysis, e. g. pandas for data frame processing and analytics, matplotlib for plotting, networkx for additional network analysis tasks, or numpy and scipy for advanced numeric and scientific computation. Furthermore, NetworKit aims to support a variety of input/output formats, for example export to the graphical network analysis software Gephi [2].

Implementation

Core data structures and algorithms of NetworKit are implemented in C++ using the C++11 standard, which allows the use of object-oriented and functional programming concepts without sacrificing performance. The graph data structure provides parallel iterators over node and edge sets using different load balancing schemes. Shared-memory parallelized is realized with OpenMP. Classes are then exposed to Python via the Cython toolchain [3]: Wrapper classes are converted to C++ code via the Cython compiler, then compiled and linked with the core into a native Python extension module. Additional functionality and a convenient interface is implemented in pure Python, yielding the final Python module.

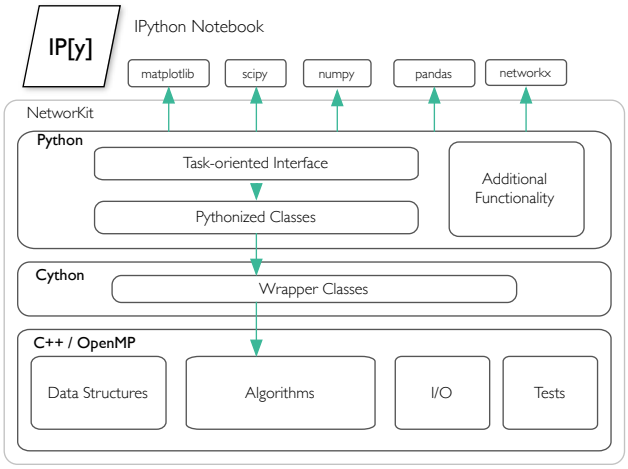


Figure 1: NetworKit architecture

Open Source

NetworKit is published[1] under the permissive MIT License to encourage review, reuse and extension by the community. We invite algorithm engineers and potential users from various research domains to benefit from and contribute to the development effort. ¿¿¿¿¿¿¿ other

References

[1] J. Alstott, E. Bullmore, and D. Plenz. powerlaw: a python package for analysis of heavy-tailed distributions. *PLOS ONE*, 9(1):e85777, 2014.

[2] M. Bastian, S. Heymann, and M. Jacomy. Gephi: an open source software for exploring and manipulating networks. In *ICWSM*, pages 361–362, 2009.

[3] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[4] U. Brandes and T. Erlebach. *Network analysis: methodological foundations*, volume 3418. Springer, 2005.

[5] M. Newman. *Networks: an introduction*. Oxford University Press, 2010.

[6] F. Perez, B. E. Granger, and C. Obispo. An open source framework for interactive, collaborative and reproducible scientific computing and education, 2013.

[7] C. L. Staudt and H. Meyerhenke. Engineering high-performance community detection heuristics for massive graphs. *arXiv preprint arXiv:1304.4453*.

[1] http://www.network-analysis.info

Detecting Anomalies in Very Large Graphs*

Michael M. Wolf† and Benjamin A. Miller

MIT Lincoln Laboratory

Numerous applications focus on the analysis of entities and the connections between them, and such data are naturally represented as graphs. In particular, the detection of a small subset of vertices with anomalous coordinated connectivity is of broad interest, for problems such as detecting strange traffic in a computer network or unknown communities in a social network. These problems become more difficult as the background graph grows larger and noisier and the coordination patterns become more subtle. In this talk, we present a statistical framework addressing this cross-mission challenge and the computational challenges involved when the data sets are very large.

The statistical framework has been developed as part of an effort called Signal Processing for Graphs (SPG), where signal processing concepts are applied to graph data in order to find these anomalies [3]. The SPG framework determines statistically whether, for the observed data, an anomalous subgraph is detected (rejection of the null hypothesis that there is no anomalous subgraph) or not (acceptance of the null hypothesis). Detection algorithms in the SPG framework are based on the concept of graph residuals, formed by subtracting the expected graph data from the observed graph data. The detection framework is designed to detect significant residuals concentrated on a small subset of vertices. The SPG processing chain has several stages including temporal integration of graph data, construction of the expected topology graph, dimensionality reduction, anomalous subgraph detection, and identification of the subgraph's vertices. Dimensionality reduction is a particularly important step, being the most computationally complex, and will be the primary focus of the presentation.

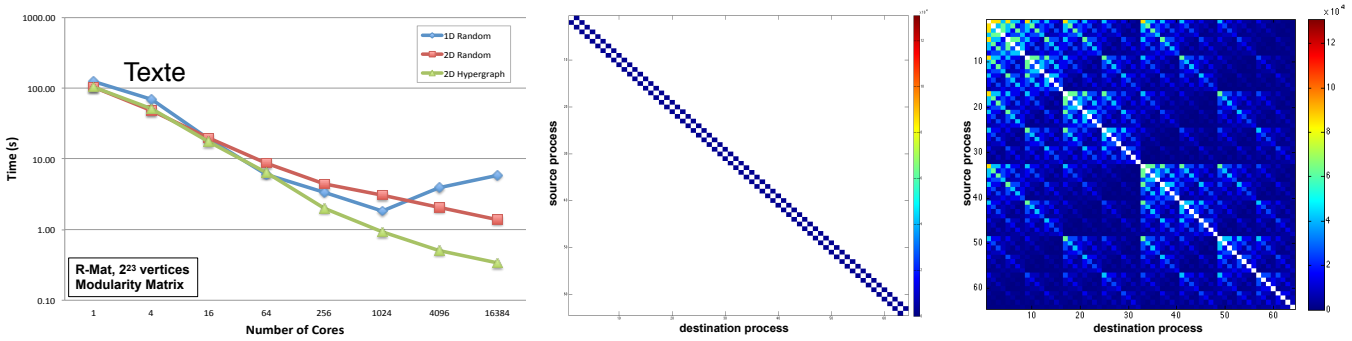
Dimensionality reduction is frequently done by decomposing the residual matrix through eigendecomposition (although SVD would work as well). While relatively strong signals can be detected with only one eigenvector, more powerful detection methods needed to detect more subtle anomalies may require hundreds of eigenvectors. Thus, our initial efforts improve the performance of this step have focused on speeding up the solution of the eigensystem, $Bx_i = \lambda_i x_i$, where $B = A - E[A]$ is the residual matrix. For the purpose of this work, we have focused on the modularity matrix [4], where the expected value is a rank-1 approximation to the observed data, $E[A] = kk^T/(2M)$, where M is the number of edges in the graph and k is the degree vector (assuming undirected edges for simplicity). We use the Anasazi eigensolver [2] to solve our eigensystem, allowing us to find anomalies in very large graphs in a reasonable amount of time. In particular, we use the block Krylov-Schur method to find the eigenvectors corresponding to the eigenvalues with the largest real components (which correspond to the largest residuals). The presentation will demonstrate the use of Anasazi to find eigenpairs in graphs with over four billion vertices.

Figure 1a shows the run time to find the first eigenvalue of a 2^{23}-vertex R-Mat matrix (a=0.5, b=0.125, c=0.125, with an average of 8 nonzeros per row) as the number of cores increases. When a simple one-dimensional distribution (1D, blue curve) is used, the scalability is limited and the run time actually increases for more than 1024 cores. 1D distributions are the de facto standard and tend to work well for traditional computational science and engineering problem. However, for matrices derived from power-law graphs, 1D partitioning tends to result in all-to-all communication in the sparse matrix-vector multiplication (SpMV) operation that dominates the eigensolver run time. Figure 1 shows the communication patterns for 1D distributions of a more traditional finite difference matrix (1b) and the R-Mat matrix (1c). In these illustrations, the color corresponds to amount of data communicated between a pair of processes (white represents no communication, blue represents little communication, and red represents much communication, with the maximum value (red) being set to be the number of rows divided by the number of cores). The 1D distribution works well for the finite difference matrix, with each process communicating with at most two other processes. However, for the R-Mat matrix, each process has to communicate with every other process.

With 1D distributions, solving our eigendecomposition problem for very large power-law graphs is infeasible with our runtime requirements. However, there has been recent work on using two-dimensional (2D) distributions that bound the number of messages to be communicated in the resulting SpMV operation. In Figure 1a, we show the improved results for two of these 2D distributions: one based on a random partitioning of rows/columns with an imposed 2D block Cartesian structure (2DR, red line, [5]) and one that uses hypergraph partitioning (with the Zoltan 1D hypergraph partitioner) instead of random partitioning to improve the communication volume (2DH, green line, [1]). Using these 2D methods, we are able to get more reasonable performance scaling and find eigenvectors for large graphs, with the 2DH method performing particularly well. In this talk, we present these results and further analysis of how these 2D methods can be used in practice.

*This work is sponsored by the Intelligence Advanced Research Projects Activity (IARPA) under Air Force Contract FA8721-05-C-0002. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA or the U.S. Government.

†Currently at Sandia National Laboratories



(a) Time to find one eigenpair. (b) 9-point 2D finite difference. (c) R-Mat matrix.

Figure 1: Strong scaling results for R-Mat matrix (a). Communication patterns (b,c) for 1D distributions over 64 cores. Row represents source process and column represents the destination process. Color corresponds to amount of data communicated (with maximum value being number of rows divided by number of cores). Jobs run on National Energy Research Scientific Computing Center (NERSC) machine Hopper.

The challenge with 2DH is its relative complexity in computing the distribution. For an 8-million-row R-Mat problem, approximately 40,000 SpMV operations are required to amortize the additional cost of calculating the 2DH distribution rather than 2DR. Since we typically need at most a few thousand SpMV operations in our eigensolver, the 2DH distribution must be effective for multiple observed graphs to be useful. We explored this effectiveness using a simple dynamic graph model in which we partition an initial graph and use our R-Mat generator to add more edges (using the defined distribution) to generate a sequence of additional graphs. Figure 2 shows the runtime (normalized by the number of edges) of the SpMV operations for the 2DR and 2DH distributions applied to several graphs that evolve from an initial graph containing 30% of the edges in the final graph. Although the 2DH distribution loses some of its advantage over the 2DR distribution, it still has a significant advantage at the final graph. Thus, it may be possible to effectively amortize the expensive 2DH distribution over several graphs and use this distribution to our computational advantage. We plan to explore this further for additional dynamic graph models.

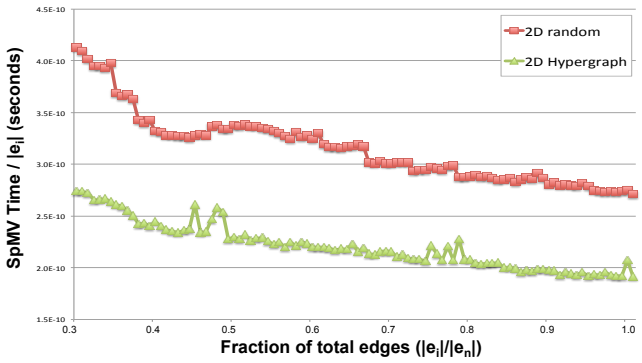


Figure 2: SpMV time in seconds (normalized by number of nonzeros) as graph evolves.

References

[1] E. G. Boman, K. D. Devine, and S. Rajamanickam. Scalable matrix computations on large scale-free graphs using 2D graph partitioning. In *Proc. Supercomputing*, pages 50:1–50:12, New York, NY, USA, 2013. ACM.

[2] C. G. Baker et al. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Softw.*, 36(3):13(1–23), July 2009.

[3] B. A. Miller, N. T. Bliss, P. J. Wolfe, and M. S. Beard. Detection theory for graphs. *Lincoln Laboratory Journal*, 20(1):10–30, 2013.

[4] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3), 2006.

[5] A. Yoo, A. H. Baker, R. Pearce, and V. E. Henson. A scalable eigensolver for large scale-free graphs using 2D graph partitioning. In *Proc. Supercomputing*, pages 63:1–63:11, New York, NY, USA, 2011. ACM.

Finding High Betweenness Centrality Vertices in Large Networks

Vladimir Ufimtsev and Sanjukta Bhowmick

Department of Computer Science, University of Nebraska at Omaha

Introduction. Betweenness centrality (BC) is a widely applied network measure for identifying important vertices in complex networks. BC measures the importance of a vertex with respect to the flow of information in a network, based on the number of shortest paths that pass through that vertex. Specifically, the BC of vertex v is defined as [3]: $BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$, where σ_{st} is the total number of shortest paths in G between nodes s and t, and $\sigma_{st}(v)$ is the total number of shortest paths in G between s and t that pass through v. Most algorithms for computing BC have to compute the values for all the vertices in the network i.e. it is not possible to compute the BC just for a specific vertex without having to compute BC for the whole network. For example, the popular Brandes method [1] for obtaining BC, cumulatively computes the values for every vertex in the network. Although the algorithm has polynomial complexity, the execution time $O(V * E)$ is still prohibitive for large-scale networks. However, in practice only the vertices with the highest BC values are required. Here we show how we can use group testing to obtain the d-highest BC vertices, in shorter time than computing the BC of all vertices (and then sorting to find the highest ones).

Group Testing to Identify High BC Vertices

We focus on identifying only the top-highest BC vertices in the network, and it is the identity not the actual BC value of the vertex that is important. We use our proposed algorithm (for calculating the BC of a specified vertex) in a group testing based technique to identify the vertices in the network that have the highest BC [7]. The central idea of group testing is that if there is a small percentage of defective units in a large population of units, it is more efficient (requires less tests) to test the units in carefully selected groups, using for example principles of superimposed code theory, rather than testing each unit separately. Group testing has a vast amount of applications including pattern matching and DNA library screening [4, 5]. In our application of group testing, the defective units correspond to vertices with high BC. We use a group testing design based on a Latin square to determine the number of groups (tests) and what vertices are part of each group. According to this design, for each test the specified vertices are grouped into a single supervertex and the betweenness centrality of that specified supervertex is calculated using our single vertex BC algorithm. If the BC value of the supervertex (group) is high (exceeds a threshold) then the result is designated as positive (1) otherwise it is negative (0). Upon completion of all the tests, the vertices that have the highest BC rank are identified. Theoretically this method guarantees that we find at least two highly ranked vertices in $3\lceil\sqrt{n}\rceil$ tests in a network on n vertices.

Preliminary Results

Using our group testing algorithm we performed experiments over a set of ten networks collected from the DIMACS Implementation Challenge Set [2] and the Stanford Network Analysis Project [6].

To evaluate the efficiency and accuracy of the method we analyze how many of the nodes identified by group testing are actually high ranking. Using the Brandes algorithm [1] we obtain the full ranking for each network and see what rank the nodes identified by group testing have. The group testing design we are using (based on Latin squares) is successful if it identifies at least the top 2 vertices. The results are given in Table 1 (reproduced from [8]). Out of the ten networks, group testing was successful on six networks (top six rows of the table), and found low ranked (below rank 10) vertices for the other four (the last four rows of the table).

On further study we observed that the networks for which group testing failed to find the high ranked vertices were the ones that were most sensitive to small perturbations in the network structure.

Therefore group testing can be also used as a method to classify networks that are robust to noise from networks that are more sensitive.

Table 1. Finding High BC Vertices Using Group Testing on Real-World Networks. The best threshold and the vertices obtained using that threshold are given. The vertices are represented by their rank, as per their BC values obtained using the Brandes method.

Name	**Vertices**	**Edges**	# of Tests	Threshold	**High BC Vertices**
Karate	34	156	18	55%	**1st, 2nd**
Chesapeake	39	340	21	30%	**1st, 2nd**
AS20000102	6474	13233	243	12%	**1st, 2nd, 3rd**
AS20000101	3570	7391	180	16%	**1st, 2nd**
Caida	16301	65910	384	21%	**1st, 2nd, 3rd**
C. Elegans	453	4050	66	35%	**1st, 2nd, 4th 10th** + 3 low ranked
Les Mis.	77	508	27	45%	**1st, 10th**, +3 low ranked
GrQc	5242	28980	219	80.3%	20 low ranked
HepTh	9877	51971	300	76%	6 low ranked
Power Grid	4941	13188	213	84%	6 low ranked

Each group of vertices can be formed and the corresponding tests can be executed independently. Therefore group testing is perfectly parallelizable. In each test we are only required to know the BC of the supervertex. However, most of the current BC computing algorithms focus on cumulatively finding the BC of all the vertices, and cannot identify the BC of only a specific vertex.

Therefore, in order to efficiently apply our group testing algorithm, we have developed an algorithm that computes the BC of one vertex only. The efficiency of the algorithm is related to the size of and the number of chordless cycles in the graph. If the graph is chordal, i.e. the largest chordless cycle is of length three, then we can compute the BC of a designated vertex in time proportional to execute one breadth first search. For larger chordless cycles, in the worst case, we have to execute a BFS for each cycle. Therefore, if the number of chordless cycles in the graph is q, then computing the BC of a vertex would take time $O(q * V)$.

In this talk we will present our results on group testing over a wider set of networks, we will demonstrate how group testing is an effective method for finding the highest BC vertices in robust networks, and how we can identify sensitive networks using this method. Finally we will present our algorithm for finding the BC of a single vertex, along with the scalability results for group testing.

References

1. U. Brandes , Faster Algorithm for Betweenness Centrality *J. Math. Sociol.*, 25, 163 (2001)
2. DIMACS 10th Implementation Challenge http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml (2011)
3. L.C. Freeman, A Set of Measures of Centrality Based on Betweenness, *Sociometry*, 40, 35 (1977)
4. A.J. Macula, L.J. Popyack , A group testing method for finding patterns in data, *Discrete Appl. Math.* 144, no. 1-2, 149–157, (2004)
5. A.J. Macula, Probabilistic nonadaptive group testing in the presence of errors and DNA library screening, *Combinatorics and Biology* (Los Alamos, NM, 1998). Ann. Comb. 3, no. 1, 61–69, (1999)
6. Stanford Network Analysis Project (SNAP) http://snap.stanford.edu/index.html
7. V. V. Ufimtsev, A Scalable Group Testing Based Algorithm for Finding d-highest Betweenness Centrality Vertices in Large Scale Networks,Poster, *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011

8. V. V. Ufimtsev, S. Bhowmick, Application of Group Testing in Identifying High Betweenness Centrality Vertices in Complex Networks *KDD Workshop on Mining and Learning with Graphs*, (2013)

Partitioning RGG's Into Disjoint (1 - ε) Dominant Bipartite Subgraphs

Zizhen Chen* David W. Matula*
*Computer Science and Engineering, Lyle School of Engineering
Southern Methodist Univeristy
Dallas, TX, 75205
E-mail:{zizhenc, matula}@smu.edu

Abstract

We show that a sufficiently large Random Geometric Graph $G(n, r)$ can be efficiently partitioned into disjoint connected bipartite subgraphs such that a sequence $B_1, B_2, \ldots, B_k$ of these subgraphs comprising over 85% of the vertices of $G(n, r)$ have vertex sets that are either dominant sets or (1 - ε) dominating for small ε. These subgraphs are applicable to the problem of backbone partitioning in wireless sensor networks (WSN's) where each backbone is desired to be connected, dominating, and amenable to efficient routing. Bipartite subgraphs of an RGG are provably planar, so deadlock free routing is readily available for these backbone subgraphs. We first employ smallest-last coloring and show that the initial k color sets sufficient to include about 50% of the vertices of $G(n, r)$ are about the same size. We then employ an adaptive "relay coloring" of the remaining vertices to extract k more independent sets matched with the initial sets as paired "relay sets" to achieve our bipartite subgraph partition.

We provide results from extensive tests for various sizes of RGG's and also for random geometric graphs with vertices on the sphere. For the spherical case we obtain that the average face size in the bipartite subgraphs is generally between five and six, which is a further desirable property for routing when these subgraphs are considered as backbones for WSN's on the surface of the earth.

Keywords: Random Geometric Graphs, Wireless Sensor Networks, Smallest-last coloring, Backbone selection.

1. INTRODUCTION

Let a *random geometric graph* (RGG) denote a graph $G(n, r)$ with vertex set formed by choosing n points in a uniform random manner on the unit square, and introducing an edge between every vertex pair whose Euclidian distance is less than r. Our problem is to partition the majority of vertices into k disjoint sets $\{V_1, V_2, \ldots V_k\}$ whose induced subgraphs $<V_1>, <V_2>, \ldots, <V_k>$ are connected bipartite subgraphs with each part an independent set that dominates all or nearly all n vertices of $G(n, r)$. We desire to create such a partition efficiently so as to be applicable for applicable with linear time scalability for RGG's with from one thousand to several million vertices and average degrees up to several hundred. Regarding uniformity, the partition should yield subgraphs of reasonably similar size and structure. Regarding the total partition size we seek that the bipartite subgraphs collectively include a large majority of the n vertices, e.g. $\frac{\sum_i^k |V_i|}{n} \gg \frac{1}{2}$.

This problem is motivated by the extensive research on wireless sensor networks (WSN's) which are typically modeled by RGG's [1], [2], [3], [4], [5], [6].

The partition $V_1|V_2|\ldots|V_k$ with S the residual "surplus" vertices of the RGG, allows that each bipartite subgraph $<Vi>$ can serve as a backbone for monitoring essentially the whole region and connectivity allows for messages to be routed through each backbone. To preserve sensor lifetime the monitoring function activity may be rotated through the k backbones. The property that each backbone has two disjoint sets each dominating (1 - ε) vertices of the graph for very small ε (e.g. ε<0.01) gives high overall monitoring effectiveness to the resulting backbone system.

Previous research investigating fully dominating set partitions has focused on the minimum degree $\delta(G(n, r))$ and attempted to find up to $\delta + 1$ suitable backbones. The minimum degree is problematic due to the boundary effect in RGG's and we avoid this issue by shifting our focus in two ways. First we determine the number of parts k by requiring they collectively include a large majority of the sensors. In a second direction we also look at spherical random geometric graphs $G_s(n, r)$ where n vertices are placed at random on the surface of the unit sphere which supports the important application of sensor backbone formations spanning the globe. Note that spherical $G_s(n, r)$ provides that all vertices have an isomorphic probabilistic environment of adjacent neighbors without any boundary bias.

Our bipartite subgraph partitioning algorithms proceeds in two phases employing a greedy selective coloring algorithm in each phase. In the first phase smallest-last coloring of $G(n, r)$ is determined with k determined so that the first k-color sets, denoted $P_1, P_2, \ldots, P_k$, are chosen so that $\cup P_i$ includes at least $\frac{n}{2}$ vertices termed primary independent sets. In a second carefully crafted coloring phase of the remaining vertices ("relay candidates") we sequentially and in a greedy manner assign each vertex to a relay color set R_i, $1 \le i \le k$, based primarily on the vertex having the greatest number of adjacencies in P_i, also maintaining that each R_i is an independent set. Then the bipartite subgraph on $P_i \cup R_i$, is searched to determine the large component with the occasional smaller components or isolated vertices deleted into the surplus set.

2. RESULTS

Screenshots of benchmark of RGG G(6400, 0.08) on square model and sphere model:

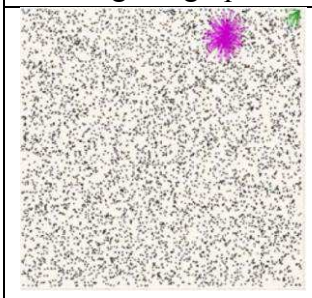
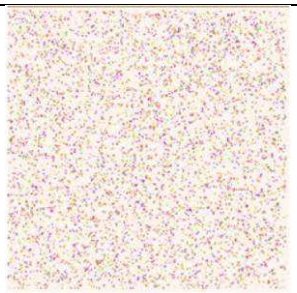
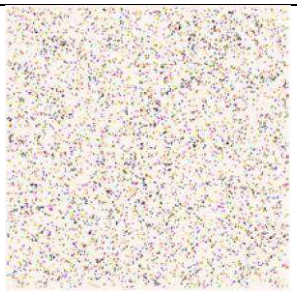
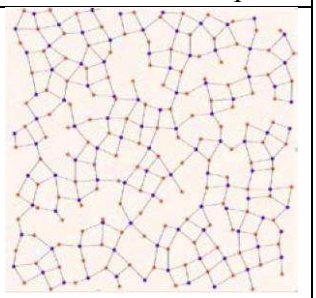
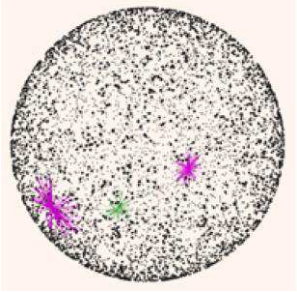
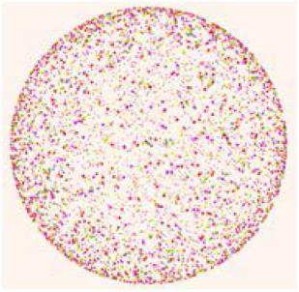
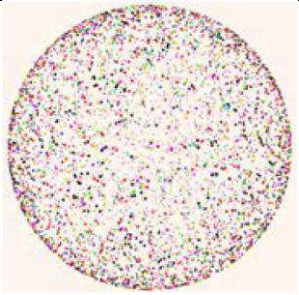
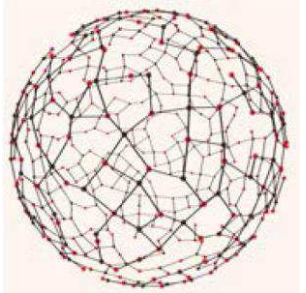
Original graph	SL-colored graph	RL-colored graph	Backbone example
			
Max. degree: 175 (Pink) Min. degree: 34 (Green)	Used colors: 64	Primary/Relay colors:24 Backbones: 24	Components: 1 Dominent: 100%
			
Max. degree: 85 (Pink) Min. degree: 35 (Green)	Used colors: 36	Primary/Relay colors:13 Backbones: 13	Components: 1 Dominent: 100%

Table 1. Simulation results on the unit square model.

Topology	G(3200, r)			G(6400, r)			G(12800, r)		
r	0.06	0.08	0.1	0.06	0.08	0.1	0.06	0.08	0.1
Min. degree	10	14	28	14	37	44	35	63	106
Max. degree	53	94	126	104	164	23	184	312	466
Avg. degree	33.94	59.80	91.83	68.63	119.24	184.82	137.57	240.87	368.06
SL-coloring colors	23	35	50	40	62	92	72	116	157
Backbones (BB)	8	13	19	15	24	35	27	45	67
Avg. BB size	344.25	213.08	145.00	376.93	230.63	155.31	409.59	244.78	163.75
Avg. BB degrees	2.38	2.58	2.72	2.63	2.81	2.88	2.88	2.97	2.99
Avg. BB components	5.88	2.46	1.42	3.33	1.25	1.20	1.48	1.16	1.01
Avg. BB faces	73.00	66.31	54.79	126.47	96.08	70.74	183.74	121.56	83.43
Avg. BB face sizes	12.04	8.60	7.39	8.27	6.82	6.37	6.48	6.02	5.91
Avg. BB dominates	99.97%	99.95%	99.99%	99.97%	99.99%	99.99%	99.99%	99.99%	99.99%

Table 2. Simulation results on the unit sphere model.

Topology	G(6400, r)			G(12800, r)			G(25600, r)		
r	0.06	0.08	0.1	0.06	0.08	0.1	0.06	0.08	0.1
Min. degree	9	20	36	25	52	90	58	114	188
Max. degree	41	60	95	69	113	175	128	215	309
Avg. degree	23.02	40.89	63.86	46.07	81.83	128.00	92.14	163.92	255.96
SL-coloring colors	18	26	37	29	46	65	50	82	115
Backbones (BB)	6	10	14	10	17	25	19	31	47
Avg. BB size	938.17	585.7	409.00	1091.30	659.47	443.36	1189.00	711.10	469.13
Avg. BB degrees	2.22	2.43	2.67	2.57	2.79	2.94	2.82	3.02	3.10
Avg. BB components	29.5	6.2	1.64	4.50	1.82	1.04	1.84	1.10	1.00
Avg. BB faces	135.83	134.5	140.29	321.00	265.35	210.88	490.79	364.71	261.21
Avg. BB face sizes	17.48	11.42	8.02	9.10	7.07	6.27	6.96	5.92	5.59
Avg. BB dominates	99.95%	99.98%	100%	99.99%	100%	100%	100%	100%	100%

3. REFERENCES

[1] P. Arun Prasad, R. Ramalakshmi, "Performance analysis of CDS-based hybrid path in WSN for condition monitoring," *Information & Communication Technologies (ICT), 2013 IEEE Conference on* , pp.798-802, 2013.

[2] R. Asgarnezhad, J. Akbari Torkestani. "A New Classification of Backbone Formation Algorithms for Wireless Sensor Networks." In *ICSNC 2011, The Sixth International Conference on Systems and Networks Communications*, pp. 47-54. 2011.

[3] D. Mahjoub, D. W. Matula, "Constructing efficient rotating backbones in wireless sensor networks using graph coloring", In *Computer Communications*, vol. 35, no. 9, pp. 1086-1097, 2012.

[4] D. Mahjoub and D. W. Matula. "Employing (1-epsilon) Dominating Set Partitions as Backbones in Wireless Sensor Networks." In *ALENEX*, pp. 98-111. 2010.

[5] D. W. Matula and L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," J. of the ACM, vol. 30, no. 3, pp. 417–427, 1983.

[6] D. W. Matula, "A min-max theorem for graphs with application to graph coloring," *SIAM Review*, nol. 10, pp. 481–482, 1968.

Characterizing asynchronous broadcast trees for multifrontal factorizations

Patrick R. Amestoy, Jean-Yves L'Excellent, Wissam M. Sid-Lakhdar

To solve sparse systems of linear equations, multifrontal methods [2] rely on partial LU decompositions of dense matrices called fronts. The dependencies between those decompositions form a tree, which must be processed from bottom to top in a topological order. We consider a parallel asynchronous setting where 1D acyclic pipelined decompositions are used. At each node N_i of the multifrontal tree, factored panels have to be broadcast to other processes involved in N_i. Because of the asynchronous environment considered, we use $w-$ary broadcast trees aiming at better controlling communication memory and pipeline efficiency than, for example, a binomial tree or a standard `MPI_IBCAST` primitive.

In our asynchronous model, memory is needed for the communication of factored panels. In particular, a process involved in a broadcast tree will store a factored panel (e.g., on reception), will relay (or just send, for the root of the broadcast tree) it to all its successors in the broadcast tree, and the memory for the panel will be freed only when all successors have received the panel sent. When memory for communications is limited (for a large problem, a typical panel to be sent might require 200 Mbytes) deadlocks may appear. In this work, we aim at avoiding deadlocks while designing efficient communication patterns, using the available communication memory as much as possible for performance.

Let us examine a simple case of deadlock. Let 1, 2, x, y, a and b be processes involved in the computation of two (fronts) tasks T_1 and T_2. Fig. 1 (left) shows broadcast tree branches of T_1 and T_2, with arrows representing messages paths. On Fig. 1 (right), an arrow $i \to j$ indicates that freeing a memory resource on j (corresponding to a message sent to i) depends on i performing the associated reception. Depending on the order of messages, there can be a cycle (in red) in the dependency graph between resources. Assuming that each process only has one communication buffer, if 1 receives a message from x and 2 receives a message from y, when 1 relays its message to 2, 2 is already full and will not be able to receive the message from 1. Similarly, 1 will not be able to receive the message from 2 leading to a deadlock [1].

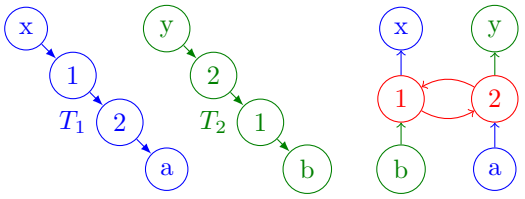


Figure 1: Branches in broadcast trees (left) with a possible cycle between resources (right).

Assuming that each process has two memory buffers for communication instead of one, the deadlock may still happen if process 1 fills its two buffers with messages from x (thus not receiving messages from 2) and 2 receives two messages from y (thus not receiving messages from 1). The only way to avoid the deadlock is to receive and relay messages to the leaves a and b before resources are full of messages in the cycle shown in Fig. 1 (right). Reserving one resource for T_1 and another one for T_2 on both 1 and 2 also prevents the deadlock, although resources may be wasted if T_1 and T_2 are not both active together. A more dynamic approach consists in receiving messages in natural order as much as possible, and then avoiding deadlocks by forcing the last available resources to be used for messages that can be relayed outside the cycles. In this situation, deadlock avoidance could here consist in having 1 and 2 receiving messages that can be relayed to the leaves a and b. Property 1 provides a simpler solution, not requiring any knowledge of distributed (dynamic) cycles [3]:

Property 1. *Assume that a global order has been defined between tasks (nodes). If, each time a process P_i only has one remaining free communication resource (others being busy), it dedicates this ressource to communications involving the smallest task it is mapped on, then deadlocks cannot occur.*

Coming back to 1D asynchronous factorizations, the overall approach is sketched in Algorithm 1 and relies on fairly standard hypothesis for a fully asynchronous context: **(H1)** Computation and relay operations associated to a message are atomic (line 3 of the algorithm). In particular, a message arriving too soon is not relayed before local operations can be done. **(H2)** At each node of a broadcast tree, if memory is available, the message is sent to all successors in the broadcast tree (send to all or to no one). **(H3)** If m_1 is sent from P_i to P_j before m_2, then m_1 is received by P_j before m_2.

```
1: while (! global termination) do
2:    if (some received messages can be processed) then
3:       process them (computations followed by relay in
         broadcast trees)
4:    else
5:       check whether a new local node can start: activation
         of new broadcast tree (multiple non-blocking sends)
6:    end if
7: end while
```

Algorithm 1: Asynchronous multifrontal scheme.

Fig. 2 shows an example of partial decomposition of a dense matrix, that can be interpreted in the multifrontal method as a chain of fronts in the multifrontal

tree. At each node of this chain (here with just a child C and a parent P), one process sends panels to other processes using broadcast trees (TC for child process 1 and TP for parent process 2). The fact that red panels must be computed and treated before blue ones is naturally represented by a causality link between TC and TP, formally defined as follows.

Definition 1. *Let TC and TP be two broadcast trees. We define the child-parent* **causality link** *between TC and TP by the relation: $\forall P_i \in TP$, if $P_i \in TC$, all activities of P_i in TC must be finished before any activity of P_i in TP can start.*

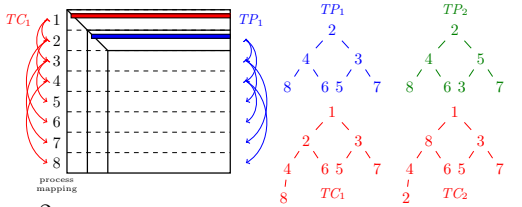


Figure 2: 1D pipelined factorization and several broadcast trees: TC for child, TP_1 or TP_2 for parent. We assume here that process mapping remains unchanged between C and P so that the root of TC does not work in TP.

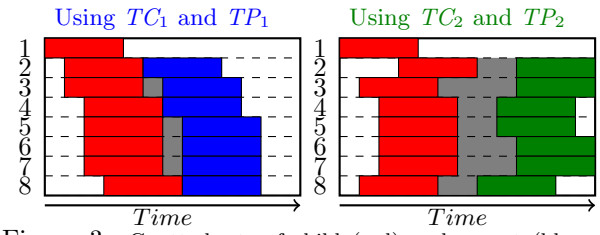


Figure 3: Gantt-charts of child (red) and parent (blue or green) operations; idle periods in gray.

On multifrontal chains or trees it may happen that local ressources of a process, say P_i, are busy because of messages arriving too early or that are not effectivly sent (receiver busy). To apply Property 1 and thus to avoid deadlocks, the last available resource should then be dedicated to communications related to the *smallest* unfinished node involving P_i (smallest in the sense of a global order compatible with causality links, that is, any topological order). In this context, the overhead will depend on the communication patterns and thus on the structure of the broadcast trees, as will be demonstrated in the following.

Definition 2. *Let TC and TP be two broadcast trees. TP is said to be* **IB-compatible** *with TC if, $\forall N \in TP \cap TC, \exists A \in \{$ancestors of N in $TP\}$, s.t. $A \in$ subtree in TC rooted at $pred_{TC}(N)$, the predecessor of N in TC.*

Property 2 (No wait)**.** *Given a child C and a parent P such that TP is IB-compatible with TC. If a process P_i in TP performs a blocking receive on a given message in TC to respect causality links, then P_i will not wait because the expected message has already been sent.*

Definition 3. ABCw trees *(Asynchronous Broad-Cast trees) are defined by the following characteristics, at each level of a multifrontal chain:*
(1) IB-compatibility of TP with TC (no wait);
(2) Width w determined by network topology;
(3) Number of nodes in each child subtree of any node is balanced: difference is at most 1 (this implies minimal height and balanced communications);
(4) Maximum pipeline efficiency between successive child and parent trees (e.g., Fig. 3 shows that TC_1/TP_1 are much better than TC_2/TP_2).

In order to build all ABCw trees for the chain, we start from a tree at the bottom node respecting (2) and (3). We then apply successive *Ascensions* to build parent trees from child trees. An ascension builds TP from TC by taking the branch in TC whose nodes are roots of the heaviest child subtrees at each level, and then making each node in that branch replace its parent. An example is given in Fig. 2, where $1 \to 2 \to 4 \to 8$ in TC_1 is replaced by $2 \to 4 \to 8$ in TP_1. The mapping of the processes at the bottom node induces the mapping of processes in the ABCw trees such as to respect the following properties: (i) at each level, the root of TP is a child of the root of TC; and (ii) the order of the roots of the successive ABCw trees in the chain follows the mapping of processes in the initial front (here 12345678). It can then be proved that by construction ascensions guarantee (1) and (4) and that, if the initial broadcast tree (at chain bottom) has properties (2) and (3), they will propagate on all broadcast trees in the chain.

To conclude, we have proposed in the context of asynchronous multifrontal methods properties avoiding deadlocks and broadcast trees providing good performance in the case of chains of nodes with no remapping between nodes. It can be shown that the lost process at each node of the chain may be reused anywhere in the multifrontal tree with no risk of deadlock. As an extension to this work, we work on using ABCw trees to: (i) the case where rows are remapped between a child and a parent (ii) the case of general trees. In both cases, the causality definition can be modified to take into account messages exchanges between child and parent nodes in the elimination tree.

References

[1] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, June 1971.

[2] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.

[3] A. N. Habermann. Prevention of system deadlocks. *Commun. ACM*, 12(7):373–ff., July 1969.

A Appendix

Proof of Property 1. Two necessary conditions for deadlock are resource starvation and the presence of cycles. As long as enough buffer memory is available, no deadlock can occur. Moreover, as no cycle exists in a broadcast tree, a cycle may only occur between distinct broadcast trees. Hence, the dependencies in a cycle are related to two or more tasks. Thus, if the processes in this cycle respect a global order between tasks when they have critically low buffer memory, we guarantee that at least two dependencies in the cycle cannot coexist simultaneously, as the processes will first communicate following the first dependency before starting / continuing the communications in the other dependency. The cycle is then broken. □

Proof of Property 2. Let P_i be a process mapped on $NC \in TC$ and on $NP \in TP$, which has received a message from $pred_{TP}(NP)$, but has not finished its work in TC. Respecting causality implies that as soon as P_i has only a single buffer resource available, it must post the reception and treat messages `msg` in TC (coming from $pred_{TC}(NC)$). The only way to guarantee that a message `msg` has already been sent is to find a path linking this event "P_i has posted the reception of `msg` from $pred_{TC}(NC)$" with the event "$pred_{TC}(NC)$ sends `msg` to NC". As the reception of a message of TP from $pred_{TP}(NP)$ by P_i means that all the ancestor processes of P_i in TP have relayed all the messages in TC (respect of causality) and as one of P_i's ancestors (A) in TP is also mapped in TC in the subtree rooted at $pred_{TC}(NC)$ (IB-compatibility of TP with TC), this implies that all the processes between this ancestor and $pred_{TC}(NC)$ in TC (in particular $pred_{TC}(NC)$) have relayed all the messages in TC (in particular `msg`). Hence, `msg` is guaranteed to have been already sent. More precisely, it has been sent by $pred_{TC}(NC)$ to the sibling of NC subtree that contains A, and thus – thanks to **(H2)** – it has also been sent to NC. □

Generalization of Property 2. Property 2 only considers a child and a parent but can be generalized to a chain of nodes where each child tree is IB-compatible with the corresponding parent tree: if P_i must receive a message corresponding to the smallest (lowest) active front P_i is mapped on, then it can be proven that the message has already been sent. The basic idea of the proof is that, if a path linking events between two succesive fronts exists, it also exists in a chain of successive fronts. This generalization is actually necessary to obtain the "No wait" property of ABCw trees not only between child and parent but also between grandchild and grandparent, ...

Comparing Different Cycle Bases for a Laplacian Solver

Erik G. Boman* Kevin Deweese† John R. Gilbert†

1 Kelner et al.'s Randomized Kaczmarz Solver

Solving linear systems on the graph Laplacian of large unstructured networks has emerged as an important computational task in network analysis [7]. Most work on these solvers has been on preconditioned conjugate gradient (PCG) solvers or specialized multigrid methods [6]. Spielman and Teng, showed how to solve these problems in nearly-linear time [8], later improved by Koutis et al. [5] but these algorithms do not have practical implementations. A promising new approach for solving these systems proposed by Kelner et al. [4] involves solving a problem that is dual to the original system.

The inspiration for the algorithm is to treat graphs as electrical networks with resistors on the edges. The graph Laplacian is defined as $L = D - A$ where D is a diagonal matrix containing the sum of incident edge weights and A is the adjacency matrix. For each edge, the weight is the inverse of the resistance. We can think of vertices as having an electrical potential and net current at every vertex, and define vectors of these potentials and currents as $\vec{v}$ and $\vec{\chi}$ respectively. These vectors are related by the linear system $L\vec{v} = \vec{\chi}$. Solving this system is equivalent to finding the set of voltages that satisfy the currents. Kelner et al.'s SimpleSolver algorithm solves this problem with an optimization algorithm in the dual space which finds the optimal currents on all of the edges subject to the constraint of zero net voltage around all cycles. They use Kaczmarz projections [3] [9] to adjust currents on one cycle at a time, iterating until convergence. They prove that randomly selecting fundamental cycles from a particular type of spanning tree called a "low-stretch" tree yields convergence with nearly-linear total work.

*Sandia National Laboratories, Sandia is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000. eboman@sandia.gov

†UC Santa Barbara Dept. of Computer Science, Supported by Contract #618442525-57661 from Intel Corp. and Contract #8-48252526701 from the DOE Office of Science. kdeweese@cs.ucsb.edu, gilbert@cs.ucsb.edu

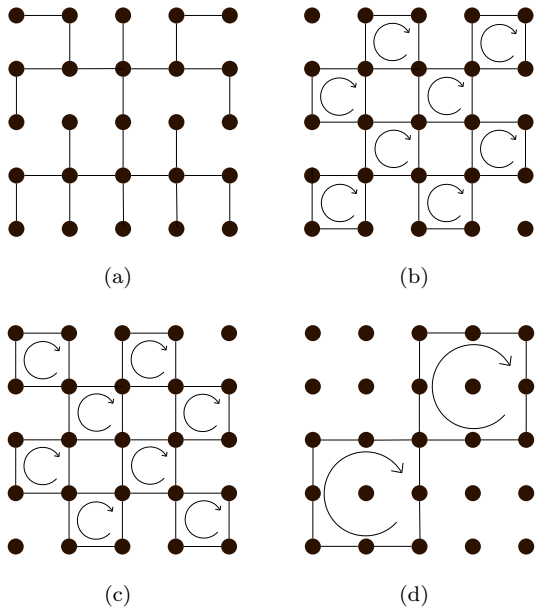


Figure 1: Grid Cycles

2 Choosing the Cycle Basis

We examine different ways to choose the set of cycles and their sequence of updates with the goal of providing more flexibility and potential parallelism. Our ideas include the following.

- Provide parallelism by projecting against multiple edge-disjoint cycles concurrently.
- Provide flexibility by using a non-fundamental cycle basis.
- Provide flexibility by using more (perhaps many more) cycles than just a basis.
- Accelerate convergence by varying the mixture of short and long cycles in the updating schedule.

Sampling fundamental cycles from a tree will require updating several potentially long cycles which will not be edge-disjoint. It would be preferable to update edge-disjoint cycles as these updates could be done in parallel. Instead of selecting a cycle basis from a

spanning tree, we will use several small, edge-disjoint cycles. We expect updating long cycles will be needed for convergence, but we consider mixing in the update of several short cycles as they are cheap to update and have more exploitable parallelism. These cycles can then be added together to form larger cycles to project against in a multigrid like approach.

An example of these cycles can be seen on the 5 by 5 grid graph in Figure 1. Figure 1(a) shows a spanning tree in which each cycle is determined by an edge not in the tree. The smallest cycles of a non-fundamental scheme are shown in Figures 1(b)(c). All the cycles in each of these two figures are edge-disjoint and can be updated in parallel. They can also be summed together as in Figure 1(d).

3 Preliminary Experiments and Results

We performed our initial experiments on grid graphs of various sizes. We used a non-fundamental set of cycles with a hierarchical ordering. The smallest set of cycles are updated. Then the cycles are coarsened and the next level of cycles are updated. This is done until reaching the perimeter cycle before resetting back to updating the smallest cycles. We also implemened the SimpleSolver algorithm in Matlab, except that we used a random spanning tree for sampling instead of a low-stretch tree. We also haven't implemented a clever data structure Kelner et al. use to quckly update edges. We also compared our results to PCG with Jacobi.

The metric we choose for comparison is the total number of edges updated, or matrix elements touched in CG. We can see the total work measured in edges updated in Table 1. Also shown in the table is an estimated potential parallelism using the work-span model [10]. The span, or critical path length, is the maximum number of edges that would have to be updated by a single processor if we can split the work over infinitly many processors.

Grid Size (Vertices)	25	289	4,225
Fundamental Cycles Work	8K	1.4M	296M
Alternative Cycles Work	1K	.08M	4M
Alternative Cycles Span	.5K	8.4K	105.8K
PCG Work	1K	.09M	5M

Table 1: Edges Updated

4 Conclusions and Future Work

Our preliminary experiments show that choosing a non-fundamental set of cycles can save significant work compared to a fundamental cycle basis, and can be at least competitive with PCG.

We are exploring ways to find a non-fundamental cycle basis of more general graphs; one challenge is how best to find large sets of short edge-disjoint cycles for parallelism. Our ideas for cycle finding include short-cuts to the spanning tree cycles and growing small cycles locally around vertices and edges. We also plan to make a rigorous comparison with several other preconditioned CG methods, including incomplete Cholesky and support-graph techniques.

We note that any of these graph Laplacian solvers can be extended to general symmetric diagonally dominant systems via standard reduction techniques. [1] [2].

References

[1] E. G. Boman, D. Chen, B. Hendrickson, and S. Toledo. Maximum-weight-basis preconditioners. *Numerical Linear Algebra Appl.*, 11:695–721, 2004.

[2] K. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996.

[3] S. Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bulletin International de lAcademie Polonaise des Sciences et des Lettres*, 35:355–357, 1937.

[4] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Pro 45th ACM Symp. Theory of Comp.*, (STOC '13), pages 911–920, New York, 2013.

[5] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving sdd systems. *CoRR*, abs/1003.2958, 2010.

[6] O. E. Livne and A. Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Scientific Comp*, 34(4):B499–B522, 2012.

[7] D. A. Spielman. Algorithms, graph theory, and linear equations in Laplacian matrices. In *Proceedings of the International Congress of Mathematicians*, volume 4, pages 2698–2722, 2010.

[8] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-sixth Annual ACM Symp. on Theory of Comp.*, STOC '04, pages 81–90, New York, NY, USA, 2004. ACM.

[9] S. Toledo. An algebraic view of the new randomized Kaczmarz linear solver. Presented at the Simons Institute for the Theory of Computing, 2013.

[10] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice Hall, 2004.

An Efficient Graph Coloring Algorithm for Stencil-based Jacobian Computations

Michael Lülfesmann and H. Martin Bücker*
*Institute for Computer Science
Friedrich Schiller University Jena
07737 Jena, Germany

***6th SIAM Workshop on Combinatorial Scientific Computing*, July 21–23, 2014, Lyon, France**

Numerical methods for the solution of partial differential equations constitute an important class of techniques in scientific computing. Often, the discretization is based on approximating the partial derivatives by finite differences on a regular Cartesian grid. The resulting computations are structured in the sense of updating a large, multidimensional array by a stencil operation. A stencil defines the update of the value at a grid point based on values at neighboring grid points.

We consider the problem of computing the Jacobian matrix of some functions that is given in the form of a computer program involving stencil operations on a regular Cartesian grid. Due to the stencil operations this Jacobian matrix is sparse. The exploitation of the stencil-type to compute this sparse Jacobian matrix using automatic differentiation with minimal computational effort can be modeled as a graph coloring problem [1]. By definition, exact solutions of this combinatorial optimization problem use the minimal number of colors. Exact solutions in terms of explicit formuæ are known for various stencil types [2, 4, 5]. However, a formula for the exact solution is not readily available for an arbitrary stencil type. So, by ignoring any structure implied by a given stencil, it is not uncommon to use coloring heuristics for general graphs. These heuristics try to approximate the exact solution and will, most likely, not attain the minimal number of colors. Recently, Lülfesmann and Kawarabayashi [3] introduced a graph coloring algorithm for an arbitrary stencil on a regular multidimensional Cartesian grid that computes the exact solution. This algorithm eliminates the need for deriving an explicit formula for the exact solution. It is based on a divide-and-conquer approach that establishes a hierarchy of vertex separators that recursively decomposes the grid into smaller and smaller subgrids. The main advantage of this algorithm is that it always computes a coloring with the minimal number of colors. However, the disadvantage of this algorithm is its high computational complexity. In fact, there are problem instances reported in [3] where, compared to the running time of a traditional graph coloring heuristic, the running time of this algorithm is larger than a factor of more than 800.

So, there is urgent need to look for alternative ways to compute exact solutions of this structured graph coloring problem while reducing the resulting running time. In this extended abstract we propose a novel graph coloring algorithm for stencil-based Jacobian computations on a regular Cartesian grid whose running time is independent of the grid size. The main advantage of this new approach is twofold: First, it computes a coloring with a minimal number of colors. Second, its computational complexity is low. The disadvantage is that we currently can not prove that this algorithm computes a solution for every given stencil type. It is currently open whether or not there is any stencil type where the algorithm terminates without computing a solution. However, we carried out extensive numerical experiments

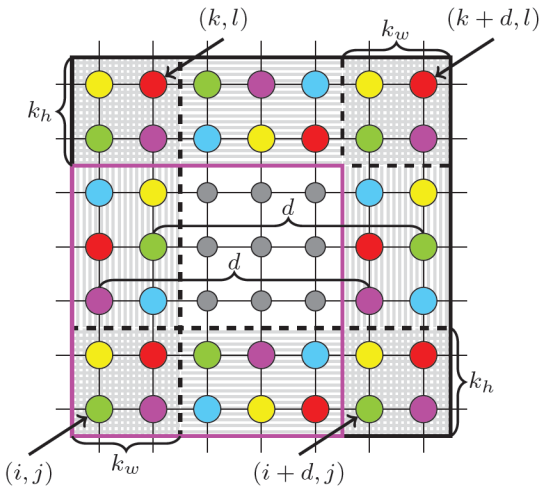


Figure 1: A 7×7 colored tile with a minimal coloring for the five-point stencil.

varying stencil types and observed that the new algorithm successfully computed a solution for all considered stencil types.

The main idea of the new algorithm is to color a small subgrid whose coloring allows to color a larger grid of arbitrary size. We call such a subgrid a *colored tile*. For the sake of simplicity, the following discussion is restricted to grids in two dimensions. However, the algorithm also generalizes to multidimensional Cartesian grids. A colored tile consists of two pairs of rectangular regions with the following property. A pair of two rectangular regions is called *consistent* if and only if the two rectangular regions have the same number of grid points in each grid dimension and all corresponding grid points are colored identically. In Figure 1, the pair of the two 2×7 rectangular regions on the left and right border of the 7×7 colored tile are consistent, because the color of a grid point (i, j) is identical to the color of the grid point $(i + d, j)$. Similarly, the top and bottom 7×2 rectangular regions are also consistent. Therefore, the 5×5 region indicated by the box with the purple frame in the bottom left corner can be used to color a larger grid by repeatedly placing this region next to each other in horizontal and vertical direction. To ensure a valid coloring, the width k_w and height k_h of the consistent rectangular regions are important and depend on the given stencil type. These values are chosen by taking into account grid points which we call *structurally non-orthogonal*.

References

[1] A. H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, 2005.

[2] D. Goldfarb and P. L. Toint. Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations. *Mathematics of Computation*, 43(167):69–88, 1984.

[3] M. Lülfesmann and K. Kawarabayashi. Sub-exponential graph coloring algorithm for stencil-based Jacobian computations. *Journal of Computational Sciene*, 5(1):1–11, 2014.

[4] D. K. Melgaard and R. F. Sincovec. General software for two-dimensional nonlinear partial differential equations. *ACM Transactions on Mathematical Software*, 7(1):106–125, 1981.

[5] G. N. Newsam and J. D. Ramsdell. Estimation of sparse Jacobian matrices. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):404–417, 1983.

Computing Approximate b-Matchings in Parallel

Arif Khan[†], Mahantesh Halappanavar[‡], Fredrik Manne[*], Alex Pothen[†]
[†]Purdue University; (khan58, apothen)@purdue.edu
[‡]Pacific Northwest National Laboratory; Mahantesh.Halappanavar@pnnl.gov
[*]University of Bergen; Fredrik.Manne@ii.uib.no

We consider sequential and shared-memory parallel (on multicore computers) algorithms that implement a half-approximation algorithm for weighted b-matching on arbitrary graphs. Consider an undirected graph $G(V, E, w)$ with vertex set V, edge set E, and weight function $w(e) \geq 0$ for each $e \in E$, and a function $f : V \to \mathbb{Z}_+$ assigning non-negative integers to the vertices. (We assume without loss of generality that $f(v)$ is less than or equal to the degree of the vertex v.) Then a b-matching on G is a subset of edges $M \subseteq E$ such that every vertex $v \in V$ has at most $f(v)$ edges in M incident on it. The values $f(v)$ for each vertex v could be be different or the same (in the latter case $f(v) = b$ for some positive integer b, and hence the name b-matching). The usual notion of matching has $f(v) = 1$ for all v, and we will call it a 1-matching. If all vertices in M are required to have degree exactly $f(v)$, we call it a perfect b-matching. A *maximum cardinality* b-matching M has the cardinality $|M|$ as large as possible. A *maximum weight* b-matching M has the sum of weights $\sum_{e \in M} w(e)$ as large as possible. In this abstract we focus on the *maximum weight* b-matching with $f(v) >= 2$ for all v.

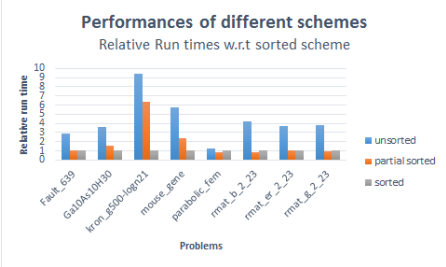
The applications of 1-matching problem include Google's Ad words problem, image recognition in computer vision, network alignment, sparse matrix computations, etc. Jabera et al. [4] have shown that b-matching is useful in various machine learning problems such as classification, spectral clustering, semi-supervised learning and graph embedding.

Let m denote the number of edges and n the number of vertices in G. The fastest exact algorithm for this problem, by Gabow and Tarjan [1], requires $O(n^{1/2}m)$ time. Fremuth-Paeger et al. [6] and Jabera et al. [4] describe exact algorithms that use min-cost flow and belief propagation techniques, respectively. Both of these algorithms have running time $O(nm)$, but the belief propagation technique is currently the fastest practical algorithm. However, these running times are prohibitive in case of even moderate-sized graphs, and hence we design linear-time approximation algorithms. The approximate edge weighted 1-matching problem has also been studied. Mestre [3] showed 1/2- and $(2/3 - \epsilon)$-approximation algorithms for b-matching by extending path-growing approximation algorithms for the 1-matching problem. Morales et al. [2] and Georgiadis et al. [5] developed $1/2-$approximation algorithms for b-matching based on the concept of *locally dominant edges*. Although the latter describes a distributed algorithm and uses different notation, their algorithm is similar to ours.

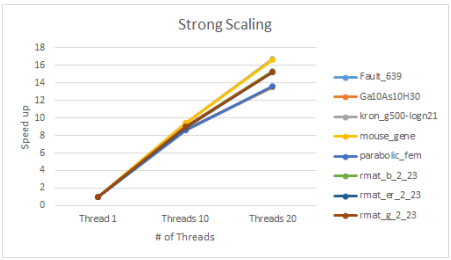
Here we propose a new $1/2-$approximation *maximum weight* b-matching algorithm, called *b-Suitor*, which we show to be practically faster than earlier algorithms. The *b-Suitor* algorithm is an extension of the *Suitor* algorithm proposed by Manne and Halappanavar [7] for solving the maximum weight 1-matching problem. All of these are serial algorithms. We also study these algorithms in the parallel (shared memory) context.

In the *b-Suitor* algorithm each vertex v proposes to a set of $f(v)$ vertices (v is a *suitor* of these vertices). Hence each vertex maintains two lists of vertices: $S(v)$ is the list of current Suitors of v, i.e., vertices that propose to match to v, and $T(v)$ is a list of vertices that v has proposed to. The process responsible for matching a vertex v includes in $T(v)$ its heaviest available $f(v)$ neighbors, where a neighbor w is unavailable if it has $f(w)$ heavier Suitors than v. This is a speculative algorithm: If v finds that it is heavier than the $f(w)$-th Suitor of w, call it y, then it unmatches y and includes itself as a Suitor of w. This process now tries to find a neighbor not included in the list $T(y)$ for y to propose to.

We can describe three variants of this algorithm, based on whether the adjacency lists of the vertices are sorted in non-increasing order of weight or not. We can choose to have unsorted



(a) Relative performance of variant algorithms



(b) Strong scaling

Figure 1: The performance of three variant Suitor algorithms, and strong scaling of the sorted Suitor algorithm for computing a 5-matching on a multicore computer.

adjacency lists, or fully sorted lists; the third option is to partially sort so that small multiples of $f(v)$ highest weighted vertices in each adjacency list are sorted. The time complexity of the unsorted algorithm is $O(m\Delta B)$, where $B = \max\{f(v) : v \in V\}$, and Δ is the maximum vertex degree. For the sorted algorithm it is $O(n + m \log \Delta)$,

We present preliminary results on computing a 5-matching from a shared memory parallel implementation of the b-Suitor algorithm. The machine is an Intel Xeon multiprocessor, with ten cores per socket, and the computer consists of two sockets. The processor speed is 3.0 GHz, and the system has 128 GB memory. Figure 1 shows the performance of eight problems, five of which are from the the University of Florida collection and three which are synthetic `RMAT` graphs. We report the first set of results for the parallel b-Suitor algorithm with $b = 5$. The results in Figure 1(a) show that sorting leads to faster algorithms, by factors upto nine, for graphs with several hundred million edges. Complete sorting seems to be better than partial sorting for most of these problems. Figure 1(b) shows strong scaling results on the twenty Intel Xeon cores, and all eight problems show good speed-ups. We will discuss the factors that influence the performance and scalability of these problems at the Workshop. We are also working with colleagues at Intel Corporation on Xeon Phi implementations.

References

[1] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM Journal of Computing*, vol. 5, no. 18, pp. 1013–1036, 1989.

[2] G. D. F. Morales, A. Gionis, and M. Sozio, "Social content matching in Mapreduce," in *"37th Int. Conf. on Very Large Data Bases (VLDB 2011)"*, vol. 4, no. 7, 2011, pp. 460–469.

[3] J. Mestre, "Greedy in approximation algorithms," in *Algorithms - ESA 2006*, Lecture Notes in Computer Science, vol. 4168. Springer, 2006, pp. 528–539.

[4] B. C. Huang and T. Jebara, "Fast b-matching via sufficient selection belief propagation," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, ser. JMLR Proceedings, vol. 15. 2011, pp. 361–369.

[5] G. Georgiadis and M. Papatriantafilou, "Overlays with preferences: Distributed, adaptive approximation algorithms for matching with preference lists," *Algorithms*, vol. 6, no. 4, pp. 824–856, 2013.

[6] Fremuth-Paeger, Christian, and Dieter Jungnickel. "Balanced network flows. I. A unifying framework for design and analysis of matching algorithms." Networks 33.1 (1999): 1-28.

[7] F. Manne and M. Halappanavar. "New effective multithreaded matching algorithms", Proceedings of IPDPS 2014, to appear.

Performance Analysis of Single-source Shortest Path Algorithms on Distributed-memory Systems

Thap Panitanarak Kamesh Madduri
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA, USA
{txp214, madduri}@cse.psu.edu

1 Introduction

We consider the problem of single-source shortest path (SSSP) computation in a distributed setting, where a large sparse graph with non-negative edge weights is partitioned across the nodes of a parallel system. SSSP is a key computation arising in large-scale network analysis and a possible candidate for inclusion in the Graph500 benchmark. *Work-efficient* SSSP algorithms are based on Dijkstra's algorithm [5]. However, there is limited concurrency to exploit in Dijkstra's algorithm, as it belongs to the *label-setting* class of shortest path algorithms. *Label-correcting* algorithms relax the constraint that the vertex with minimum weight be picked at each step, and these algorithms do not use a priority queue. The Bellman-Ford algorithm [1] is an example of a label-correcting and is more amenable to parallelization, but it is not work-optimal for graphs with non-negative edge weights. The Delta-stepping [11] algorithm can be considered a hybrid approach combining Bellman-Ford and Dijkstra's algorithm. Prior work has studied shared-memory [10] and distributed-memory [6, 7] implementations of Delta-stepping. There is one key parameter in this algorithm, Δ, whose setting determines the number of parallel phases, the work performed, the load-balance across multiple tasks in a parallel system, and hence the running time.

2 Our Contributions

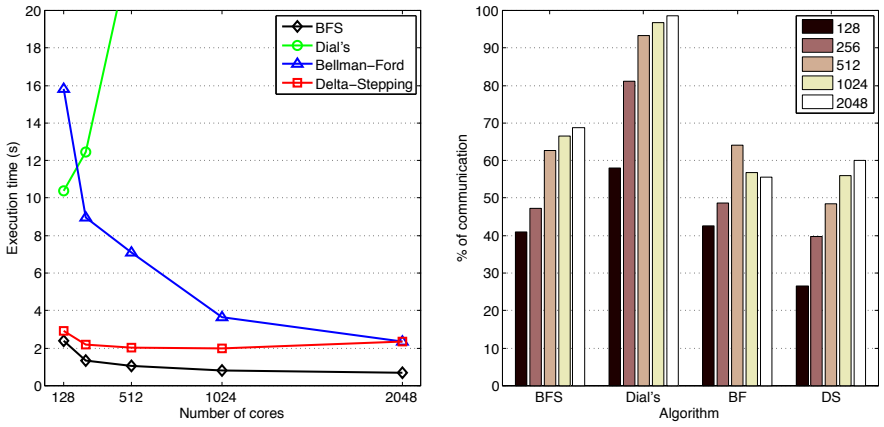
We have developed optimized parallel implementations of three SSSP algorithms – Dial's algorithm [4], Bellman-Ford, and Delta-stepping – for graphs with positive integer edge weights. Dial's algorithm is a special case of Dijkstra's algorithm for graphs with integer weights, and does not require a priority queue. Bellman-Ford also does not use a priority queue because it is a label-correcting algorithm. We use a bucket array data structure, parameterized by Δ, as the priority queue in our implementation of Delta-stepping. All three approaches are bulk-synchronous. We use a distributed compressed sparse row representation for the graph. On a system with p tasks, each task holds n/p vertices and all outgoing edges from these vertices. The distance array is also partitioned and distributed in a similar manner. Delta-stepping has light and heavy edge relaxation phases. We have designed our method so that all communication happens at the end of these phases, and all relaxations are performed locally. Our approach is thus a bulk-synchronous modification of the implementation in [10]. Further, our code has the following optimizations:

- The buckets are implemented as dynamic arrays. Each bucket is allocated only if a vertex being added to it, and it can be resized when needed. At the end of a light edge relaxation phase of bucket i, it can be deallocated and the memory can be reused for other bucket allocations, since there will be no more insertions to that bucket. Moreover, we use two auxiliary arrays of size n/p to provide constant time insertions and deletions of vertices in each bucket [2].
- A semi-sorting routine is used as a preprocessing step of the algorithm. It reorders the edges of each vertex based on edge weights and the value of Δ such that all light edges appear before heavy ones.
- Local lookup arrays are used to track the tentative distance of every vertex, avoiding duplicate requests being sent.

Additional details are discussed in a technical report [13].

We collect parallel performance results on the TACC Stampede cluster. This is a 10 PetaFlop/s Dell Linux cluster with more than 6400 Dell PowerEdge server nodes. Each node has two Intel Xeon E5 Sandy Bridge processors and an Intel Xeon Phi coprocessor. We do not use the coprocessor in the current study. Our implementation uses MPI. The main collective communication routines used in our code are Alltoallv, Alltoall, and Allreduce. We use synthetic graphs generated using the Graph500 reference implementation v1.2 [8]. The generated graphs have skewed degree distributions and a very low graph diameter. We experiment with uniform and normal edge weight distributions, as well as integer (includes Dial) and double-precision (weights in $(0, 1]$) edge weights. We also vary the maximum weight in case of graphs with integer weights.

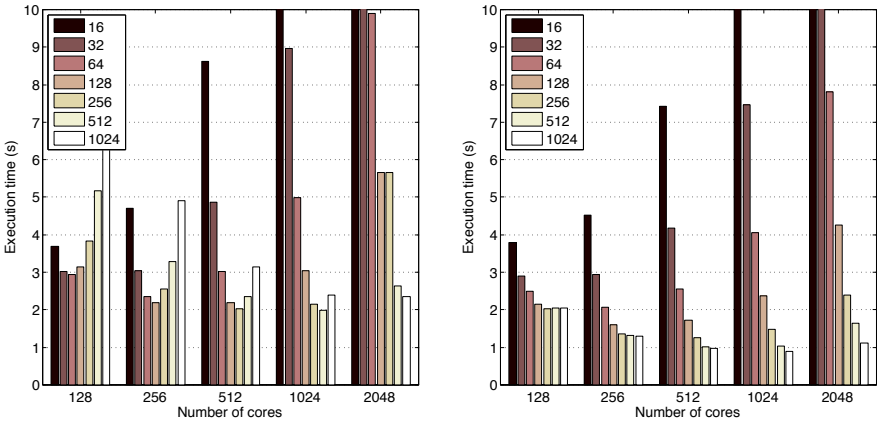
Our single-node Delta-stepping performance (16 MPI tasks) is faster than the shared-memory Chaotic relaxation approach of Galois [12, 9] on a large collection of graphs. It is 1.4× faster with uniform edges and 3.36× faster with normal weight distributions. It is also nearly 10× faster than the Parallel Boost graph library implementation on a single node.



(a) Execution time (b) Communication time

Figure 1: **Strong scaling:** Overall execution time and percentage of time spent in communication for Dial's algorithm, Bellman-Ford, and Delta-stepping, on SCALE 27 Graph500 graphs with uniform weights and max weight set to 1024.

Multinode strong scaling results are shown in Figure 1. We also report performance of a parallel Breadth-First Search (BFS) for comparison. Delta-stepping scales up to 512 cores (32 nodes), but beyond that point, there is computational load imbalance. Bellman-Ford is initially slower than Delta-stepping, but scales better due to fewer number of parallel phases. Dial's algorithm shows the worst scaling, but note that its performance is dependent on the maximum edge weight. Dial performance is comparable to BFS if all weights are set to 1. Delta-stepping would incur a significant overhead due to bucket maintenance in that setting.



(a) Uniform weight distribution (b) Normal weight distribution

Figure 2: The effect of Δ on a SCALE 27 Graph500 network.

Note from Figure 2 that the performance of Delta-stepping is heavily dependent on the value of Δ used in the case of uniformly-distributed edge weights. In general, larger values of Δ are preferable at higher task concurrencies.

To alleviate the load imbalance seen at 512 cores, we are also developing a parallel implementation of Delta-stepping based on a 2D edge-based partitioning and distribution of the graph. Our 2D approach supports arbitrary partitions of the graph (e.g., with 512 MPI tasks, 32×16, 64×8, 128×4, etc. are possible, and the 512×1 case corresponds to 1D row-based partitioning). The MPI collective Allgatherv is additionally used in the 2D implementation. While we verified that the 2D implementation improves load balance, our current 2D implementation is still slower than the best 1D approach (after tuning for Δ) due to bucketing implementation overhead.

In a recent paper [3], Chakaravarthy et al. present a new parallel algorithm for SSSP, and a highly optimized implementation for IBM Blue Gene/Q systems. Assuming a 1D graph distribution, this Delta-stepping based algorithm introduces a new edge pruning strategy to reduce inter-node communication, and a new load-balancing strategy for graphs with skewed degree distributions. We are extending our Delta-stepping implementation to include both these optimization strategies.

In current work, we are developing a hybrid MPI-OpenMP implementation to mitigate the load imbalance issue seen in strong scaling. We are also developing a automated scheme to choose the value of Δ given the task concurrency and the weight distribution. Thirdly, we are exploring hybrid Delta-stepping and Bellman-Ford approaches, where we switch to Bellman-Ford after a certain number of Delta-stepping phases.

Acknowledgments

This work is supported by NSF grant ACI-1253881 and used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant OCI-1053575.

References

[1] R. Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

[2] P. Briggs and L. Torczon. An efficient representation for sparse sets. *ACM Letters on Programming Languages and Systems*, 2(1-4):59–69, 1993.

[3] V. Chakaravarthy, F. Checconi, F. Petrini, and Y. Sabharwal. Scalable single source shortest path algorithms for massively parallel systems. In *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2014.

[4] R. Dial. Algorithm 360: Shortest path forest with topological ordering. *Commun. ACM*, 12:632–633, 1969.

[5] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] N. Edmonds, T. Hoefler, and A. Lumsdaine. A space-efficient parallel algorithm for computing betweenness centrality in distributed memory. In *Proc. ACM Int'l. Conf. on High Performance Computing (HiPC)*, 2010.

[7] N. Edmonds, J. Willcock, and A. Lumsdaine. Expressing graph algorithms using generalized active messages. In *Proc. ACM Int'l. Conf. on Supercomputing (ICS)*, 2013.

[8] Graph500 benchmark. `http://www.graph500.org/`, last accessed Feb 2014.

[9] M. Kulkarni, K. Pingali, B. Walter, G. Ramanarayanan, K. Bala, and L. P. Chew. Optimistic parallelism requires abstractions. In *Proc. ACM SIGPLAN Conf. on Programming language design and implementation (PLDI)*, 2007.

[10] K. Madduri, D. A. Bader, J. Berry, and J. R. Crobak. An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. In *Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007.

[11] U. Meyer and P. Sanders. Δ-stepping: a parallelizable shortest path algorithm. *J. Algs.*, 49(1):114–152, 2003.

[12] D. Nguyen, A. Lenharth, and K. Pingali. A lightweight infrastructure for graph analytics. In *Proc. ACM Symp. on Operating Systems Principles (SOSP)*, 2013.

[13] T. Panitanarak and K. Madduri. Performance analysis of single-source shortest path algorithms on distributed-memory systems. Technical Report CSE #14-003, The Pennsylvania State University, 2014.

Improving the Runtime Efficiency and Solution Quality of Independent Task Assignment Heuristics

E. Kartal Tabak[a], B. Barla Cambazoglu[b], Cevdet Aykanat[c]

[a] *HAVELSAN A.S., Ankara, Turkey*
[b] *Yahoo! Labs, Barcelona, Spain*
[c] *Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey*

We focus on the independent task assignment problem which is defined as assigning N independent tasks to K heterogeneous processors. In this problem, given an Expected-Time-to-Compute matrix that identifies the expected execution times of each independent task on each processor, the objective is to generate a task-to-processor assignment that achieves a minimum makespan. This problem is known to be NP-complete and MinMin, MaxMin, and Sufferage are successful heuristics that are widely used in the literature to solve this problem [1, 2, 3, 4]. All of these heuristics are constructive in nature and they all run in $O(KN^2)$ time. In [5], we propose a $O(KN \log N)$-time MinMin algorithm that achieves the same solution quality as the conventional MinMin algorithm. The proposed algorithm achieves this asymptotic improvement through utilizing a processor-oriented approach instead of the task-oriented approach of the original MinMin algorithm. In [5], we also propose to improve the performance of MaxMin and Sufferage heuristics, by combining the proposed asymptotically faster MinMin heuristics with these two heuristics. The proposed MaxMin heuristic improves the runtime performance of the conventional MaxMin and also improves the solution quality by adaptive use of MinMin and MaxMin assignment decisions during the assignment iterations. We propose a similar improvement on Sufferage heuristic that leads to an improvement on the runtime without disturbing the solution quality. The proposed algorithm achieves critical assignment decisions by conventional Sufferage in order not to disturb the solution quality and achieve non-critical assignment decisions by the fast MinMin algorithm. For the assignment of the 2.5 million tasks of a real-world dataset to 16 heterogeneous processors, the conventional MinMin algorithm generates a solution in three weeks, whereas the proposed MinMin heuristic generates the same assignment in less than a minute. Experimental results on these real-world datasets also show that the proposed MaxMin and Sufferage heuristics run considerably faster than the original heuristics. On the average, the proposed MaxMin heuristic is found to generate considerably better solutions than the original MaxMin, whereas the proposed Sufferage heuristic is found to generate slightly better solutions than the original Sufferage heuristic.

References

[1] Robert Armstrong, Debra Hensgen, and Taylor Kidd. The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in run-time Predictions. In *Proceedings of the 7th IEEE Heterogeneous Computing Workshop*, pages 79–87, Washington, DC, USA, 1998. IEEE Computer Society.

[2] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Lasislau L. Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[3] Oscar H. Ibarra and Chul E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM*, 24(2):280–289, April 1977.

[4] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[5] E. Kartal Tabak, B. Barla Cambazoglu, and Cevdet Aykanat. Improving the Performance of Independent Task Assignment Heuristics MinMin, MaxMin and Sufferage. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1244–1256, May 2014.

Towards the Parallel Resolution of the Langford Problem on a Cluster of GPU Devices

Hervé Deleau*, Christophe Jaillet*, Michaël Krajecki*, Julien Loiseau*,
Luiz Angelo Steffenel* and François Alin†
*Université de Reims Champagne-Ardenne, Reims, France
†Lycée Franklin Roosvelt, Reims, France
{*firstname.lastname*}*@univ-reims.fr, julien.loiseau@etudiant.univ-reims.fr*

I. The Langford problem

The Langford problem is a classic permutation problem [1], [2]. While observing his son manipulating blocks of different colors, Langford noticed that it was possible to arrange three pairs of blocks of different colors (e.g., yellow, red, blue) in such a way that color 1 cubes were separated by 1 block, color 2 by 2 blocks, etc. (Fig. 1).

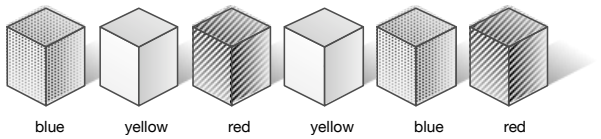


Figure 1. L(2,3): arrangement for 6 blocks of 3 colors.

The n^{th} instance of the Langford problem consists in counting the number $L(2,n)$ of such pairs arrangements (up to a symmetry). This study considers the standard Langford problem but could be generalized to any number s of blocks having the same color, in order to get the $L(s,n)$ value. Martin Gardner presented instance 4 of the problem (2 cubes and 4 colors) as being part of a collection of small mathematical games and stated that $L(2,n)$ has solutions for all n such that $n = 4k$ or $n = 4k-1$ for $k \in \mathbb{N} \setminus \{0\}$.

A. Miller's algorithm: a tree search approach

The Langford problem can be modeled as a tree search problem where look for all possible solutions. In order to solve $L(2,n)$, we consider a tree of height n where:

- every node of the tree corresponds to the place in the sequence of the cubes of a determined color;
- at the depth p, the first node corresponds to the place of the first cube of color p in first position, and the ith node corresponds to the positioning of the first cube of color p in position i, where $i \in [1, 2n-1-p]$;
- every leaf of the tree symbolizes the positions of all the cubes;
- a leaf is a solution if it respects the color constraint defined by the Langford problem: all the cubes must be in different places.

This search tree is usually implemented in a bottom-up approach (Fig. 2), where the top color is the nth color, as this approach allows tree pruning to remove symmetric results and unsolvable branches.

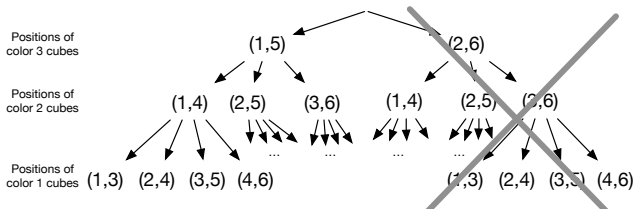


Figure 2. Search tree for $L(2,3)$ with symmetry pruning.

B. Godfrey's algorithm: algebraic method

The Miller's approach, limited to this naive tree search evaluation with backtracking, suffers from combinatorial explosion. It allowed to get $L(2,19)$ in 1999 after 2.5 years on a DEC alpha computer, but the compute time is estimated to be 10 times higher from an instance to the following one.

In 2002, an algebraic representation of the Langford problem has been proposed by Godfrey[1].
Consider $L(2,3)$ and $X = (X_1, X_2, X_3, X_4, X_5, X_6)$. It proposes to model instance 3 by $F(X,3) = (X_1X_3 + X_2X_4 + X_3X_5 + X_4X_6) \times (X_1X_4 + X_2X_5 + X_3X_6) \times (X_1X_5 + X_2X_6)$. In this approach, each term represents a position for both cubes of a given color ; the number of solutions is equal to the coefficient of $X_1X_2X_3X_4X_5X_6$ in the polynomial development. More generally, the number of solutions of instance n corresponds to the coefficient of $X_1X_2X_3X_4X_5...X_{2n}$ in $F(X,n)$.

[1]http://legacy.lclark.edu/~miller/langford/godfrey/method.html

If $G(X,n) = X_1...X_{2n}F(X,n)$ then Godfrey has shown that:

$$\sum_{(x_1,...x_{2n})\in\{-1,1\}^{2n}} G(X,n)_{(x_1...x_{2n})} = 2^{2n+1}L(2,n)$$

So:

$$\sum_{(x_1...x_{2n})\in\{-1,1\}^{2n}} (\prod_{p=1}^{2n} x_p) \prod_{p=1}^{n} \sum_{i=1}^{2n-p-1} x_i x_{i+p+1} = 2^{2n+1}L(2,n)$$

The computation of $L(2,n)$ is in $O(4^n \times n^2)$ and an efficient long integer arithmetic is needed.

By using this approach, M. Godfrey has solved $L(2,20)$ in one week on three PCs in 2002. Later, Krajecki et al. [3] solved the $L(2,23)$ and the $L(2,24)$ problem instances, the latter in 3 months, using a dozen of computers. In spite of the evolution of CPUs processing power, the solution for the next problem instance - $L(2,27)$ - would require several months of intensive computing on a whole cluster.

II. Langford deployment on multiGPU clusters

Since the end of 2000's, GPUs become a fast and less expensive alternative to massive parallel computing on CPUs. The number of GPU cores that can be aligned in a single machine is much more expressive (for example, 2688 GPU cores in a Nvidia K20Xm Kepler GPU processor, against 16 cores in a Intel Xeon CPU). Nowadays supercalculators include GPUs, and multi-GPU architectures become more frequent in the TOP500 list.

One of the main limitations of GPUs is that their cores are simpler that CPU ones and the threads spread on these cores work synchronously. This prevents to take advantage of their potential on irregular applications, based on multiple tests and branchings.

Our approach uses a bottom-up tree (i.e., starting from the n^{th} color) and combines Miller and Godfrey's techniques to efficiently perform the computation.

The Miller's tree search allows to prune the search space in order to highly reduce the search effort, but it is based on backtracking and thus cannot benefit of GPUs use. This step is therefore performed on CPU.

Our Miller's implementation is based on a binary representation of the "color" codes (for example, a color in level 1 has code "101", while a color in level 3 has code "10001") with bit-shifting and bit-wise XOR operations (Fig. 3).

Using a distributed computing middleware such as MPI or CloudFIT [4], the generated consistent masks

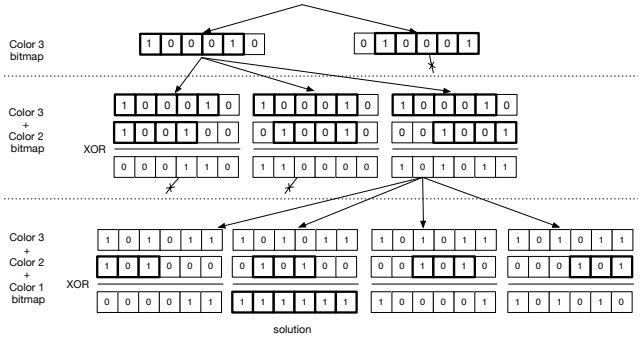


Figure 3. Bitmap evaluation of color alignments.

are treated by compute CPU-GPU clients as follows: the CPU prepares a large set of more refined masks in order to prepare grids and blocks to feed the GPU, which traverses the relative sub-tree with the regular Godfrey approach.

In order to take advantage of the GPUs compute efficiency, much effort has to be made to optimize the Godfrey implementation. In addition, code tuning imposes to fix the depth of the sub-trees to be considered by the GPU kernels; the depth dedicated to the clients grids generation is deduced from the server [Miller] masks generation and the GPU kernel [Godfrey] depth.

We have already developed the implementation of the Miller tasks generation, their distribution over the compute clients with client-server or cloud distribution, the generation of the sub-masks sets for GPU kernel computation, and a Miller regularized implementation of the kernel. The results prove our concept. The last effort to be done is to implement the GPU kernel version of the Godfrey approach, which we currently work on. We aim at the resolution of L(2,27) on the ROMEO cluster[2], a GPU-enhanced cluster ranked 151^{th} on the TOP500 listing (Nov. 2013).

References

[1] M. Gardner, *Mathematics, Magic and Mystery*, 1956.
[2] J. E. Simpson, "Langford Sequences: perfect and hooked," *Discrete Math*, vol. 44, no. 1, pp. 97–104, 1983.
[3] C. Jaillet, M. Krajecki, and A. Bui, "Parallel tree search for combinatorial problems: a comparative study between openmp and mpi," *Studia Informatica Universalis*, vol. 4, no. 2, pp. 151–190, December 2005.
[4] L. Steffenel, O. Flauzac, A. S. Charao, P. P. Barcelos, B. Stein, S. Nesmachnow, M. K. Pinheiro, and D. Diaz, "Per-mare: Adaptive deployment of mapreduce over pervasive grids," in *Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'13)*, Compiegne, France, Oct 2013.

[2]https://romeo.univ-reims.fr/

Linear Programming for Mesh Partitioning under Memory Constraint : Theoretical Formulations and Experimentations

Sébastien Morais[1,2], Eric Angel[1], Cédric Chevalier[2], Franck Ledoux[2], and Damien Regnault[1]

[1] IBISC Laboratory, University of Evry Val d'Essonne, France
[2] CEA, DAM, DIF, F91297 Arpajon, France

In many scientific areas, the size and complexity of numerical simulations lead to make intensive use of massively parallel simulations on High Performance Computing (HPC) architectures. Such architectures are mainly modelized as a set of processing units (PU) where memory is distributed. Distribution of simulation data is crucial: it has to minimize the computation time of the simulation while ensuring that the data allocated to every PU can be locally stored in memory.

In this work, we focus on numerical simulations using finite elements or finite volumes methods, where physical and numerical data are carried on a mesh. The computations are then performed at the cell level (for example triangle and quadrilateral in 2D, tetrahedron and hexahedron in 3D). More specifically, computing and memory cost can be associated to each cell. Depending on the numerical scheme to apply, a graph or an hypergraph representation of the mesh is built to perform partitioning. Such a representation is then used by tools like Metis, Patoh, Scotch or Zoltan to distribute the mesh. Traditional approaches consist in balancing the computing load between PUs while minimizing:

- either the edge cut, or hyper-edge cut, between parts ;
- or the size of a vertex separator.

Such objective functions do not rely on an important characteristic of the numerical methods used in simulation codes. The computation performed on cell i requires data from adjacent cells. As a consequence, an usual approach is to duplicate some cells between PUs. Such cells are commonly called "ghost cells" and we obtain a partitioning with covering. Although it is assumed that edge cuts is proportional to the total communication volume, it is not [Hen98]. And, for identical reason the memory footprint of ghost cells is not explicitly taken into account while minimizing edge cuts.

Load balancing is usually achieved in two steps:

1. Cells are distributed according to a balance criterion, without taking care of ghost cells, to be reached;

2. Then ghost layers are built to allow the resolution of numerical scheme locally to every PU.

However, the partitioning obtained after phase (1) does not take into account the memory footprint of ghost cells added in phase (2). For 2D meshes the size of the edge cut between two parts grows as $O(n^{\frac{1}{2}})$ and for 3D meshes it grows as $O(n^{\frac{2}{3}})$, with n the number of cells. Then, distributing a mesh on a large number of processors can bring the simulation to break off due to a lack of physical memory on a PU.

In this context, we propose a new approach for the mesh partitioning on k parts, which takes into account ghost cells and the memory constraint on each PU. We formalize this new problem by means of integer linear programming [Dan63]. In this way, we obtain a problem similar to `make_span` minimization in scheduling [CPW98], where the variables are: $x_{i,p}$ which is equal to 1 if the computations associated to the cell i are performed on part p and 0 otherwise; $y_{i,p}$ which is equal to 1 if the cell i is stored in memory on part p and 0 otherwise. The problem is:

Function:	`make_span`	$\min \ C_{max}$	
Constraints:	Assignment	$\sum_{p=1}^{k} x_{i,p} = 1$	$\forall i \in \text{Mesh}$
	Time	$\sum_{i \in \text{Mesh}} x_{i,p} \ c_{i,p} \leq C_{max}$	$\forall p \in [\![1, k]\!]$
	Memory	$\sum_{i \in \text{Mesh}} y_{i,p} \ \omega_{i,p} \leq Mem_p$	$\forall p \in [\![1, k]\!]$
	Ghost	$x_{i,p} \leq y_{i',p}$	$\forall (i, i')$ adjacent and $\forall p \in [\![1, k]\!]$

This model optimizes the computation time and obtains the resulting partition, through the variables $x_{i,p}$, while ensuring that the memory size including ghost cells does not exceed the memory bounds of each processor, through the auxiliary variables $y_{i,p}$.

To compare our approach with existing solutions, we also modelize some classical combinatorial problems such as graph partitioning, hypergraph partitioning, and graph partitioning with vertex separator by using integer linear programming. Having these various problems in the same formalism supplies first elements of comparison that we complete by comparing their solutions on benchmarks.

References

[CPW98] B. Chen, C.N. Potts, and G.J. Woeginger. A review of machine scheduling : Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization*, 1998.

[Dan63] George.B. Dantzig. *Linear Programming and Extensions*, 1963.

[Hen98] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? *Lecture Notes in Computer Science*, 1998.

OPTIMIZING THE INEXACT NEWTON KRYLOV METHOD USING COMBINATORIAL APPROACHES

Marcelo Carrion, Brenno Lugon
Maria Cristina Rangel, Lucia Catabriga, Maria Claudia Silva Boeres
Universidade Federal do Espírito Santo - Brazil

{marcelo.tperc,brennolugon}@gmail.com,{crangel,luciac,boeres}@inf.ufes.br

1 Introduction

Combinatorial Scientific Computing is an important interdisciplinary field combining issues from Combinatorial Optimization to solve efficiently Scientific Computing problems. In this work, we solve a 3D nonlinear problem using a Newton-type method that requires, at each step, the evaluation of a Jacobian matrix and the solving of a linear system. The Jacobian evaluation is optimized through matrix partitioning and a matrix reordering scheme is used to accelerate the convergence of the preconditioned iterative GMRES solver, as shown in Figure 1.

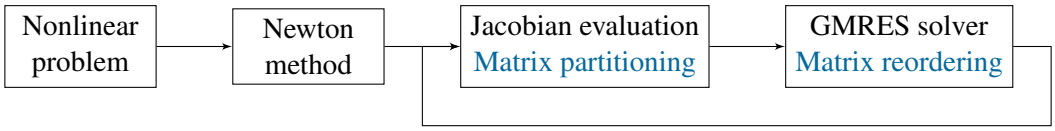


Figure 1: Problem flow and optimizations (in blue) scheme.

2 Inexact Newton Krylov Method

The Newton-type algorithm requires the solution of linear systems at each iteration. When iterative Krylov methods are used to solve these linearized systems, the resulting methods are known as Inexact Newton-Krylov methods. Inexact Newton methods are especially well suited for large-scale problems and have been used very successfully in many applications. Their success depends mainly on three factors: (i) quality of initial Newton step, (ii) robustness of Jacobian evaluation and (iii) robustness of the Krylov iterative method, the last two which we focus on.

3 Optimizations

Sparsity in the derivative matrices can be exploited to compute the nonzero entries efficiently. In finite difference approximations, efficiency can be achieved by partitioning the columns of a sparse Jacobian matrix into a few groups of structurally orthogonal ones. In each group, no two columns have a nonzero in a common row and all nonzeros can be estimated through one finite difference operation, improving the whole evaluation time of the matrix. This partitioning problem is generally modeled and solved as a graph coloring problem [1], but a different approach is used for the matrices treated here. Since they have a trivial sparsity pattern, we drew inspiration from [2] to obtain a set expression, presented in the next section, that defines a minimum partitioning.

Sparse matrix reordering schemes with the purpose of minimizing the bandwidth were also implemented in this work, as alternatives to speed up the convergence of the preconditioned GMRES method. The preconditioning technique used is based on incomplete LU factorization ILU(p), where p is the *level of fill* used to control the number of new elements generated during the process. The preconditioning calculations are optimized when we use a matrix reordering, once with the reduced bandwidth less floating-point operations are made. For the tests we adopted the Sloan reordering [3] for its good solutions achieved for the matrices generated in our specific problem.

4 Test Problem

For the numerical experiments we use a 3D heat transfer problem defined by the nonlinear differential equation:

$$-\nabla \cdot (K(u)\nabla u) = 0 \quad \text{in} \quad \Omega = (0,1) \times (0,1) \times (0,1) \tag{1}$$

where u is the temperature, the thermal conductivity is considered as $K(u) = 0.0000002u^2 + 0.00001u + 0.001$ and the boundary conditions are $u(x,y,0) = u(x,0,z) = u(1,y,z) = 10$ and $u(x,y,1) = u(x,1,z) = u(0,y,z) = 100$. Consider a discretization of Ω into an uniform grid with $N = n \times m \times l$ unknowns points, respectively, in the x, y, z directions. We approximate the derivatives by combining forward, backward and centered finite differences, arriving to the nonlinear system of equations $F(\boldsymbol{u}) = 0$, where $F : \mathbb{R}^N \to \mathbb{R}^N$ is a nonlinear vector function, $\boldsymbol{u} = (u_1, u_2, \ldots, u_N)^T$ is the unknown vector and each component of F depends only on the seven unknowns $u_{I-m\cdot n}$, u_{I-n}, u_{I-1}, u_I, u_{I+1}, u_{I+n} and $u_{I+m\cdot n}$, for $I = 1, 2, \ldots, N$. Each iteration of the Newton's method is given by $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{s}^k$, where $\boldsymbol{s}^k$ is calculated by the solution of the linear system $J(\boldsymbol{u}^k)\boldsymbol{s}^k = -F(\boldsymbol{u}^k)$. The Jacobian matrix

J represents the variation of F with respect of $\boldsymbol{u}$, that is considered as a forward finite difference approximation. We may terminate the iteration when the relative nonlinear residual $\|F(\boldsymbol{u}^k)\|/\|F(\boldsymbol{u}^0)\|$ is small. Given its heptadiagonal structure, J can be partitioned into seven groups of structurally orthogonal columns, determined by $G_p = \{i+n(j-1)+m\cdot n(k-1) \mid 1\leq i\leq n, 1\leq j\leq m, 1\leq k\leq l, (i+3j+2k) \bmod 7 = p\}$, for $p = 0,\ldots,6$.

5 Experimental Results

For the described problem, we developed programs in the C language. All computational tests were run on an Intel Core i5-3570 3.40GHz $\times 4$ machine with 4GB of RAM under Ubuntu 12.04. To store the sparse matrices derived from the problem we use an optimized storage scheme called Compressed Sparse Row (*CSR*). Eight problem instances were considered and Table 1 shows, for each instance, its dimension N, the p parameter to the ILU(p) preconditioner and the CPU time, in seconds, for computing the solution in four cases: with no optimization (T_{NO}), using only the matrix partitioning (T_{MP}) and the matrix reordering (T_{MR}) techniques and finally with both optimizations (T_{OP}). The values in parentheses on columns T_{MP}, T_{MR}, T_{OP} indicate the percentage time reduction ($red\%$) from time T_{NO}. For all instances, the optimized Jacobian evaluation performed only seven finite difference operations, as opposed to N in the regular computation. Figure 2 shows the approximate solution obtained for a mesh with 100.000 unknowns.

Instance	N $(n\times m\times l)$	p	T_{NO}	T_{MP} $(red\%)$	T_{MR} $(red\%)$	T_{OP} $(red\%)$
1	10.000 (100×10×10)	5	23,8	12,5 (47,6%)	16,4 (31,1%)	5,1 (78,6%)
2	10.000 (100×10×10)	10	196,7	185,5 (5,7%)	42,7 (78,3%)	31,4 (84,0%)
3	50.000 (200×50×5)	5	340,6	84,7 (75,1%)	300,7 (11,7%)	45,6 (86,6%)
4	50.000 (200×50×5)	10	1.911,9	1.655,3 (13,4%)	354,9 (81,4%)	100,5 (94,7%)
5	100.000 (100×50×20)	5	1.784,0	255,6 (85,7%)	1.698,8 (4,8%)	178,7 (90,0%)
6	100.000 (100×50×20)	10	6.027,5	4.499,5 (25,4%)	2.255,8 (62,6%)	733,7 (87,8%)
7	300.000 (200×50×30)	5	15.991,3	1.600,4 (90,0%)	15.679,1 (2,0%)	1.344,8 (91,6%)
8	300.000 (200×50×30)	10	29.148,3	14.753,7 (49,4%)	17.522,7 (39,9%)	3.186,3 (89,1%)

Table 1: Set of chosen parameters and computational results.

6 Analysis and Conclusion

Examining tests 1 through 8, we observe a higher level of fill-in induces a higher contribution of the reordering strategy on the final time reduction, while on the lower level the matrix partitioning strategy is responsible for the biggest improvement. Furthermore, by increasing the dimension of the problem but maintaining the same level of fill-in, the reduction achieved by the Jacobian optimization increases while the reduction achieved by the reordering decreases. Nevertheless, the final time reduction is always higher than 78%, showing the strength in the use of the optimization strategies. It is worth mentioning that the final execution time is lower when using ILU(5). The preliminary results obtained in this work show that the optimization strategies imposed on the Jacobian evaluation and the GMRES solver significantly reduced the final execution time for the benchmark 3D nonlinear heat transfer problem.

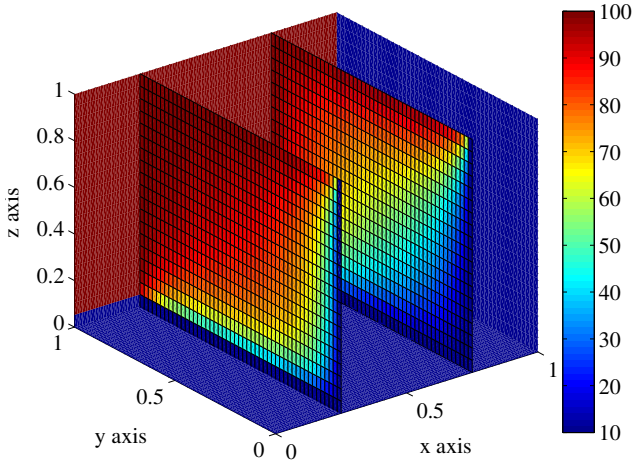


Figure 2: Temperature distribution for a 100.000 mesh.

Acknowledgement

This work was partly supported by FAPES n^o 48511579/2009, CNPq 552630/2011-0 and CNPq 307020/2012-6.

References

[1] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your jacobian? graph coloring for computing derivatives," *SIAM REV*, vol. 47, pp. 629–705, 2005.

[2] D. Goldfarb and P. L. Toint, "Optimal estimation of jacobian and hessian matrices that arise in finite difference calculations," *Mathematics of Computation*, vol. 43, pp. 69–88, 1984.

[3] S. W. Sloan, "An algorithm for profile and wavefront reduction of sparse matrices," *Internacional Journal for Numerical Methods in Engineering*, vol. 23, pp. 239–251, 1986.

LARGE SPARSE MATRIX REORDERING SCHEMES APPLIED TO A FINITE ELEMENT ANALYSIS

Brenno Lugon, Lucia Catabriga
Universidade Federal do Espírito Santo - Brazil
Corresponding Author: **Lucia Catabriga**

brennolugon@gmail.com, luciac@inf.ufes.br

1 Introduction

The finite element method is one of the most used numerical techniques for finding approximated solutions of partial differential equations. Generally, the finite element formulations may require the solution of linear systems of equations involving millions of unknowns that are usually solved by Krylov space iterative update techniques, from which the most used is the Generalized Minimal Residual method (GMRES). In this work, we analyze reordering strategies applied to an incomplete **LU** preconditioner for the resulting system of a SUPG finite element formulation considering two free softwares for unstructured mesh generation. The first one, EASYMESH (`web.mit.edu/easymesh_v1.4/www/easymesh.html`), perform a renumeration of nodes in order to decrease the matrix bandwidth while the second one, GMSH (`geuz.org/gmsh/`), consider a naive numeration.

2 ILU(p) preconditioner and reordering schemes

The preconditioner basic idea is to replace the given system $Ax = b$ by the system $M^{-1}Ax = M^{-1}b$, where M is a "suitable" approximation to A such that $M^{-1}A$ is well conditioned. From a practical point of view, preconditioner operations should be memory-efficient and require few arithmetic operations. Hence a sparse incomplete factorization strategy in the form of $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are the incomplete LU factors might be an appropriate candidate. Incomplete factorization algorithms are sensitive to the ordering of equations, therefore the efficiency of elimination and consequently the construction of an incomplete factorization applied to a preconditioned iterative solver will be influenced by the numeration of the nodal unknowns [1].

There are several approaches to reordering nodal unknowns for efficient linear system solution. These reorderings correspond to row and column matrix interchanges. The goal is to reduce the matrix bandwidth, i.e., reduce the maximum of distances between the first nonzero element of a row i and the main diagonal. This is NP-complete problem and several algorithms exist that are generally able to finding a relatively good solution in a reasonable amount of time. It is well known that reordering techniques are efficient choices when we use incomplete factorization algorithms [2]. However, we want to show how advantageous these reorderings can be in final CPU time when we examine softwares for mesh generation with distinct characteristics for nodal numeration.

3 Test problem

For the experiments, we consider a benchmark problem described by the following convection-diffusion equation:

$$\boldsymbol{\beta}.\nabla u - \nabla.(\boldsymbol{\kappa}\nabla u) = f, \quad \text{in } \Omega = [0,1] \times [0,1] \tag{1}$$

where u represents the quantity being transported (e.g. temperature, concentration). The problem described a pure convection of a scalar on the domain Ω, where the diffusivity is given by $\boldsymbol{\kappa} = 1 \times 10^{-6} I$, the flow direction is 45° from the x-axis, $\|\boldsymbol{\beta}\| = 1$ and the function $f = 0$. Figure 1 shows the problem set up and the boundary conditions and Figure 2 shows the approximated solution.

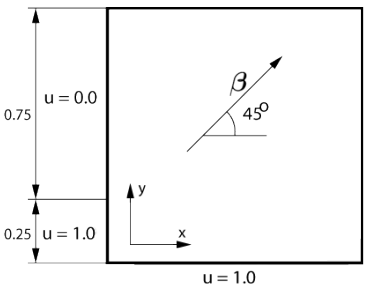


Figure 1: Boundary conditions of the related problem.

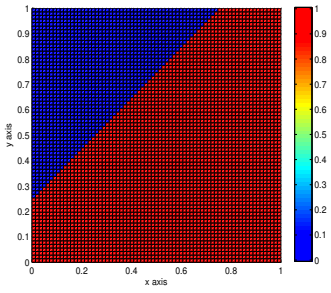


Figure 2: Approximated solution for a 10.000 mesh.

4 Experimental results

All programs were developed by the author in C language and the tests were run on an Intel Core i5 2.53GHz processor with 4GB of RAM, under Ubuntu 12.10. To store the resulting matrix we use the well-known CSR optimized storage scheme. Figures 3 and 4 show the sparsity pattern of resulting matrices derived from GMSH and EASYMESH.

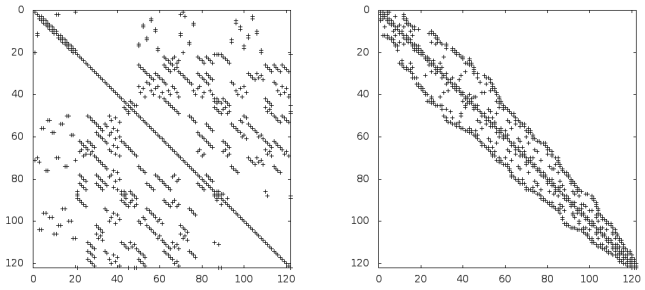


Figure 3: Sparsity pattern of resulting matrices by GMSH
Left: without reordering **Right**: reordering with RCM

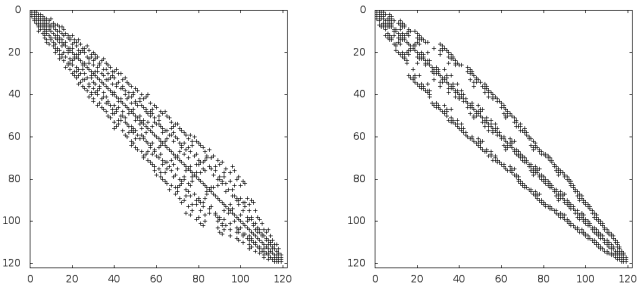


Figure 4: Sparsity pattern of resulting matrices by EASYMESH
Left: without reordering **Right**: reordering with RCM

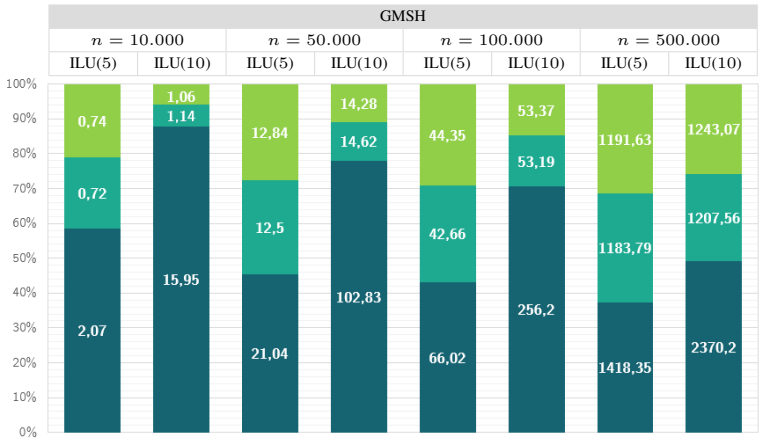


Figure 5: CPU time (seconds) to solve GMSH resulting matrices with no reordering ■, reordering with RCM ■ and with Sloan ■, for different ILU preconditioners.

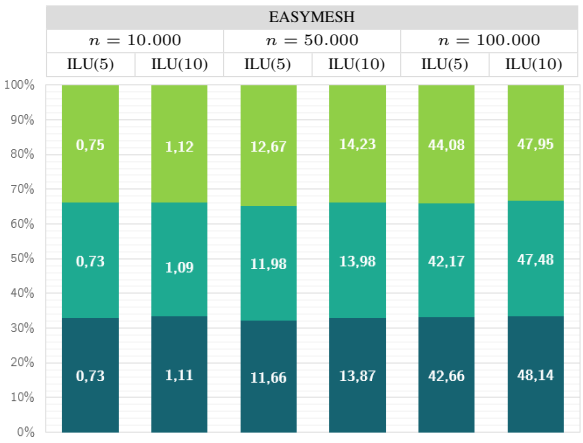


Figure 6: Same as Figure 4, but now considering EASYMESH.

	$n = 10.000$	$n = 50.000$	$n = 100.000$	$n = 500.000$
GMSH	0,65	4,43	10,58	65,23
EASYMESH	47,61	2.813,56	11.266,33	more than 48h

Table 1: CPU time (seconds) to generate meshes with GMSH and EASYMESH for different sizes.

5 Analysis and conclusions

Two reordering algorithms, Reverse Cuthill-McKee (RCM) and Sloan, were used for tests as well as two levels of fill-in for ILU(p) preconditioner. Figure 5 shows us that, for all cases, reordering strategies cause significant gain of CPU time for matrices derived from GMSH. On the other hand, for matrices derived from EASYMESH in Figure 6, the reordering does not cause any positive impact. This fact has to do with the nodal unknowns numeration, i.e., the matrix sparsity pattern, that may cause substantial fill-in during the preconditioner calculations and raise the number of arithmetic operations. Regarding the reordering algorithms, RCM and Sloan had comparative results. If we examine the ILU preconditioner effectiveness in GMSH matrices, we can see in Figure 5 that, as the p parameter increases, higher is the impact of the reordering in the final CPU time. Now, with respect to the mesh generators, Table 1 shows that the computational time to generate meshes were quite different for each software. Meshes generated by EASYMESH demanded much more CPU time than meshes generated by GMSH, which was expected since GMSH does not perform any kind of optimization for numeration nodal unknowns.

We can conclude that it is not necessary to use a reordering if you choose to use a mesh generator that apply an optimization (renumeration of nodes). However, we notice that it is much more advantageous to decide for a software that does not implement any optimization during the mesh generation and make use of reordering strategies during the solution process, since the CPU time to generate an optimized mesh, as does EASYMESH, can be incredibly high.

Acknowledgement

This work was partly supported by FAPES n^o 48511579/2009, CNPq 552630/2011-0 and CNPq 307020/2012-6.

References

[1] J. Camata, A. Rossa, A. Valli, L. Catabriga, G. Carey, and A. Coutinho, "Reordering and incomplete preconditioning in serial and parallel adaptive mesh refinement and coarsening flow solutions," *International Journal for Numerical Methods in Fluids*, vol. 69, no. 4, pp. 802–823, 2012.

[2] M. Benzi, "Preconditioning techniques for large linear systems: A survey," *J. Comput. Phys.*, vol. 182, pp. 418–477, Nov. 2002.

Network Partitioning in Scientific Simulations - A Case Study

Hélène C. Coullon and Rob H. Bisseling

High-level parallel programming, or implicit parallel programming is one of the most important research domains to bring the use of parallel architectures within easy reach for non computer scientists. The large domain of scientific computations has, a priori, no requirement limits for performance and parallelism, and is one of the primary domains that need implicit parallel programming solutions. Among all possible scientific computations, simulations based on *partial differential equations* (PDEs) are particularly time-consuming. In this context, the high-level parallel programming library SkelGIS has been implemented. SkelGIS [2,3] is an implicit parallelism, and a C++ header-only, library to solve mesh-based PDEs in parallel while preserving a sequential programming style.

Numerical methods to solve partial differential equations (PDEs) discretize both time and space to run a simulation on a computer. The discretization of the space domain is called a mesh. Parallelization of a simulation implies the need for a good load balancing of the mesh among the processors, and for a minimization of the communication volume during computations, which could be modeled as a graph partitioning problem. In some specific simulations, as for example in artery blood-flow or river water-flow simulations, a network is created to represent the domain with two different types of elements: nodes and edges. A network could be considered as a general graph where computations are carried out on both edges and nodes and where communications are needed from nodes to edges and from edges to nodes, possibly at different time steps. Thus, parallelization of such applications raises a specific graph partitioning problem, where both edges and nodes handle computations and communications at each time iteration.

Partitioning of simulations with several computation supersteps such as our network simulation can, in principle, be done by invoking a multi-constraint hypergraph partitioner [4] as PaToH for example. Our approach is different: we use the single-constraint partitioner Mondriaan [5] but make sure to satisfy both constraints of load balancing, while minimizing the communication. Two different methods have been explored: the first one, named *single-partitioning method*, is composed of two steps: (1) the communication superstep from nodes to edges is translated to a hypergraph partitioning problem [1], to distribute the edges, and (2) a heuristic is applied to distribute the nodes of the network, taking into account the distribution of the edges; the second one, named *double-partitioning method*, is decomposed in three steps: (1) the communication step from nodes to edges is translated to a hypergraph partitioning problem to distribute the edges, (2) the communication step from edges to nodes is translated to a hypergraph partitioning problem to distribute the nodes, and (3) a permutation problem is solved to match both distributions. We expect performance results for the poster presentation.

References

[1] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE TPDS*, 10(7):673–693, 1999.

[2] H. Coullon, M.-H. Le, and S. Limet. Parallelization of Shallow-Water Equations with the Algorithmic Skeleton Library SkelGIS. In *ICCS*, pages 591–600, 2013.

[3] H. Coullon and S. Limet. Algorithmic skeleton library for scientific simulations: SkelGIS. In *HPCS*, pages 429–436, 2013.

[4] B. Uçar and C. Aykanat. Partitioning sparse matrices for parallel preconditioned iterative methods. *SIAM J. Sci. Comput.*, 29(4):1683–1709, 2007.

[5] B. Vastenhouw and R. H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Rev.*, 47(1):67–95, 2005.

REDUCING ELIMINATION TREE HEIGHT FOR UNSYMMETRIC MATRICES

ENVER KAYAASLAN† AND BORA UÇAR‡

Abstract. The elimination tree for unsymmetric matrices is a recent model playing important roles in sparse LU factorization. This tree captures the dependencies between the tasks of some well-known variants of sparse LU factorization. Therefore, the height of the elimination tree roughly corresponds to the critical path length of the task dependency graph in the corresponding parallel LU methods. We propose algorithms to symmetrically permute the rows and columns of a given unsymmetric matrix so that the height of the elimination tree is reduced, and thus a high degree of parallelism is exposed. The proposed algorithms are obtained by generalizing the most successful approaches used in sparse Cholesky factorization. We test the proposed algorithms on a set of real world matrices and report noticeable reduction in the elimination tree heights with respect to a possible exploitation of the state of the art tools used in Cholesky factorization.

1. Introduction. The standard elimination tree [15] has been used to expose parallelism in sparse Cholesky, LU, and QR factorizations [1, 3, 8, 11]. Roughly, a set of vertices without ancestor/descendant relations corresponds to a set of independent tasks that can be performed in parallel. Therefore, the total number of parallel steps, or the critical path length, is equal to the height of the tree on an unbounded number of processors [12]. Obtaining an elimination tree with the minimum height for a given matrix is NP-complete [14]. Therefore, heuristic approaches are used. One set of heuristic approaches is to content oneself with the graph partitioning based methods. These methods reduce some other important cost metrics in sparse Cholesky factorization, such as the fill-in and the operation count, while giving a shallow depth elimination tree [7]. When the matrix is unsymmetric, the elimination tree for LU factorization [4] would be useful to expose parallelism as well. In this respect, the height of the tree, again, corresponds to the number of parallel steps or the critical path length for certain factorization schemes. In this work, we develop heuristics to reduce the height of elimination trees for unsymmetric matrices. To the best of our knowledge no other work looked at this problem on its own.

Let $\mathbf{A}$ be a square matrix and let $\mathrm{G}(\mathbf{A}) = (\mathrm{V}(\mathbf{A}), \mathrm{E}(\mathbf{A}))$ be the standard directed graph model. The minimum hight of an elimination tree of $\mathbf{A}$ is equivalent to the graph theoretical notion of the cycle-rank of $\mathrm{G}(\mathbf{A})$. Gruber [6] shows that computing the cycle-rank is NP-complete, justifying the need for heuristics for large problems. One reasonable heuristic is to use a graph partitioning tool, such as MeTiS [9], on the symmetrized matrix (we present comparisons with this method). One can also use some local ordering heuristics [2]; but as their analogue for symmetric matrices, these are not expected to be very effective.

2. Methodology. We propose a recursive approach to reorder a given matrix so that the elimination tree is reduced. The main procedure takes an irreducible unsymmetric matrix $\mathbf{A}$ as its input, and produces a permutation yielding an upper bordered block diagonal form, as shown in (2.1)

$$\mathbf{A}_{\mathrm{BBT}} = \mathbf{P}\mathbf{A}\mathbf{P}^T = \begin{bmatrix} \mathrm{A}_{11} & \mathrm{A}_{12} & \dots & \mathrm{A}_{1\mathrm{K}} & \mathrm{A}_{1\mathrm{B}} \\ & \mathrm{A}_{22} & \dots & \mathrm{A}_{2\mathrm{K}} & \mathrm{A}_{2\mathrm{B}} \\ & & \ddots & \vdots & \vdots \\ & & & \mathrm{A}_{\mathrm{KK}} & \mathrm{A}_{\mathrm{KB}} \\ \mathrm{A}_{\mathrm{B}1} & \mathrm{A}_{\mathrm{B}2} & \dots & \mathrm{A}_{\mathrm{BK}} & \mathrm{A}_{\mathrm{BB}} \end{bmatrix}. \tag{2.1}$$

At each recursive call, the procedure partitions the current matrix into a 2×2 block structure where the off-diagonal blocks have only a few nonzeros. Then a covering for each off diagonal blocks is found and the better one is used to have a BBT form with two blocks. Then, the recursion is

†INRIA, France and LIP, ENS Lyon(enver.kayaaslan@ens-lyon.fr).

‡CNRS and Laboratoire de l'Informatique du Parallélisme, (UMR CNRS -ENS Lyon-INRIA-UCBL), Université de Lyon, 46, allée d'Italie, ENS Lyon, F-69364, Lyon Cedex 7, France, bora.ucar@ens-lyon.fr).

matrix	MeTiS	BBT-3	BBT-50	matrix	MeTiS	BBT-3	BBT-50
Averous/epb1	401	325	323	Hohn/fd18	422	307	311
Bai/rw5151	268	168	168	Hohn/sinc12	1836	1206	1181
Goodwin/goodwin	422	369	371	Hollinger/g7jac040	991	762	756
Graham/graham1	549	466	467	Lucifora/cell1	193	125	133
Grund/bayer02	198	123	119	Nasa/barth	142	99	102
Grund/bayer10	211	141	141	Nasa/barth4	133	80	83
Hamrle/Hamrle2	103	67	67	Nasa/barth5	185	117	103
Hohn/fd12	260	199	202	Shen/e40r0100	617	523	597
Hohn/fd15	381	250	251	TOKAMAK/utm5940	448	387	388

TABLE 3.1
Height of the elimination tree on a set of matrices.

applied to each diagonal block. The recursion stops when the current matrix has a size smaller than a parameter (we tested with 3 and 50). Then another heuristic based on feedback vertex sets is used to order the smallest matrices. The overall approach is the unsymmetric analog of pioneering work on extracting vertex separators from edge separators [13].

3. Results. We present results on matrices used in previous study, where the unsymmetric elimination trees were algorithmically studied [5]. Table 3.1 compares MeTiS and the proposed method with the stopping condition of the recursion being 3 and 50, which are shown as BBT-3 and BBT-50. In this table, BBT-3 and BBT-50 obtain tree heights whose geometric mean to the heights obtained by MeTiS is 0.71 and 0.72, where each run is the median of 5 runs (MeTiS has randomization inside).

Given this strikingly good results, we further performed tests with other data. We used the matrices (a subset from [10], where the matrices with a pattern symmetry of a most 0.90 are used, and only two matrices from each group is used). In this data set, the geometric mean of the heights of trees obtained by BBT-50 to MeTiS is found to be 0.91.

The poster presentation will include algorithmic details and further details.

REFERENCES

[1] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez, *Multifrontal QR factorization for multicore architectures over runtime systems*, in Euro-Par 2013 Parallel Processing, F. Wolf, B. Mohr, and D. Mey, eds., vol. 8097 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 521–532.

[2] P. Amestoy, X. Li, and E. Ng, *Diagonal Markowitz scheme with local symmetrization*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 228–244.

[3] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Computer methods in applied mechanics and engineering, 184 (2000), pp. 501–520.

[4] S. C. Eisenstat and J. W. H. Liu, *The theory of elimination trees for sparse unsymmetric matrices*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 686–705.

[5] S. C. Eisenstat and J. W. H. Liu, *Algorithmic aspects of elimination trees for sparse unsymmetric matrices*, SIAM J. Matrix Anal. Appl., 29 (2008), pp. 1363–1381.

[6] H. Gruber, *Digraph complexity measures and applications in formal language theory*, Discrete Mathematics & Theoretical Computer Science, 14 (2012), pp. 189–204.

[7] A. Guermouche, J.-Y. L'Excellent, and G. Utard, *Impact of reordering on the memory of a multifrontal solver*, Parallel Computing, 29 (2003), pp. 1191–1218.

[8] J. Hogg, J. Reid, and J. Scott, *Design of a multicore sparse Cholesky factorization using DAGs*, SIAM Journal on Scientific Computing, 32 (2010), pp. 3627–3649.

[9] G. Karypis and V. Kumar, *A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN, (1998).

[10] K. Kaya and B. Uçar, *Constructing elimination trees for sparse unsymmetric matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 345–354.

[11] J. W. Liu, *Computational models and task scheduling for parallel sparse Cholesky factorization*, Parallel Computing, 3 (1986), pp. 327–342.

[12] ———, *Reordering sparse matrices for parallel elimination*, Parallel Computing, 11 (1989), pp. 73 – 91.

[13] J. W. H. Liu, *A graph partitioning algorithm by node separators*, ACM Trans. Math. Softw., 15 (1989), pp. 198–219.

[14] A. Pothen, *The complexity of optimal elimination trees*, Tech. Rep. CS-88-13, Pennsylvania State Univ., 1988.

[15] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Transactions on Mathematical Software, 8 (1982), pp. 256–276.

ON NEWTON'S METHOD FOR AN INVERSE FIRST PASSAGE PROBLEM

YINGJUN DENG, ANNE BARROS AND ANTOINE GRALL

In this poster, we will introduce the inverse first passage problem (IFPT) for an Ornstein-Uhlenbeck process and some numerical problems in our work. The IFPT problem is targeted to reproduce the corresponding boundary for a given stochastic process such that the first passage density can fit a given distribution function. This problem attracts plenty of attention recently in various applications such as financial risk management, reliability analysis [1] etc..

Cheng etc. [2] have proved the IFPT problem is well-posed, but some numerical problems remain in the solving procedures of IFPT problems. Following the integral equation method proposed in [3] and [4], the IFPT problem can be translated to an integral equation:

$$f(x,t) = 0, \forall x \geq L(t), \tag{0.1}$$

where $L(t)$ is the desired true solution and $L(0)$ is given. This property proposes a constraint for the numerical scheme to verify whether the solution is the minimal solution to the equation.

In the solving procedure, we tried the simplest secant method because $f_x(x,t)$ cannot be expressed for $x < L(t)$. An interesting thing is that although $f_x(x,t)$ cannot be provided, the left-side derivative at true solution can be given from preliminary analysis:

$$\lim_{x \to L(t)-} f_x(x,t) = g(t) > 0, \tag{0.2}$$

which is independent of $L(t)$.

How to maximally use this information for accelerating the numerical scheme remains a problem for us. The ordinary Newton's method requires the derivative information at every iterative value, and therefore it is hard to be used directly in this case. We are interested to try various searching methods in this problem to increase the numerical efficiency.

REFERENCES

1. Yingjun Deng, Anne Barros, and Antoine Grall, *Calculation of failure level based on inverse first passage problem*, Proceedings of Annual Reliability & Maintainability Symposium, 2014.
2. Xinfu Chen, Lan Cheng, John Chadam, and David Saunders, *Existence and uniqueness of solutions to the inverse boundary crossing problem for diffusions*, Annals of Applied Probability **21** (2011), no. 5, 1663–1693.
3. Goran Peskir, *On integral equations arising in the first-passage problem for brownian motion*, Journal of Integral Equations and Applications **14** (2002), no. 4, 397–423.
4. Cristina Zucca and Laura Sacerdote, *On the inverse first-passage-time problem for a Wiener process*, Annals of Applied Probability **19** (2009), no. 4, 1319–1346.

INSTITUT CHARLES DELAUNAY, UMR CNRS 6279& UNIVERSITY OF TECHNOLOGY OF TROYES, BP 2060 - 10010 TROYES CEDEX, FRANCE.
E-mail address: yingjun.deng@utt.fr

Key words and phrases. inverse first passage time, Newton's method.