RWTH Aachen

Department of Computer Science
Technical Report

Fifth SIAM Workshop on Combinatorial Scientific Computing, May 19–21, 2011, Darmstadt, Germany

Markus Beckers, Johannes Lotz, Viktor Mosenkis and Uwe Naumann (Editors)

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2011-09

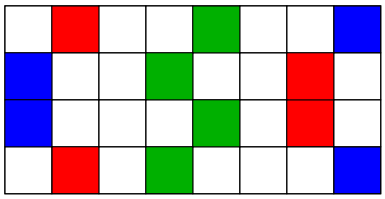
RWTH Aachen · Department of Computer Science · October 2011

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Fifth SIAM Workshop on Combinatorial Scientific Computing

May 19–21, 2011, Darmstadt, Germany



Abstracts

Markus Beckers, Johannes Lotz, Viktor Mosenkis, and Uwe Naumann (Editors)
LuFG Informatik 12: Software and Tools for Computational Engineering
RWTH Aachen University, Germany

The human desire for meaningful numerical simulation of physical, chemical, and biological phenomena in science and engineering has been increasing with the growing performance of the continuously improving computer systems. As a result of this development, we are (and will always be) faced with a large (and growing) number of highly complex numerical simulation codes that run at the limit of the available high-performance computing resources. These codes often result from the discretization of systems of differential equations. The runtime correlates with the resolution that often needs to be very high in order to capture the real behavior of the underlying system. There is no doubt that the available hardware will always be used to the extreme. Improvements in the runtime of the simulations need to be sought through research in numerical algorithms and their efficient implementation on parallel architectures.

Many of the resulting problems are combinatorial in nature. Most of those are known to be computationally hard in the sense that efficient (polynomial in the required time and memory space) algorithms for their exact solution are unlikely to exist. For example, we have to deal with partitioning, elimination ordering, coloring, and matching problems for graphs and hypergraphs in various contexts. A good approximation of the solution to these abstract problems may lead to a significant decrease in the runtime of numerical programs that implement solvers for partial differential equations, nonlinear optimization algorithms, or solvers for generalized Eigenvalue problems. Problem sizes are typically now in the millions of unknowns; and with emerging large-scale computing systems, this size is expected to increase by a factor of thousand over the next five years. Moreover, simulations are increasingly used in design optimization and parameter identification which is even more complex and requires the highest possible computational performance and fundamental enabling algorithmic technology.

What binds together the community of combinatorial scientific computing is the focus on practical use of graph algorithms and combinatorial algorithms to address a variety of different problems that all arise in scientific computing. This shared common denominator allows us to interact productively and to advance the state of the art in several different problem areas.

The Fifth SIAM Workshop on Combinatorial Scientific Computing represents another important milestone. Three CSC experts have been invited to present plenary talks on various important aspects of CSC:

- Thomas F. Coleman (University of Waterloo, Canada): *Efficient Automatic Differentiation for Nonlinear Systems and Continuous Optimization (by using graphs)*; joint with SIAM Workshop on Optimization;
- Burkhard Monien (Paderborn University, Germany): *Recent Trends in Graph Partitioning for Scientific Computing*;
- Trond Steihaug (Bergen University, Norway): *Sparse Matrix Structures and Higher Derivatives.*

This collection of extended abstracts covers all accepted contributed oral and poster presentations. It is meant to give both participants of CSC11 and other interested readers an overview of recent results and ongoing research and development projects in the area of Combinatorial Scientific Computing.

Contents

1 Computational Challenges in Optimization for Electrical Grid Operations and Planning

Richard Li-Yang Chen, Genetha A. Gray, Patricia D. Hough, Ali Pinar, John Siirola, and Jean-Paul Watson
(*Sandia National Labs, United States)*

Corresponding Author: **Genetha Gray** (gagray@sandia.gov)

In the United States, the electrical grid has changed very little in the last 100 years, and in fact still works mainly on the principle of one-way flow of consumption. The drivers for the new smart grid include new energy generation methods, balanced loads and reduced peaking, improved reliability and security, and the desire for a two-way flow system. To address these goals, solutions to some fundamental short-term operations and long-term planning problems in power systems are needed. Almost all of these problems have a combinatorial component such as adding lines to a network, finding cuts, making assignments to nodes of a graph. However, we need to solve these problems with uncertain information, thus we use stochastic mixed-integer optimization for modeling these problems, which represent a significant challenge since even with major advances in algorithms over the last decade, their size and complexity makes them intractable without the extreme scale computational power. Therefore, this work is based on the belief that the new high performance computing technologies will radically change the landscape of electric power systems, both due to widely-available parallelism in many-core systems, and the extreme-scale computing platforms. In this presentation, we will give a basic overview of electrical grid operations and planning, point to some specific areas of research, and describe some results obtained for a couple of core optimization problems, unit commitment and network expansion.

One of the most basic problems in electrical grid planning and operations is unit commitment. The unit commitment problem is to determine an optimal on/off schedule for a set of power generating units that both meets load demands and satisfies operational constraints. Unit-commitment is considered for both short (e.g., hours or days) and long (e.g., weeks or months) time horizons. Uncertainties resulting from load forecasts, network outages, and discrete events must be considered in order to make a robust unit commitment decision. As an example, we will consider unit commitment with uncertainty due to wind speeds or solar availabilities. This is a fundamental power systems operational problem that can be formulated as a mixed integer stochastic optimization problem. Another such problem in electrical grid planning and operations is network expansion. Here, the focus is determining how best to upgrade the system in order to meet future demands. Like unit commitment, network expansion can be posed as a mixed integer stochastic optimization problem. In planning contexts, like all generation and transmission expansion problems, inherent uncertainties in future demand, budgets, and technologies add to the challenge of obtaining a robust solution.

The computational challenges associated with solving stochastic mixed-integer problems like the unit commitment and the network expansion problems are significant. There are two primary and related factors. First, the (finite) number of scenarios required to approximate the joint distributions of uncertain parameters leads to notoriously difficult deterministic mixed-integer optimization problems. Second, unrealistic modeling simplifications are often required to achieve computationally tractability, leading to more costly and potentially infeasible solutions. High-performance computing technologies have been proposed to mitigate both concerns. In this presentation, we will review these technologies and present some results for their application to the electrical grid planning and operation problems.

Decomposition strategies have been developed to deal with the computational intractability of finite-sample approximations of the underlying stochastic optimization problems. These include the well-known L-Shaped (or Benders), Progressive Hedging, Dual Decomposition, and Dantzig-Wolfe algorithms. These algorithms operate by sub-dividing the finite-sample deterministic optimization problem into a number of sub-problems, which are then solved iteratively to obtain globally optimal or near-optimal solutions. Typically, the sub-problems can be solved in parallel, leading to obvious deployment strategies on high-performance computing platforms. However, a number of critical issues arise when even modest (e.g., 100s) of scenarios are considered. For example, sub-problem solve time variability leads significant drops in parallel efficiency, leading to the need for asynchronous versions of these algorithms. Similarly, certain schemes scale better than others due to further imbalances in sub-problem difficulty as the algorithm proceeds, e.g., as occurs in the L-shaped method. We survey prior work in this area, outline the challenges and key future research directions.

Modeling simplifications to obtain computational tractability are wide-ranging in the grid optimization literature. For example, unit commitment is often solved without transmission constraints, and generation and transmission expansion are often treated independently. Similarly, security (i.e., "n-1") constraints are often ignored, or potential deployment functionality such as transmission switching are avoided. Alternatively, the choice of performance metric can have a significant bearing on computational tractability in stochastic optimization. Of particular note is the impact of risk-oriented metrics such as conditional value-at-risk or risk-adverse concepts such chance constraints have on computationally difficult. High-performance computing holds significant potential for solving these more complex and realistic deterministic optimization problems, either as sub-problems of a stochastic optimization problem or in isolation. In particular, parallel branch-and-cut algorithms for mixed-integer programs can yield tractable run-times for such problems. We explore the applicability of parallel branch-and-cut algorithms for solving complex grid optimization problems, considering both commercial and open-source solvers.

Finally, we note that both problems are particularly difficult in a security-conscious, cost-constrained world where there is a need to quantify and enumerate the tradeoff between uncertain risks (e.g., grid stability, renewable penetration, security threats,) and the costs to mitigate those risks. This need motivates the development of multi-objective optimization of stochastic systems, but it also introduces a number of new challenges including quantification of the tradeoff curve itself, managing the potentially astronomical computational

budget and educating decision makers in how to interpret and use the results. In this talk, we will further describe these challenges and present some preliminary ideas for addressing them.

2 Detailed Aerodynamic Shape Optimization Based on an Adjoint Method with Shape Derivatives

Caslav Ilic
(*German Aerospace Center Braunschweig, Germany)*

Stephan Schmidt and Volker Schulz
(*University of Trier, Germany)*

Nicolas Gauger
(*RWTH Aachen University, Germany)*

Corresponding Author: **Caslav Ilic** (caslav.ilic@dlr.de)

Aerodynamic Optimization

A modern commercial passenger (or cargo) airplane is a highly efficient transportation system, in terms of how much fuel it consumes per passenger (or unit payload) per unit distance travelled. The increase in efficiency must continue, however, due to the limited amount of natural fuel sources and the desire to have more environment-friendly ("greener") airplanes. This calls for introduction of powerful numerical means of airplane *design optimization*, in the disciplines of aerodynamics, structure, acoustics, and many more.

One of the main drivers of efficiency is the external shape of the airplane. The shape is progressively refined through several design phases. In the final, *detailed design* phase, high-fidelity CFD models with millions of state variables are used to simulate the performance of the airplane, and thousands of design parameters may be available to define its aerodynamic shape. At the moment, the only practical optimization method at this scale of complexity is the *adjoint-based gradient descent.*

Optimization Methodology

Gradient descent optimization methods use the gradient information to quickly converge the objective function to its minimum. In each optimization cycle, the gradient information is used to determine the descent direction, and then a step of certain length is taken into that direction, producing an updated set of design parameters. This is repeated until the convergence criteria is reached.

We parametrize the aerodynamic shape by simply associating a direction to each node on the corresponding boundaries in the computational mesh, and then moving the node along that direction ("free-node" parametrization). While this provides maximum degree of freedom for the shape, it also results in very high number of design parameters – from hundreds in 2D cases, to tens of thousands in 3D cases. This approach remains feasible by proper choice of elements of the optimization algorithm.

We compute the gradient using the *adjoint* method. In this way, the gradient can be theoretically computed with the effort roughly equal to that for the

flow simulation (the primal problem), independently of the number of design parameters. In practice this is frequently not attained, because of the additional need to compute sensitivities of the flow residual to mesh deformation (e.g. using costly finite differencing). Instead, we apply *shape differentiation* to derive continuous surface formulation of the gradient, which can be evaluated in single pass over the flow domain boundaries, achieving independence of the number of design parameters.

With the gradient in hand, we apply a *one-shot reduced SQP* method to compute the descent direction taking constraints into account. One-shot means that only a small number of simulation steps are made between optimization cycles, instead of fully converging the flow and adjoint solutions in each cycle. The Hessian approximation for the SQP is taken to be the discrete *Laplace-Beltrami operator* on the discretized shape. Aside from the primary effect of accelerating the convergence (preconditioning), this choice of Hessian has the geometric smoothing effect on the shape.

Free-node parametrization has natural multi-level character, whereby an indicator-based adaptation (e.g. residual, or dual-weighted residual) of the mesh automatically produces coarser or finer parametrizations as well. We use this to demonstrate that optimization can be further sped up through a multi-level approach.

The primal and adjoint flow solutions are computed by the DLR Tau solver. The flow is modelled using Euler equations, with continuous adjoint formulation.

Results and Outlook

The outlined method has been successfully applied to several 2D and 3D cases in transonic flow, with the goal of reducing (or completely removing) shock waves, which give rise to the *wave drag*. This component of the drag has high dependency on the shape of the airplane (wing, fuselage), and if not catered for, it becomes the primary cause of loss of efficiency and performance. In all cases, we used as constraints the lift and the enclosed volume of the shape.

In 2D, we optimized an RAE 2822 airfoil at Mach number 0.73 and angle of attack 2 deg, achieving full removal of shocks. The number of design parameters was 510. The optimization runtime was tenth of that of the classical adjoint approach, which used finite differencing for the flow residual sensitivities and smooth parameterization of 32 Hicks-Henne bump functions.

In 3D, we optimized an Onera M6 wing at Mach number 0.83 and angle of attack 3 deg. The lambda-shock that was formed on the upper surface was completely removed at the end of optimization. The number of design parameters was 18,000. We also optimized a blended wing-body configuration VELA, at Mach 0.85 and AoA 1.8 deg, with 110,000 design parameters. Shocks were not completely removed in this case, but they were sufficiently weakened to reduce the drag by 28%.

Current work focuses on extending the method to Navier-Stokes flow. The crucial element is construction of shape derivatives, which becomes very tedious when both viscosity and compressibility are taken into account. It is particularly hard on implementation, since there are no "physical checkpoints" which could be used to verify implementation part by part. To that end, we work on constructing a limited automatic symbolic derivator, relying on the rules of shape derivation. It would take as input a set of symbolic s-expressions of the

governing PDEs and the objective function, and produce as output a set of C or Fortran expressions for computing the gradient.

3 Computing Strong Bounds in Combinatorial Optimization

Hans Mittelmann
(*Arizona State University, United States)*

Corresponding Author: **Hans Mittelmann** (mittelmann@asu.edu)

As is well-known semidefinite relaxations of discrete optimization problems can yield excellent bounds on their solutions. We present three examples from our collaborative research. The first addresses the quadratic assignment problem and a formulation is developed which yields the strongest lower bounds known for larger dimensions. Utilizing the latest iterative SDP solver and ideas from verified computing a realistic problem from communications is solved for dimensions up to 512. A strategy based on the Lovasz theta function is generalized to compute upper bounds on the spherical kissing number utilizing SDP relaxations. Multiple precision SDP solvers are needed and improvements on known results for kissing numbers in dimensions up to 23 are obtained. Generalizing ideas of Lex Schrijver improved upper bounds for general binary codes are obtained in many cases.

In our earlier work [1, 2] we had developed a series of SDP relaxations for the QAP which lead to improved lower bounds for a number of the QAPLIB instances of medium and, in particular, large sizes. The QAPs considered included those ssociated with Hamming and Manhattan distances as well as more general QAPs. While some of the QAPLIB instances model real-life problems, such as the (small) NUG series, there are well-known QAP applications that lead to rapidly increasing QAPs, for example, in the area of communications. The associated distance is Hamming and thus the size 2^d with d up to 9. Our approach in [1, 2] would have been too costly for the largest cases and we utilized an iterative SDP solver in order to reduce both memory requirements and CPU time. In [4] we were then able to solve the case $d = 8$ for the index assignment problem in case of 1-bit errors. An important enhancement of the methods in [1, 2] was applied which then in [6] also allowed to solve the case d=9 with both one-bit and multiple bit errors.

Other available methods to compute lower bounds for the QAPs produce either poor bounds cheaply or excellent bounds at such a cost that a size of 50 is already beyond a reasonable computational effort, see [4, 6]. In order to provide feasible solutions and an upper bound we also ran a number of heuristics and report results for the so-called iterated local search.

Closely related to the optimal index assignment problem in communications is the fundamental question how many binary words of length exist that have a mutual minimum Hamming distance d. In communications a subset of this set called the code book is used for coding. In earlier work by Lex Schrijver important improvements had been obtained over the well-known bound due to Delsarte's SDP-based approach. In particular he had considered n-tuples of words with $n = 3$. The resulting SDPs were large but had symmetry that could be exploited to make their solution manageable.

In the work [5] quadruples of words were used resulting in even larger but highly symmetric SDPs. The new problem that arose on the computational side was that the problems were so ill-conditioned that double precision with all available SDP solvers did not produce acceptable accuracy. It became necessary to apply multiple precision codes (both SDPA and CSDP were used in quadruple versions, SDPA also in some cases in octuple precision). This did not only lenghten the required CPU times substantially but also required some subtle parameter choices in case of SDPA not to have the computations terminate prematurely. There are no known strategies guaranteeing convergence.

In a majority of the cases tabulated we were able to improve the known upper bounds, see [5].

Different from the work on binary codes our work on the spherical kissing number [3] lead to improvements of all known (upper) bounds as tabulated, for example, by Wikipedia. The problem of placing spheres of a given radius around a central sphere was already a point of contention between Isaac Newton and David Gregory more than 300 years ago. To settle the issue in three dimensions in Newton's favor took until 1953. Because of its relevance for spherical codes the kissing number is also of interest in higher dimensions. Based on earlier work by C. Bachoc and F. Vallentin, together with the latter, we applied SDP relaxations, essentially computing the Lovasz theta number as a bound for the stability number of a certain (packing) graph, As for the binary codes the SDPs were large which did not cause a problem but they were also ill-conditioned and multiple precision computations were needed. Not only were all known bounds improved but also an outstanding conjecture proven due to Conway and Sloane.

All papers listed below are accessible through `plato.asu.edu/papers.html`.

Bibliography

[1] H.D. Mittelmann, J. Peng, *Estimating Bounds for Quadratic Assignment Problems Associated with the Hamming and Manhattan Distance Matrices based on Semidefinite Programming*, SIAM J. Optim. 20, 3408-3426 (2010)

[2] J. Peng, H. D. Mittelmann, X. Li, *A New Relaxation Framework for Quadratic Assignment Problems based on Matrix Splitting*, Math. Prog. Comp. 2, 59-77 (2010)

[3] H. D. Mittelmann and F. Vallentin, *High Accuracy Semidefinite Programming Bounds for Kissing Numbers*, Exper. Math. 19, 174-179 (2010)

[4] X, Wu, H. D. Mittelmann, X. Wang, and J. Wang, *On Computation of Performance Bounds of Optimal Index Assignment*, in proceedings of 2010 Data Compression Conference

[5] D. C. Gijswijt, H. D. Mittelmann, and A. Schrijver, *Semidefinite code bounds based on quadruple distances*, submitted to IEEE Trans. on Information Theory

[6] X. Wu, H. D. Mittelmann, X. Wang, and J. Wang, *On Computation of Performance Bounds of Optimal Index Assignment*, submitted to IEEE Trans. on Communications

4 Clustering via optimization

Suely Oliveira and David E. Stewart
(*University of Iowa, United States)*

Corresponding Author: **David Stewart** (dstewart@math.uiowa.edu)

In this abstract the problem of unsupervised clustering of data is considered. This is the problem of assigning data items to one of a small number of clusters of related data items. Here we will consider the data items to be vectors of numbers of consistent dimension. These vectors can be generated from actual data items, such as text documents, images, or other measurements and observations by various means according to the kind of data treated. The different components of these vectors are often scaled and/or shifted so as to make, for example, the mean of each component equal to zero, and the standard deviation of each component equal to one.

From data in this form we can obtain measures of similarity and difference between the data items using the distance between vectors: $d_{ij} = \phi(\|v_i - v_j\|)$ where v_i is the vector representing data item i. Typically $\phi(s) = s$ or $\phi(s) = s^2$. Representation of the problem of clustering as an optimization problem is quite old. For example, there is the formulation of Rao [3] for N data items and M clusters:

$$\min_x \quad \sum_{i,j=1}^{N} d_{ij} \sum_{k=1}^{M} x_{ik}x_{jk} \qquad \text{subject to} \tag{1}$$

$$\sum_{k=1}^{M} x_{ik} = 1 \qquad \text{for all } i, \tag{2}$$

$$\sum_{i=1}^{N} x_{ik} \geq 1 \qquad \text{for all } k, \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for all } i \text{ and } k. \tag{4}$$

The decision variables are the x_{ij} where $x_{ij} = 1$ means that data item i is in cluster j, and zero otherwise. The objective function (1) $\sum_{i,j=1}^{N} d_{ij} \sum_{k=1}^{M} x_{ik}x_{jk}$ is the sum of distances d_{ij} where i and j are in the same cluster. Equation (2) indicates that every data item is in one and only one cluster. Inequality (3) indicates that each cluster contains at least one data item.

The problem (1–4) is a combinatorially hard optimization problem, belonging to the family of quadratic binary programs. Recent work has focused on new ways of representing and approximating such problems (see, for example, [1]), especially using the convex cone of symmetric copositive matrices. Unfortunately, determining if a symmetric matrix is copositive is NP-hard [2]. This has led to a great deal of work on approximating this cone by means of semi-definite matrices [2].

A number of direct SDP relaxations of (1–4) have been proposed. One of

these is

$$\min_{X,\,Y} D \bullet Y \qquad \text{subject to} \tag{5}$$

$$Xe = e, \qquad X \geq 0, \tag{6}$$

$$\begin{bmatrix} Y & X \\ X^T & I \end{bmatrix} \succeq 0. \tag{7}$$

This has dropped the explicit constraint (3). However, direct SDP relaxations of (1–4) have a common problem. If X is a feasible matrix for the original problem (1–4), then so is XP for any permutation matrix P, and furthermore, the objective function is the same. This simply expresses the fact that changing the labels of the clusters does not change either the objective function or feasibility. Since SDP's are convex optimization problems, and algorithms for SDP's tend to converge to the centroid of the solution set, the computed solutions will have this same symmetry. This means that the (computed) solution X^* will have all columns the same, and so $X^* = ee^T/M$, which gives no information about how to cluster the data.

We have been working on a non-convex continuous relaxation of (1–4) which uses a number of ideas from the SDP relaxations. However, we retain $D \bullet XX^T$ in the objective function rather than $D \bullet Y$ with $Y \succeq XX^T$ as in the SDP relaxation. These relaxations have a parameter $\alpha \geq 0$ that can be interpreted in terms of statistical information theory, or as a convexification parameter. It can also be used to control the number of clusters found. Effective methods for finding local minima have been used. These methods are scalable to large-scale data bases with thousands to millions of data items.

Bibliography

[1] Samuel Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Math. Program.*, 120(2, Ser. A):479–495, 2009.

[2] P. Parrilo. *Structured Semidenite Programs and Semi-algebraic Geometry Methods in Robustness and Optimization.* PhD thesis, 2000.

[3] M. R. Rao. Cluster analysis and mathematical programming. *Journal of the American Statistical Association*, 66(335):622–626, 1971.

5 Multithreaded Algorithms for Graph Coloring

Assefaw Gebremedhin and Alex Pothen
(*Purdue University, United States)*

Umit Catalyurek
(*The Ohio State University, United States)*

John Feo and Mahantesh Halappanavar
(*Pacific Northwest National Laboratory, United States)*

Corresponding Author: **Assefaw Gebremedhin** (agebreme@purdue.edu)

We present a set of efficient multicore and massively multithreaded algorithms for a prototypical graph problem, graph coloring. The algorithms are implemented—and shown to perform and scale well—on a collection of platforms with varying degrees of multithreading capabilities. The platforms considered include a 128-processor Cray XMT, a 16-core Sun Niagara 2, and an 8-core Intel Nehalem system. We find that obtaining good performance on these machines involves designing algorithms that pay careful attention to and take advantage of the programming abstractions and hardware features the machines provide. The resultant algorithms are different from algorithms that do well on earlier machines that support shared memory and distributed memory programming models.

Graph coloring is an abstraction for partitioning a set of binary-related objects into subsets of independent objects. A need for such a partitioning arises in situations where there is a scarce resource that needs to be utilized optimally. One example is in discovering concurrency in parallel computing. Graph coloring is known to be NP-hard to solve optimally; in fact it is known to be NP-hard to approximate to within $O(n^{1-\epsilon})$ for all $\epsilon > 0$, where n is the number of vertices in the graph [3]. Despite such inapproximability results, however, greedy coloring algorithms that employ good ordering techniques yield near optimal solutions on graphs that arise in practice [2].

Graph algorithms have a number of well recognized features that make them particularly challenging to parallelize with emphasis on performance and scalability: Runtime is dominated by memory latency rather than processor speed; there exist little computation to hide memory access costs; data locality is poor; and available concurrency is low. For these reasons, there are few graph algorithms that perform and scale well on distributed memory machines.

Researchers have had more success on shared-memory platforms, and interest in these platforms is growing with the increasing abundance and popularity of multicore architectures. The primary mechanism for tolerating memory latencies on most shared memory systems is the use of *caches*, but caches have been found to be rather ineffective for many graph algorithms. A more effective mechanism is *multithreading*. By maintaining multiple threads per core and switching among them in the event of a long latency operation, a multithreaded processor uses parallelism to hide latencies. Unlike caches, which

"hide" only memory latencies, thread parallelism can hide both memory and synchronization overheads. Thus, multithreaded, shared-memory systems are more suitable platforms for graph algorithms than either distributed memory machines or single-threaded, multicore shared-memory systems.

In this presentation, we discuss primarily two different parallel distance-1 coloring algorithms we developed for shared-memory, multithreaded systems. The first algorithm relies on *speculation* and *iteration*, and is suitable for any shared-memory system, including multicore platforms. The algorithm is derived from the parallelization framework for coloring on distributed-memory architectures developed in [1]. We benchmarked the algorithm on the Cray XMT, Intel Nehalem, and Sun Niagara 2 systems mentioned earlier. These systems represent a broad spectrum of multithreading capabilities and memory structure: the Cray XMT has a flat, cache-less memory system and utilizes massive multithreading (128 threads per processor) as the sole mechanism for tolerating latencies in data access, the Intel Nehalem relies primarily on a cache-based hierarchical memory system as a means for hiding latencies and supports only two threads per processor, and the Sun Niagara 2 offers a middle path by utilizing a moderate number of hardware-threads along with a hierarchical memory system. We found that the limited parallelism and coarse synchronization of the iterative algorithm fit well with the limited multithreading capabilities of the Sun and Intel processors.

The iterative algorithm ran equally well on the Cray XMT, but it does not take advantage of the system's massively multithreaded processor and hardware support for fast synchronization. To better exploit the XMT's unique hardware features, we developed a fine-grained, *dataflow* algorithm requiring single word synchronization, which is the second algorithm we discuss in this presentation. This XMT-tailored algorithm achieves shorter runtime and uses fewer colors than (the generic) iterative algorithm when run on the XMT. The iterative algorithm has the attractive feature of being portable on different architectures.

We assess the scalability and performance of both algorithms using a set of massive synthetic graphs carefully designed to include instances that test-stress the algorithms. We show that the dataflow algorithm scales well (nearly ideally for certain classes of graphs) on the XMT. The iterative algorithm scales in a similar fashion on all three platforms considered, with increasing relative performance on platforms with greater thread concurrency. Further, the number of colors used by the parallel algorithms is fairly close to what the sequential algorithm uses. In turn, the number of colors the sequential algorithm uses is only a small factor of the optimal (we were able to determine the factory by computing an appropriate lower bound). Hence, there is negligible loss in quality of solution due to parallelization.

In addition to the results on distance-1 coloring, we will briefly discuss results from ongoing work on related topics: parallelization of ordering techniques for reducing the number of colors required; and parallelization of algorithms for other coloring problems needed in the context of automatic differentiation.

Bibliography

[1] Doruk Bozdağ, Assefaw Gebremedhin, Fredrik Manne, Erik Boman, and Umit Catalyurek. A framework for scalable greedy coloring on distributed-

memory parallel computers. *Journal of Parallel and Distributed Computing*, 68(4):515–535, 2008.

[2] Assefaw Gebremedhin, Duc Nguyen, Alex Pothen, and Mostofa Patwary. ColPack: Graph coloring software for derivative computation and beyond. Submitted to ACM TOMS, 2010.

[3] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.

6 Fast Conservative Estimation of Hessian Sparsity

Ebadollah Varnik, Lukas Razik, Viktor Mosenkis, and Uwe Naumann
(*RWTH Aachen University, Germany)*

Corresponding Author: **Ebadollah Varnik** (varnik@stce.rwth-aachen.de)

Background

We propose a method in context of *Algorithmic Differentiation* (AD) [2] for *conservative estimation* of the sparsity pattern of the Hessian $\mathbb{R}^{m\times n\times n} \ni \nabla^2 F = \nabla^2 F(\mathbf{x}) \equiv (f''_{k,j,i})^{k=1,\ldots,m}_{i,j=1,\ldots,n}$ of vector functions $\mathbf{y} = F(\mathbf{x})$ given as computer programs in C and C++ that map a vector $\mathbb{R}^n \ni \mathbf{x} \equiv (x_i)_{i=1,\ldots,n}$ of input values onto a vector $\mathbb{R}^m \ni \mathbf{y} \equiv (y_j)_{j=1,\ldots,m}$ of output values. We assume that F can be decomposed into the single assignment code (SAC) at every point of interest as

$$\begin{aligned} x_i &\equiv v_i && \text{for } i = 1,\ldots,n \\ v_k &= \varphi_k(v_i)_{i\prec k} && \text{for } k = n+1,\ldots,q \\ y_j &\equiv v_{j+n+p} && \text{for } j = 1,\ldots,m \end{aligned}$$

where $q = n + p + m$ and $i \prec k$ denotes a direct dependence of v_k on v_i meaning that v_i is an argument of the *elemental function* $\varphi_k \in \Phi = \{+, -, *, /, \sin, \cos, \cdots\}$. The transitive closure of this relation is denoted by $\prec^+$. Moreover, we assume F to be *canonical* in the sense that no algebraic simplifications such as $\log(e^x) = x$ are possible. Furthermore, for mathematical correctness of the derivative calculus F is assumed to be two times continuously differentiable in some neighborhood of the given argument $\mathbf{x}$. Conceptually, the runtime algorithm for *exact* Hessian sparsity estimation (EHP) works on the SAC of F by propagating first- and second-order dependencies

$$\begin{aligned} fod(v_k) &= \{i \mid \exists\, \mathbf{x} \in \mathbb{R}^n : \frac{\partial v_k}{\partial x_i}(\mathbf{x}) \neq 0\} \quad \text{and} \\ sod(v_k) &= \{(i,j) \mid \exists\, \mathbf{x} \in \mathbb{R}^n : \frac{\partial^2 v_k}{\partial x_i \partial x_j}(\mathbf{x}) \neq 0\} \end{aligned}$$

of every SAC variable v_k on inputs $i, j \in X = \{1,\ldots,n\}$ with $k \in V = \{1,\ldots,q\}$. Thereby, we distinguish between linear and nonlinear type of elemental functions for $\varphi_k \in \Phi_l = \{+, -\}$ and $\varphi_k \in \Phi_n = \{*, \sin, \cos, \ldots\}$, respectively. As shown by Walther [3] for $m = 1$ EHP is of complexity $O(\tilde{n}^2)\cdot OPS(F)$, where $OPS(F)$ and $\tilde{n}$ denote the operation count of F and the maximal number of nonzeros per row in $\nabla^2 F$, respectively. It has to be mentioned here that the quadratic complexity of EHP is caused by the computation of sod_k as cross product of fod_i and union of sod_i of their arguments $i \prec k$ in case of linear and nonlinear operations, respectively. In the following section we introduce the conservative algorithm (CHP) and compare its runtime with EHP.

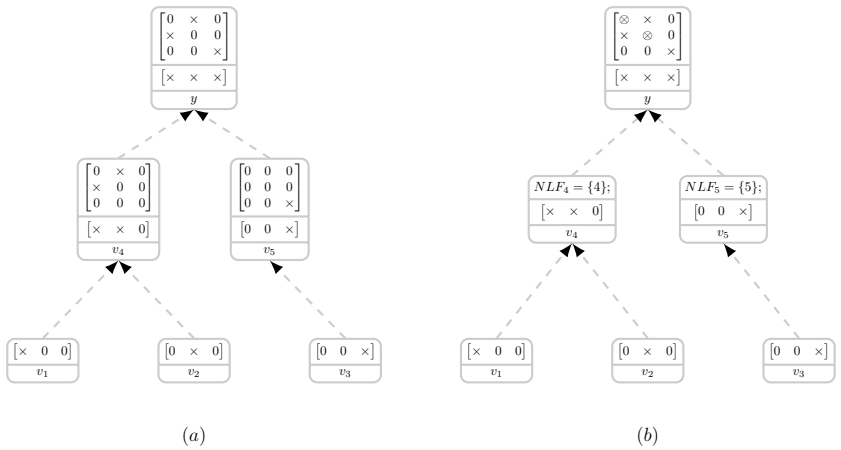


Figure 1: Exact (a) and overestimate (b) computation of the Hessian pattern. Exact and overestimated nonzeros are denoted by symbols $\times$ and $\otimes$, respectively.

Conservative Sparsity Detection

We assume in the following that F is partially separable as $F(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x})$ into nonlinear functions f_i, which we refer to as *nonlinear frontier* (NLF) components of F. Thus, differentiating F with respect to $\mathbf{x}$ yields $\nabla^2 F = \sum_{i=1}^{N} \nabla^2 f_i(\mathbf{x})$. Thus, the exact and conservative sparsity pattern of $\nabla^2 F$ is given by

$$sod(\mathbf{y}) = \bigcup_{i=1}^{N} sod(f_i) \quad \text{and}$$
$$csod(\mathbf{y}) = \bigcup_{i=1}^{N} csod(f_i)$$

, respectively. Thereby,

$$csod(f_i) = fod(f_i)^2 = fod(f_i) \times fod(f_i)$$

denotes the *conservative second-order dependencies* of f_i on $\mathbf{x}$ with $sod(f_i) \subseteq csod(f_i)$. Thus, we can overestimate the sparsity pattern of $\nabla^2 F$ first by computing $fod(f_i)$ of all NLF components f_i followed by building a union of the cross products $fod(f_i)^2$.

Algorithm 1 illustrates the computation of $csod(v_j)$ with $j \in Y$ of the outputs $\mathbf{y}$ on the SAC of F. Thereby, in addition to the computation of $fod(v_k)$ with $k \in V$, we propagate the NLF set defined as $nlf(v_k) = \begin{cases} \bigcup_{i \prec k} nlf(v_i) & \varphi_k \in \Phi_l \\ \{k\} & \varphi_k \in \Phi_n \\ \emptyset & \text{otherwise.} \end{cases}$

Obviously, the NLF of a linear operation results from the union of the NLF of its arguments as shown in line 7, whereas the NLF of a nonlinear operation is given by itself as shown in line 10. We emphasize here that the computationally expensive cross products along with their unions as shown in line 12 are

1: **for** $i = 1$ **to** n **do**
2: $fod_i = \{i\}$
3: $nlf_i = \emptyset$
4: **end for**
5: **for** $k = n+1, \ldots q$ **do**
6: **if** $\varphi_j \in \Phi_l$ **then**
7: $fod_k = \bigcup_{i \prec k} fod_i$
8: $nlf_k = \bigcup_{i \prec k} nlf_i$
9: **end if**
10: **if** $\varphi_j \in \Phi_n$ **then**
11: $fod_k = \bigcup_{i \prec k} fod_i$
12: $nlf_k = \{k\}$
13: **end if**
14: **end for**
15: **for** $j = n+p+1$ **to** q **do**
16: $csod_j = \bigcup_{i \in nlf_j} fod_i \times fod_i$
17: **end for**

Algorithm 1: [**CHP**]

performed only for output variables in the number of their NLF components. We can show that

$$OPS(CHP) \leq N \cdot O(\tilde{n}^2) + OPS(F) \cdot O(\tilde{n}) \ ,$$

where $N = |\bigcup_{j \in Y} nlf_j|$ and $\tilde{n}$ denotes the maximal number of nonzeros per row in $\nabla^2 F$. FIG. 1 (a) [(b)] demonstrates on the following example SAC

$$\begin{aligned} &v_1 = x_1; \quad v_2 = x_2; \quad v_3 = x_3; \\ &v_4 = v_1 \cdot v_2; \quad v_5 = v_3^2; \quad y = v_4 + v_5; \end{aligned}$$

the computation of fod and sod $[nlf]$[1]. The exact second-order pattern sod_4 of v_4 results from the cross product of $fod(v_1)$ and $fod(v_2)$, where the multiplication operation is marked as the NLF component of v_4 by $nf(v_4) = \{4\}$ as shown in (b). Thus, we get

$$\begin{aligned} csod(y) &= fod(v_4) \times fod(v_4) \cup fod_5 \times fod_5 \\ &= \{(1,1),(1,2),(2,1),(2,2)\} \cup \{(3,3)\} \end{aligned}$$

with $fod(v_4) = \{1,2\}$ and $fod(v_5) = \{3\}$ as the conservative second-order dependency of y on $\mathbf{x} = (x_1, x_2, x_3)$.

Numerical Results

We present first performance results in FIG. 2 of the conservative algorithm CHP. Thereby, we compare the runtime of CHP with EHP implemented in AD tool ADOL-C on Hessian matrices of an objective function that arise in Simulated

[1] fod, nlf and sod, $csod$ are denoted as vectors and matrices of dimension 3 and (3×3), respectively.

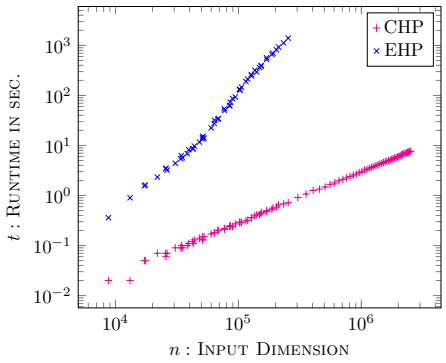


Figure 2: Runtimes of CHP and EHP on SMB Code.

Moving Bed (SMB) process a model for liquid chromatographic separation described by Assefaw et al. [1]. We observe here a linear growth in input dimension n of CHP, whereas EHP tends to increase quadratically with n. We observed also that the resulting compression rate based on the conservative sparsity pattern is not worse even to some extent better than the one gained on the exact one, on which we expect to report results in more detail at the time of the workshop.

Bibliography

[1] Assefaw H. Gebremedhin, Alex Pothen, and Andrea Walther. Exploiting sparsity in Jacobian computation via coloring and automatic differentiation: A case study in a simulated moving bed process. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, number 64 in LNCSE, pages 327–338. Springer, Berlin, 2008.

[2] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition*. Society for Industrial and Applied Mathematics (SIAM), 2008.

[3] Andrea Walther. Computing sparse Hessians with automatic differentiation. *ACM Transaction on Mathematical Software*, 34(1):3:1–3:15, 2008.

7 Enabling Non-Graph-Expert Use of Very-large-scale Graph Analysis

John Gilbert and Adam Lugowski
(*University of California, Santa Barbara, United States)*
Aydin Buluc
(*Lawrence Berkeley National Laboratory, United States)*
Steve Reinhardt
(*Microsoft Corporation, United States)*

Corresponding Author: **Steve Reinhardt**
(Steve.Reinhardt@microsoft.com)

Introduction

Deep analysis of very large graphs is vital for practitioners in several disciplines, yet the means of analyzing graphs in distributed memory remain a focus of intense research by algorithm implementers. The absence of an interface to current graph-analysis packages for non-graph-expert users means that most potential users do not benefit from the current packages and do not give feedback to algorithm implementers on the (non)suitability of particular algorithms. We built a Python interface of graph abstractions to the Combinatorial BLAS, a state-of-the-art package known to have excellent performance and scalability. We programmed the Graph500 benchmark with only a few calls to this Knowledge Discovery Toolbox (KDT) interface with strong performance and are implementing further applications to expand KDT's coverage. While our implementation uses the Combinatorial BLAS, the interface could be implemented on top of other infrastructure, such as Parallel Boost Graph Library [8], Small-world Network Analysis and Partitioning [2] or MultiThreaded Graph Library [3]. We propose the KDT interface as a starting point for a community-accepted interface, enabling non-graph-expert domain experts to analyze large graphs now while permitting algorithm developers to develop better implementations with a proven avenue to end users.

Background

The Graph Analysis and Pattern Detection Toolbox

In 2005, Gilbert, Shah, and Reinhardt [6] implemented a prototype Graph Analysis and Pattern Discovery Toolbox in the M language of MATLAB^{TM} that provided graph abstractions in terms of the distributed sparse matrices of Star-P [5], enabling graph analysis without awareness of how sparse matrices are used to solve graph problems. Shah used this prototype [7] to implement the graph analysis benchmark of the Synthetic Scalable Compact Application benchmarks [10]. This prototype proved that graph abstractions in very-high-level languages are usable by non-graph-experts and can solve real problems with strong performance.

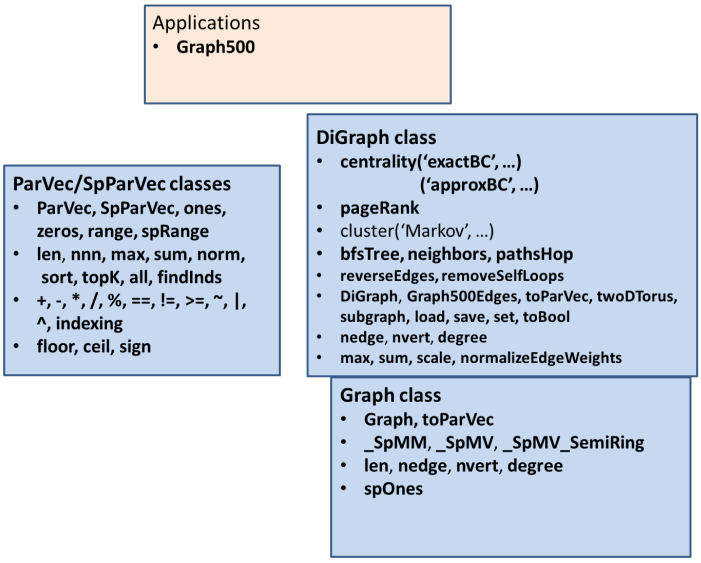


Figure 1: Current and Near-Future KDT Contents

The Combinatorial BLAS

In 2009, Buluc and Gilbert developed the Combinatorial BLAS [4] as a scalable high-performance library enabling graph analysis and data mining. In addition to its focus on the sparse-matrix functionality needed to solve graph problems, it extended previous distributed-sparse-matrix implementations by supporting 2-dimensional distributions and delivering unprecedented performance on distributed-memory clusters.

Current Work

Knowledge Discovery Toolbox via Python

With the lessons of the Graph Analysis and Pattern Discovery Toolbox in hand and the Combinatorial BLAS available as a portable, high-performance infrastructure, building a widely usable graph-analysis package was clearly practical. We have created Knowledge Discovery Toolbox (KDT) interface in Python, initially a serial version based on SciPy, but more importantly a distributed-memory version based on the Combinatorial BLAS.

KDT currently includes the classes and methods shown in Figure 1 in **bold** type, with near-future extensions shown in normal type. The data-parallel interface creates and uses objects solely via the provided (distributed) classes, as the size of distributed objects makes them unlikely to fit in the memory of a single cluster node.

Given the Combinatorial BLAS functionality, KDT methods were straightforward to implement. For example, the bfsTree method is implemented in about 20 lines of Python code. The exact/approximate betweenness centrality method is about 110 lines including docstrings and took 11 hours to port from the serial SciPy version. This bodes well for adding more functionality to KDT

and for evolving its functionality to respond to user needs. Our focus for KDT is on implementing higher-level algorithms such as centrality and clustering, as those are the functions we believe non-graph-experts want. We envision bfsTree being used primarily as a building block for other higher-level functions.

KDT is hosted at `kdt.sourceforge.net` and released under the New BSD open-source license.

Future work

Our current work has focused on solving a few sample directed-graph problems with robust performance for distributed memory. An important next step is to broaden the set of problems that are addressable by KDT, and so we are implementing more applications, which we expect will point out missing functionality or improvements to existing functionality. We expect to extend KDT to bipartite graphs, attributes as needed for semantic graphs, and spectral methods next.

Another important class of potential users is those solving graph problems with disk-based infrastructure such as Hadoop [1] or Dryad [9]. Implementing KDT based on such infrastructure will enable those users to solve their problems via the same graph abstractions, and potentially open up possibilities for mixed in-memory/on-disk problem-solving.

Summary

KDT is intended for users who need the benefit of deep graph analysis without deep understanding of the algorithms and their implementation. The initial KDT distributed-memory implementation proves the usefulness of the approach both in terms of ease of use and performance. We will extend it for more applications and hone the essential set of kernels.

Bibliography

[1] D. Cutting A. Bialecki, M. Cafarella. Hadoop: a framework for running applications on large clusters built of commodity hardware, 2005. http://lucene.apache.org/hadoop.

[2] David A. Bader and Kamesh Madduri. SNAP, Small-world Network Analysis and Partitioning: An open-source parallel graph framework for the exploration of large-scale networks. In *IPDPS*, pages 1–12, 2008.

[3] Jonathan W. Berry, Bruce Hendrickson, Simon Kahan, and Petr Konecny. Software and algorithms for graph queries on multithreaded architectures. *Parallel and Distributed Processing Symposium, International*, 0:495, 2007.

[4] Aydın Buluç and John R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. Technical Report UCSB-CS-2010-18, UCSB Computer Science Department, Oct 2010. Submitted to International Journal of High Performance Computing Applications (IJHPCA).

[5] A. Edelman. The Star-P high performance computing platform. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–1197 –IV–1200, April 2007.

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

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

[8] D. Gregor, N. Edmonds, B. Barrett, and A. Lumsdaine. The parallel boost graph library, 2005. http://www.osl.iu.edu/research/pbgl.

[9] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[10] J. Kepner, D. P. Koester. et al. Hpcs scalable synthetic compact application (ssca) benchmarks, 2004. http://www.highproductivity.org/SSCABmks.htm.

8 Scalable Parallel Solution Techniques for Data-Intensive Problems in Distributed Memory

Nicholas Edmonds and Andrew Lumsdaine
(*Indiana University, United States)*

Corresponding Author: **Nicholas Edmonds** (ngedmond@cs.indiana.edu)

Computational science and engineering has traditionally been comprised of linear algebra problems induced by large systems of PDEs. These problems often possess underlying natural locality which can be determined analytically, often via static analysis. An increasingly important class of data-intensive applications is emerging that lacks this natural, or domain-induced, locality. Rather than being analytically deducible, the dependency structure of this class of computations is determined by the input data itself. This data-carried dependency structure is expressed at run time and offers limited opportunities for static analysis. These computations benefit from a variety of fine-grained and dynamic solution techniques well suited to the irregular and non-local nature of these problems.

Introduction

Traditional compute-intensive applications, such as those based on discretized systems of PDEs, possess natural locality due to the local nature of the underlying operators. This natural locality allows the dependency structure of the computation to be largely determined before any portion of the computation occurs. Because locality can be determined analytically at compile-time, the granularity of the application can be coarsened through static analysis and the grouping of computations with shared data dependencies. This class of problems often possesses good separators in the dependency graph which allows for straightforward parallelization using SPMD techniques such as the coarse-grained BSP [4] "compute-communicate" model.

Data-intensive applications, such as those based on graphs differ from compute-intensive applications in a number of important ways. Rather than possessing natural, domain- or operator-induced locality these computations are highly **non-local**. The dependency structure of the computation is **irregular** and embedded in the input data itself. This means that rather than having dependency information available at compile time and being able to perform static analysis and optimization on it, the dependency structure of the computation is discovered **dynamically** at run time. Finally, because dependency information is discovered at run-time, work grouping or coalescing cannot be performed at compile-time and thus the *natural* granularity of the computation remains fine-grained.

Expressing Fine-Grained Applications

Coarse-grained BSP solutions often group operations which do not share computational dependencies in order to coarsen the granularity of an application and

achieve better performance on real hardware. This approach is reasonable in applications with a dependency structure which can be determined analytically. In data-intensive applications with run-time-discovered dependency structures this approach can artificially extend the critical path of the application however. It is thus desirable to express applications at the finest level of granularity possible. Coarsening may still be necessary at run-time, but by providing the run-time mechanism that performs this coarsening with the full, fine-grained dependency graph of the application we enable it to make the most effective decisions about how to perform this coarsening. In order to expose maximum concurrency, expressing applications as independent, asynchronous collections of (possibly dependent) tasks has proven to be an effective programming style. Maintaining the consistency of application metadata in the presence of asynchronous concurrent tasks is necessary if the tasks are to be treated as independent operations. The methods by which data consistency is enforced are crucial to the performance of this model. This requirement for complex, atomic metadata manipulation precludes the use of most "one-sided" or passive-target RMA approaches but can be incorporated into Active Message-based solutions [5].

Run-Time Support

Executing fine-grained applications at their natural level of granularity on modern HPC hardware has a number of problems. HPC hardware has largely been designed to solve traditional coarse-grained, compute-intensive problems. One of the most important manifestations of these design goals with respect to fine-grained applications is the fact that modern high-performance networks become injection-rate limited before they become bandwidth limited in the presence of large numbers of small messages. Secondly, in data-intensive applications where the dependency graph has no good separators, communication patterns are likely to be dense. In compute-intensive applications the number of peers a single processing element (PE) communicates with is often constant or logarithmic due to the natural locality of the underlying operators. In data-intensive applications the number of peers can easily be linear in the number of PEs. While modern networks support collective all-to-all communication relatively well, irregular exchange of small messages between $\mathcal{O}(P^2)$ pairs of PEs is less well supported for large P and incurs significant per-connection overhead to manage the P^2 connections.

The aforementioned limitations of modern HPC networks imply that in order to efficiently map fine-grained expressions of data-intensive problems to current hardware, a run-time layer which can dynamically transform the application expression to an efficient implementation is required. Performing software routing on a sparser virtual topology allows the number of peers each PE communicates with to be reduced from $\mathcal{O}(P)$ to $\mathcal{O}(\log P)$ or even $\mathcal{O}(1)$. Coalescing small messages into larger messages which better exploit network bandwidth increases latency but may yield better performance in bandwidth-bound regions of the application.

Applications

We now demonstrate the application of the programming and execution model (briefly) described above, to an application within the data-intensive domain.

Breadth-first search (BFS) is a simple graph kernel which produces a breadth-first numbering of the vertices of the graph starting with a source vertex s. We assume a vertex distribution of the adjacency list (row-wise distribution of the adjacency matrix).

```
Input: Vertex s, neighbors(v) a function returning the neighboring
       vertices of v, Q_0, Q_1 distributed queues
Output: d[v] the bfs distance of v from s
∀ v ∈ V: d[v] = ∞;
d[s] ← 0;
Q_0, Q_1 ← ∅;
enqueue(Q_0, {s, 0});
while Q_0 ≠ ∅ do
    while Q_0 ≠ ∅ do
        {u, dist} ← dequeue(Q_0);
        if dist < d[u] then
            d[u] = dist;
            foreach v ∈ neighbors(u) do
                enqueue(Q_1, {v, dist + 1});
            end
        end
    end
    barrier;
    /* Communicate non-local elements of Q_1                    */
    swap(Q_0, Q_1);
end
```

Algorithm 1: Coarse-grained BFS

In a traditional SPMD or BSP-style implementation such as Algorithm 1: we expand one level of the BFS (line 6), then communicate all the vertices in the next level (line 13), and repeat until the next level is empty (line 5). This approach has the effect of producing large blocks of data to communicate collectively, but fails to capture the fine-grained dependency structure of the application and thus does not express the full concurrency available.

```
Input: v the vertex to discover, d_v the tentative distance of v from s,
       neighbors(v) a function returning the neighboring vertices of v
if d_v < d[v] then
    d[v] ← d_v;
    foreach u ∈ neighbors[v] do
        discover(u, d_v + 1);
    end
end
```

Algorithm 2: `discover`(v, d_v) – active message handler for vertex discovery

In the fine-grained expression of the algorithm the core operation, discovering new vertices (Algorithm 2), is expressed as an independent asynchronous operation which may be invoked locally or remotely, and may have many instances executing concurrently. The main body of the algorithm (Algorithm 3) simply discovers the source vertex (line 3) and waits for the chain of executions

Input: Vertex s
Output: d[v] the distance of v from s
$\forall\, v \in V$: d[v] $= \infty$;
`begin_epoch`();
`discover`($s, 0$);
`end_epoch`();

Algorithm 3: Computing a BFS tree using active messages.

of Algorithm 2 this triggers to complete (line 4). There are a number of assumptions implicit in this formulation of BFS. First, because the same method is invoked to discover a vertex regardless of whether the vertex is local or remote with regard to the discovering process, some method of resolving vertices to address spaces must exist. As previously discussed, for performance reasons it is also assumed that an underlying run-time layer coalesces these individual method calls bound for remote processors into larger messages sufficient to overcome the injection-rate limitations of the network, and optionally routes them through a virtual topology to minimize connection overhead.

Results

We have employed the techniques described above to implement a generalized active message library (AM++ [6]) capable of performing the runtime transformations described. Comparing the performance of traditional BSP-style formulations of graph algorithms in older versions of the Parallel Boost Graph Library [1] with asynchronous formulations of the same algorithms implemented using AM++ demonstrates markedly improved performance (see Figure 1). Additionally, by expressing algorithms as collections of asynchronous, concurrent tasks we simplify the task of leveraging thread-level parallelism as well as process-level parallelism.

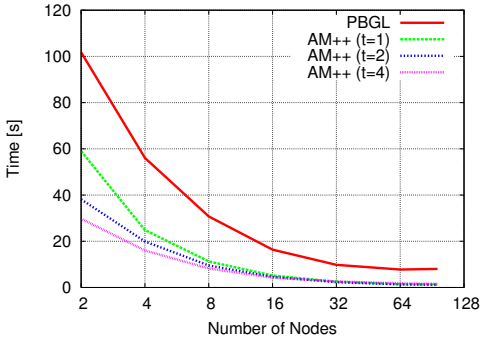


Figure 1: Strong scaling performance of Parallel BGL and AM++-based implementations of parallel BFS (2^{27} vertices and 2^{29} edges).

Bibliography

[1] Douglas Gregor, Nick Edmonds, Alex Breuer, Peter Gottschling, Brian Barrett, and Andrew Lumsdaine. The Parallel Boost Graph Library. `http://www.osl.iu.edu/research/pbgl`, 2005.

[2] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. *SIGARCH Computer Architecture News*, 21(2):289–300, 1993.

[3] Nir Shavit and Dan Touitou. Software transactional memory. In *Principles of Distributed Computing*, pages 204–213. ACM, 1995.

[4] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[5] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active Messages: A mechanism for integrated communication and computation. In *International Symposium on Computer Architecture*, pages 256–266, 1992.

[6] Jeremiah Willcock, Torsten Hoefler, Nicholas Edmonds, and Andrew Lumsdaine. AM++: A generalized active message framework. In *Parallel Architectures and Compilation Techniques*, September 2010.

9 Hypergraph Partitioning for Computing Matrix Powers

Erin Carson, Nicholas Knight, and James Demmel
(*UC Berkeley, United States)*

Corresponding Author: **Erin Carson** (ecc2z@eecs.berkeley.edu)

Motivation Krylov Subspace Methods (KSMs) are a class of iterative algorithms commonly used in scientific applications for solving linear systems, eigenvalue problems, singular value problems, and least squares. Standard KSMs are communication-bound, due to a sparse matrix vector multiplication (SpMV) in each iteration. This motivated the formulation of *Communication-Avoiding* KSMs, which remove the communication bottleneck to increase performance. A successful strategy for avoiding communication in KSMs uses a matrix powers kernel that exploits locality in the graph of the system matrix A. The matrix powers kernel computes k basis vectors for a Krylov subspace (i.e., $\mathcal{K}_k(A, v) = span\{v, Av, ..., A^{k-1}v\}$) reading A only once. Since a standard KSM reads A once per iteration, this approach effectively reduces the communication cost by a factor of k [7, 8].

The current implementation of the matrix powers kernel [8] partitions the matrix A given the computed dependencies using graph partitioning of $A + A^T$. However, the graph model inaccurately represents the communication volume in SpMV and is difficult to extend to the case of nonsymmetric matrices. A hypergraph model remedies these two problems for SpMV [2, 5, 3]. The fundamental similarity between SpMV and the matrix powers kernel motivates our decision to pursue a hypergraph communication model.

Contribution We construct a hypergraph that encodes the matrix powers communication, and prove that a partition of this hypergraph corresponds exactly to the communication required when using the given partition on the rows of A. Although the hypergraph construction represents an additional preprocessing cost, such a cost is justified when the resulting partition requires significantly less communication than a partition obtained using a naive or graph partitioning approach. We evaluate this tradeoff for various classes of matrices, and provide guidance in selecting the optimal partitioning method in terms of preprocessing cost and communication required in the resulting partition. Additionally, we evaluate the effectiveness of various heuristics and sparsification strategies for reducing the cost of constructing and partitioning the hypergraph for matrix powers.

Using the PaToH hypergraph partitioning software [4] and matrices from the U.F. Collection [6], we achieve up to an 80% reduction in total (expand) communication volume over the graph partitioning approach, the largest improvements for structurally unsymmetric matrices (e.g., west1505). In general, the matrix powers kernel shows the most benefit on well-structured (bounded

aspect ratio) matrices. We demonstate that our techniques will improve matrix powers performance for classes of graphs such as physical meshes that are structurally unsymmetric.

A Hypergraph Model for Matrix Powers Our formulation is based on the column-net model, previously described in literature for SpMV [2, 5, 3]. The column-net model was chosen because the matrix powers kernel requires a rowwise partition, ensuring no communication occurs during the computation. We first demonstrate correctness of this formulation for $y = A^k \cdot x$ where $k = 1$, and then consider the case where $k > 1$. The column-net model still holds for the computation of $y = A^k \cdot x$, provided that we find the column-nets for matrix A^k.

However, matrix powers does not simply calculate an SpMV for some power of A. In fact, matrix powers performs an SpMV for all powers of A up to k, and then returns the vectors $[x, Ax, ..., A^k x]$. In general, each power of A expresses a different set of dependencies so it is insufficient to evaluate the dependencies for A^k and claim that those encompass all communication in matrix powers. We show how to combine the dependencies for matrix powers A^1 through A^k in a way to account for all dependencies. We perform this combination using a union operation, since once a dependency has been accounted for, the communicated value is available locally (without subsequent communication) for all iterations. These new nets, which we call *k-level column-nets*, represent the cumulative dependencies.

We form a hypergraph H_k with the same vertices as H, above, but now use the k-level column-nets. We have encapsulated dependencies by nets in H_k so that the cost of a P-way partition in H_k tells us the communication that will occur in the execution of the matrix powers kernel. It is worth noting that minimizing communication in the matrix powers kernel is NP-Hard since the transformation $[A, k, x] \rightarrow H_k$ can be performed in polynomial number of steps, giving a polynomial-time reduction from minimizing communication in the matrix powers kernel to finding the minimum cost cut of H_k.

Using Heuristics to Reduce Preprocessing Cost Constructing the full k-level column nets adds a significant cost to the preprocessing algorithm, especially for large values of k. We evaluate the effectiveness of the following strategies (and combinations of these strategies) for reducing the algorithmic cost of both constructing and partitioning our hypergraph for matrix powers:

- Sparsification of the input matrix
- Dropping large nets from consideration during partitioning
- Approximating k-level column-nets with $k = 1$ + iterative swapping [9]

Current Work Current work involves considering the costs of moving the redundant entries of the input vector x as well as the rows of matrix A independently. That is, our previous formulation assumes a unit cost for each data dependency, when it would be more accurate to weight the k-level dependencies with unit cost $w_i = 1$, and the 1- through $(k-1)$-level dependencies by a weight $w_i = 1 + nnz_i$, where nnz_i is the number of nonzeros in matrix row

A_i. This would be formulated as a multi-constraint hypergraph partitioning problem, discussed in [1] in the context of SpMV.

Additionally, ongoing work involves exploring the effectiveness of such a strict communication-avoiding approach in practice. The communication costs between cores are far less expensive than, for example, between nodes in a network. Other partitioning schemes, like the 2D decompositions evaluated in [5], would require communication at each step of the matrix powers kernel (since columns are partitioned as well), but have greater flexibility in reducing the overall volume of communication. We believe this flexibility may play a role in an optimal organization of the algorithm in a shared-memory environment, and would also provide robustness in pathological cases like a dense row in A.

Bibliography

[1] C. Aykanat, B.B. Cambazoglu, and B. Ucar. Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, 2008.

[2] U.V. Catalyurek and C. Aykanat. *Decomposing irregularly sparse matrices for parallel matrix-vector multiplication, In Parallel Algorithms for Irregularly Structured Problems. Lecture Notes in Computer Science, Vol. 1117/1996*, pages 75–86. Springer Berlin / Heidelberg, 1996.

[3] U.V. Catalyurek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse- matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.

[4] U.V. Catalyurek and C. Aykanat. Patoh: A multilevel hypergraph partitioning tool, version 3.0. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey., 1999.

[5] U.V. Catalyurek, C. Aykanat, and B. Ucar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM J. Sci. Comput.*, 32(2):656–683, 2010.

[6] T. Davis. University of florida sparse matrix collection. NA Digest, 92, 1994.

[7] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in computing krylov subspaces. Technical Report No. UCB/EECS-2007-123, 2007.

[8] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, 2009.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partition. *Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

10 Toward architecture aware graph partitioning

Andreas Stathopoulos
(*College of William and Mary, United States)*

Costas Bekas and Alessandro Curioni
(*IBM Research, Zurich, Switzerland)*

Corresponding Author: **Andreas Stathopoulos** (andreas@cs.wm.edu)

Graph partitioning is critical for good performance in many fields, from laying out the circuits on a chip to efficiently parallelizing scientific applications. Substantial research effort has been invested in graph partitioning over the last two decades, which has resulted in a much better theoretical understanding of the problem and several good quality practical tools. The solutions are typically based on spectral relaxations of the discrete problem, multilevel algorithms, or combinations of the two.

In this talk, we focus mainly on applications where a graph is to be partitioned over a very large number of processors in a way that balances computation and minimizes interpocessor communication. Generalizations to other frameworks such as data mining are straightforward but not explored. The interprocessor communication to be minimized is traditionally modeled as the edge-cut between the partitioned domains. However, in this model there is no distinction between domains and processors. Thus, the edge-cut does not take into account that different processor pairs may have different communication costs since the communication network may involve several physical hops, different or congested hardware links, or a complete heterogeneous infrastructure as is the case in cloud computing.

There have been several attempts in the literature to extend partitioning models to include information from the parallel architecture. Hendrickson and Leland introduced a multilevel spectral model for the hypercube network topology. Together with Driessche, they later extended this to other network topologies using domain biases. These methods have been implemented in the Chaco software. Another effort is by Pellegrini and collaborators where they consider an architectural graph to guide the assignment of domains to processors during a multilevel partitioning algorithm. These ideas are implemented in the (PT)-SCOTCH partitioning tool where a choice of several typical network topologies is offered. However, neither of these efforts have resolved satisfactorily the problem as many researchers have noted over the years. Recently, there has been renewed interest in this architecture aware partitioning but most efforts have been adhoc and geared toward special applications.

One of the most successful tools for graph partitioning is the Metis software by Karypis and Kumar. However, Metis tackles mainly the edge-cut problem and does not involve information about the communication network. A couple of years ago, Moulitsas and Karypis attempted to correct this by providing a general architecture-aware objective function which they approximately minimized in a specific predictor-corrector framework. The predictor step consisted

of a usual Metis application, while the corrector step applied a classic randomized refinement based on the architecture-aware objective. They reported very promising results but also mentioned that a more thorough approach would involve minimizing directly the architecture-aware objective function during the multiple levels of Metis. This is the approach we follow in this research.

Starting from the sequential Metis code that also optimizes the number of connected components, we first note that the coarsening phase should remain unchanged as it deals with the connectivity of the application graph and not its assignment to processors. At the coarsest level, Metis generates p domains. At that time we must solve the Quadratic Assignment Problem (QAP): Given a communication cost function $C(p_1, p_2)$ for any pairs of processors, and a weighted adjacency matrix E for the p domains, assign the p domains onto p processors so that the total volume of network-aware communication times is minimized, i.e., $\min \sum_{i,j} C(i,j)E(\pi(i), \pi(j))$ over all permutations π. QAP is a particularly difficult NP-hard problem which resists most efficient continuous relaxations. The architecture-aware partitioning problem, therefore, is at least as hard as QAP; With one exception: QAP depends on p the number of processors, not on $|G|$ the size of the graph. When p is not too large, efficient simulated annealing methods can produce fairly good assignments. We have used such a newly developed method to solve the first processor assignment at the coarsest level.

During every level of the refinement phase, Metis moves graph hypernodes between domains to reduce the edge-cut while satisfying a few other constraints. We have changed this phase of Metis to minimize the network-aware volume of communication time. The goal is to steer the refinement heuristics toward picking closest communicating network pairs and implicitly minimizing the number of neighboring domains. To implement this objective function without increasing the overall complexity of the algorithm, we follow the same boundary strategy as Metis, i.e., boundary nodes participate in the relaxation exchange between domains if they can potentially decrease their edge-cut. Once considered in the boundary, however, they perform the moves that minimize the architecture aware time. One can argue that with a good initial QAP assignment, the boundary is similar, but what matters is how the relaxation works.

In this talk, we discuss further the new heuristics for the implementing architecture-aware graph partitioning and we report results from sequential runs for a variety of graphs on several network topologies and heterogeneous networks.

11 On hypergraph partitioning based sparse matrix ordering

Iain S. Duff
(*Atlas Centre, RAL, United Kingdom)*

Bora Uçar
(*CNRS and ENS Lyon, France)*

Corresponding Author: **Bora Uçar** (bora.ucar@ens-lyon.fr)

We are interested in two classical and related problems in sparse matrix factorization. The first one is to reduce the number of nonzeros in the **R** factor of the **QR** decomposition of a rectangular sparse matrix. The second one is to reduce the number of nonzeros in the Cholesky factor of a given sparse symmetric positive definite matrix. For the first one, we compare two known alternatives from the literature where there is no previous comparison study. For the second one, we investigate a known result on pattern-wise decomposition of sparse matrices, generalize the result and develop algorithmic tools to obtain new ordering methods.

Fill-reducing ordering for QR

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a sparse rectangular matrix with a full column rank. The **QR** factorization of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix, and $\mathbf{R} \in \mathbb{R}^{m \times n}$ is a sparse upper trapezoidal matrix. The number of nonzeros in $\mathbf{R}$ depends on the column permutation of $\mathbf{A}$ (due to the equivalence of the thin QR factorization of $\mathbf{A}$ and the Cholesky factorization of $\mathbf{A}^T\mathbf{A}$ [7, Theorem 5.2.2]).

A desirable form of $\mathbf{A}^T\mathbf{A}$ which leads to a reduced number of nonzeros in its Cholesky factorization is the doubly bordered block diagonal form. When one has small border size, the problem reduces to ordering rows/columns while respecting the block structure. In order to have a sparse $\mathbf{R}$ in the **QR** decomposition one can therefore compute the pattern of $\mathbf{A}^T\mathbf{A}$, order it and apply the resulting ordering to the columns of $\mathbf{A}$. As is known, $\mathbf{A}^T\mathbf{A}$ can be much denser than $\mathbf{A}$ and therefore this approach is avoided. There are methods that work only on $\mathbf{A}$ (for a concise discussion see [4, Chapter 5]). We compare two such methods from the literature: the first is the COLAMD [5] algorithm. This algorithm performs an approximate minimum degree ordering on $\mathbf{A}^T\mathbf{A}$ without forming the product. In the second method, $\mathbf{A}$ is permuted into the singly bordered block diagonal form (diagonal blocks are not necessarily square) by a hypergraph partitioning method [1]. One has to order the columns by respecting the singly bordered structure. We combine the constrained COLAMD with PaToH [2] to accomplish this task. We are not aware of any previous comparisons between these two known alternatives. We perform comparisons to validate the hypergraph partitioning-based approach.

For the initial comparisons, we use PaToH as a black box, and with a post process we make sure that the diagonal blocks in $\mathbf{A}$ have more rows than

columns. Technically we do not need a good balance among the sizes of the blocks but when PaToH is used as a black box, it obtains such partitions. Even so, among 14 matrices that we used (the matrices have the following ids in the UFL collection: 261, 981, 1332, 1870, 1871, 1872, 1964, 2025, 2032, 2069, 2112, 2128, 2129, 2134), the hypergraph based approach obtains on average 16% fewer nonzeros than COLAMD. The best result we obtain is for the matrix `ch7-6-b4` (whose id is 2025) with 45% reduction, and the worst result we obtain is for the matrix `n4c5-b4` (whose id is 2112) with 3% increase. These matrices are not really large; we expect even better performance on larger matrices.

Fill-reducing ordering for Cholesky

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a sparse symmetric positive matrix. The Cholesky factorization of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is a lower triangular matrix with positive diagonal entries. A symmetric permutation on $\mathbf{A}$ is performed to reduce the number of nonzeros in $\mathbf{L}$. There are many alternatives for finding such a permutation (see [6] for a recent survey). To the best of our knowledge, all existing methods are based on the standard graph representation of a symmetric matrix, except the work in [3]. In this latter work, for a given $\mathbf{A}$, the authors find a sparse matrix $\mathbf{B}$ such that $\mathbf{A} = \mathbf{B}^T\mathbf{B}$ holds pattern-wise. Then the matrix $\mathbf{B}$ is ordered as in the previous section to permute $\mathbf{A}$ into doubly bordered block diagonal form, and a constrained approximate minimum degree algorithm is run to have the final ordering on $\mathbf{A}$. The gist of this approach is the algorithm that finds a matrix $\mathbf{B}$ that is the most useful for the partitioning purposes.

We show that within this framework one only needs to find a matrix $\mathbf{B}$ such that $\mathbf{A} \subseteq \mathbf{B}^T\mathbf{B}$ holds pattern-wise. Therefore, the equality that is enforced in [3] is an unnecessary restriction. By removing this restriction, we are able to search for a $\mathbf{B}$ within a much larger search space. We follow a few heuristics to find an upper triangular matrix $\mathbf{B}$ such that $\mathbf{B}^T\mathbf{B}$ covers all of the nonzeros of $\mathbf{A}$ and can possibly have more nonzeros. Our objective is to try to reduce the number of nonzeros in $\mathbf{B}$ rather than trying to obtain exactly the same number of nonzeros in $\mathbf{A}$ and $\mathbf{B}^T\mathbf{B}$. Our algorithm performs a number of iterations. At each iteration we find a spanning tree (or a forest) of the graph corresponding to the matrix containing the entries that have not been covered yet in $\mathbf{B}^T\mathbf{B}$. Then the algorithm adds the pattern of this tree to the currently found $\mathbf{B}$ and proceeds to the next iteration. Our approach tends to find $\mathbf{B}$ matrices that have fewer nonzeros than those found in [3].

For the initial comparisons we choose a set of matrices from those used in [3] which have ids 50 200, 201, 340, 346, 350, 362, 752, 760, 763, 764, 868, 1238, 1310, 1311, 1605 in the UFL collection. We compare the number of nonzeros in the $\mathbf{L}$ factors with the numbers reported for the AMD and MeTiS based ordering methods in the UFL collection. The averages of the ratios of our method to AMD and MeTiS are 0.94 and 0.99, respectively. The best result with respect to AMD is obtained for the `finan512` matrix, where the ratio is 0.56 (the worst ratio is 1.09 obtained for `shuttle_eddy`). The best and the worst results with respect to MeTiS are 0.83 and 1.28 obtained for `bcsstk28` and `k1_san`, respectively. The $\mathbf{B}$ matrices that we find have about 35% of the nonzeros of the original matrices. We do not compare the run times as most of our algorithms are implemented in MATLAB. We plan to do run time comparisons

after improving and implementing the algorithms in a common programming language.

Bibliography

[1] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1860–1879.

[2] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at `http://bmi.osu.edu/~umit/software.htm`, 1999.

[3] Ü. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, *Hypergraph partitioning-based fill-reducing ordering*, Technical Report OSUBMI-TR-2009-n02, The Ohio State University, Department of Biomedical Informatics, 2009.

[4] T. A. Davis, *Direct Methods for Sparse Linear Systems*, no. 2 in Fundamentals of Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[5] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, *A column approximate minimum degree ordering algorithm*, ACM Trans. Math. Softw., 30 (2004), pp. 353–376.

[6] I. S. Duff and B. Uçar, *Combinatorial problems in solving linear systems*, Technical Report TR/PA/09/60, CERFACS, Toulouse, France, 2009.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 3rd ed., 1996.

12 Graph partitioning with separation of terminals

Roxana Ionuţiu
(*Jacobs University Bremen, Germany and TU Eindhoven, the Netherlands)*

Joost Rommes
(*NXP Semiconductors, the Netherlands)*

Wil Schilders
(*TU Eindhoven, the Netherlands)*

Corresponding Author: **Roxana Ionuţiu**
(r.ionutiu@jacobs-university.de)

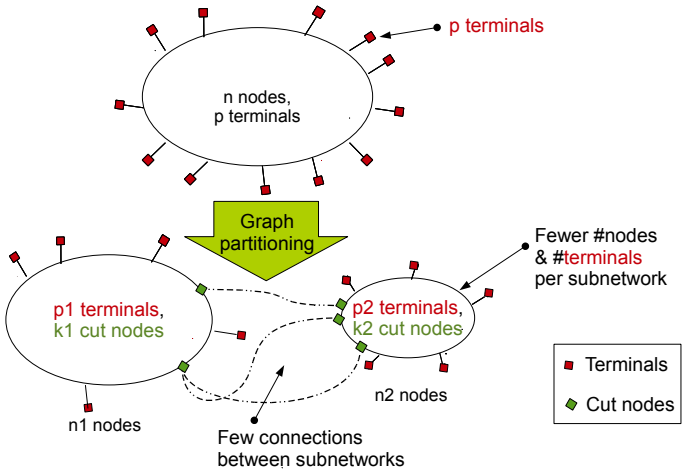
A graph partitioning problem is presented with applications in circuit simulation. Interpreted as graphs, very large electrical circuits are partitioned into smaller components which are minimally connected among each-other. Extending the standard partitioning scope, we formulate an additional constraint for appropriately distributing a special subset of nodes (terminals) across partitions. If incorporated inside partitioning tools, the terminal constraint could further enhance analysis of challenging electrical circuits.

Introduction

In circuit simulation very large electrical networks have to be analyzed, which contain millions of nodes interconnected via basic circuit elements. An analogy with graphs is immediate: the circuit nodes are the graph vertices, while the basic circuit elements form the edges. Networks of industrial relevance have an additional feature: a large subset of their nodes are the input/output nodes, called *terminals*, which connect the large network with remaining circuitry. Terminals may form a large fraction of the total number of nodes. E.g. if n is total number of nodes of which p are terminals typical values are $n = 10^5$, $p = 10^3$. Using graph partitioning such large multi-terminal networks are split into components which are treated individually in further analysis steps. We will show for instance how graph partitioning helps to efficiently obtain, from an original large circuit, a reduced circuit which is "small and sparse" [i.e., its graph representation ideally has fewer vertices (circuit nodes) and fewer edges (basic circuit elements) than the original].

Problem definition

While existing graph partitioners mainly aim at minimizing the communication among components, a new problem emerges when a special subset of nodes (here, terminals) should be taken into account: components should be minimally connected and also satisfy a desirable terminal to node ratio. Fig. 1 provides



Given is a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of $n = |V|$ vertices (nodes) and $E = \{(v_i, v_j)|\ v_i, v_j \in V$ and v_i is directly connected to $v_j;\ 1 \leq i, j \leq n\ \}$ is the set of edges. Let $P \subset V$ form a special subset of nodes called *terminals*, $|P| = p < n$.
Problem: find a partitioning of G into *subnets* which are (a) minimally connected and (b) distribute the p terminals subject to a user-defined constraint.

Figure 1: Graph partitioning with separation of terminals

a rough problem visualization and formulates the partitioning task. More precisely, let G be partitioned into two[2] *subnets* $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$, $|V_1| = n_1$, $|V_2| = n_2$, $E_1 = \{(v_i, v_j) \in E|\ v_i, v_j \in V_1;\ 1 \leq i, j \leq n_1\}$, $E_2 = \{(v_i, v_j) \in E|\ v_i, v_j \in V_2;\ 1 \leq i, j \leq n_2\}$. Let $P_1 \subset V_1$ terminals fall under V_1, and $P_2 \subset V_2$ terminals fall under V_2, $|P_1| = p_1$, $|P_2| = p_2$. Let $K_1 \subset V_1$ and $K_2 \subset V_2$ be the *cut nodes*[3] separating G_1 and G_2, $|K_1| = k_1$, $|K_2| = k_2$, $K_1 \cap K_2 = \emptyset$. G_1 and G_2 thus communicate via the edge set $E_{12} = \{(v_i, v_j) \in E|v_i \in K_1, v_j \in K_2;\ 1 \leq i \leq k_1,\ 1 \leq j \leq k_2\}$. The problem is to find *the smallest set of cut nodes* K_1 and K_2[4] which partition G so that the following quantity is minimized:

$$\min_{K_1 \subset V_1,\ K_2 \subset V_2} \underbrace{\frac{(p_1+k_1)(p_1+k_1-1)}{2} + \frac{(p_2+k_2)(p_2+k_2-1)}{2} + k_1k_2 + p_1k_2 + p_2k_1}_{Maxfill} \qquad (8)$$

Note that subnet sizes n_1 and n_2 need not be the same. Rather, it is more important that terminals are appropriately distributed. As will be shown, in model reduction [1] objective (8) controls sparsity: the quantity to minimize is

[2] A natural extension is a partitioning into $k > 2$ subnets; the 2-way partitioning is presented here for simplicity.
[3] If $p_K > 0$ terminal nodes fall under the cut nodes, then $p_1 + p_2 + p_K = p$.
[4] In other words, find the smallest node separator $K = K_1 \cup K_2$.

the maximum fill-in generated by reducing[5] each subnet individually.

Preliminary results

A network with $n = 16862$ nodes of which $p = 646$ terminals was partitioned into two subnets communicating via a set of cutnodes (separator). Two partitioning algorithms (Mondriaan [3], and Nested dissection [2] via Metis [4]) were applied without explicitly minimizing (8). Results are recorded below. Note that with Mondriaan the subnets differ in size, while with Nesdis they are almost equal. This flexibility contributed to a smaller *Maxfill* value via Mondriaan. Incorporating an additional constraint into existing partitioners as to explicitly minimize fill-in remains an open problem.

	Mondriaan			
	#nodes	#terminals	#cutnodes	*Maxfill*
Subnet 1	$n_1 = 14260$	$p_1 = 509$	$k_1 = 58$	216771
Subnet 2	$n_2 = 2602$	$p_2 = 123$	$k_2 = 58$	
	Nesdis			
Subnet 1	$n_1 = 8507$	$p_1 = 328$	$k_1 = 105$	252276
Subnet 2	$n_2 = 8355$	$p_2 = 302$	$k_2 = 104$	

Bibliography

[1] Ionutiu, R. , Rommes, J., Schilders, W.H.A.: Model reduction for multi-terminal RC circuits. 8th International Conference Scientific Computing in Electrical Engineering, Toulouse, France, September 19-24, 2010. Submitted / in press.

[2] Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 887: Cholmod, supernodal sparse Cholesky factorization and update/downdate. ACM Trans. Math. Softw. **35**(3), 1–14 (2008).

[3] Yzelman, A. N. and Bisseling, R. H: Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods. SIAM Journal on Scientifc Computing, **31**, 3128–3154 (2009). Software online: `http://www.staff.science.uu.nl/~bisse101/Mondriaan`

[4] G. Karypis and V. Kumar, *METIS, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, version 4.0 ed., University of Minnesota, Department of Computer Science / Army HPC Research Center, Minneapolis, MN 55455, September 1998. [Online]. Available: `http://glaros.dtc.umn.edu/gkhome/metis/metis/download`

[5] Reduction refers to a Gaussian-type elimination of internal nodes (nodes that are neither terminals nor cut-nodes).

13 Piecewise Linearization by Algorithmic Differentiation

Andreas Griewank and Felix Dalkowski
(*Humboldt-Universität zu Berlin, Germany)*

Corresponding Author: **Andreas Griewank**
(griewank@math.hu-berlin.de)

Function model

Using the framework of ([1]) we suppose that the vector function $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$ in question is evaluated by a sequence of assignments

$$v = u \circ w \quad \text{or} \quad v = \varphi(u)$$

where $\circ \in \{x, -, *\}$ is a polynomial arithmetic operation and

$$\varphi \in \Phi \equiv \{\text{rec}, \text{sqrt}, \sin, \cos, \exp, \log, \ldots, \textbf{abs}, \min, \max, \textbf{sign}, \ldots\}$$

a univariate function. Here we are particular concerned with the three Lipschitz continuous elementals **abs**, min, max and the function **sign**, which has a discontinuity in the interior of its domain. All of them are piecewise linear and on can easily express the former three in terms of the latter, namely **sign**. It also yields the Heaviside functions and thus can be used as representer of all nonsmooth piecewise linear functions. However, we will distinguish as *composite Lipschitz continuous* the situation where **sign** is uses by the programmer in such a way that the result is still continuous. context of min, max and **abs**.

Piecewise linearization

Now we can simply replace all smooth elementals by their tangent approximation and **sign** and thus implicitly also min, max, **abs** by themselves. Starting from an increment vector Δx for the independents we can then propagate increments Δv for all intermediate quantities according to the following rules:

$$\begin{aligned}
\Delta v &= \Delta u \pm \Delta w \quad \text{for} \quad v = u \pm w \\
\Delta v &= u * \Delta w + \Delta u * w \quad \text{for} \quad v = u * w \\
\Delta v &= c * \Delta u \quad \text{with} \quad c \equiv \varphi'(u) \quad \text{for} \quad v = \varphi(u) \not\equiv \textbf{sign}(u) \\
\Delta v &= \textbf{sign}(u + \Delta u) - \textbf{sign}(u) \quad \text{for} \quad v = \textbf{sign}(u)
\end{aligned}$$

We will denote the resulting increment Δy as $\Delta F(x; \Delta x)$. If all sign elementals are noncritical in that none of their arguments $u = u(x)$ vanish exactly at the given x then F is locally differentiable at x and we have for all sufficiently small Δx by the chain rule $\Delta y = \Delta F(x; \Delta x) \equiv F'(x)\Delta x$ where $F'(x) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix. As observed in [1] we thus have differentiability at almost all points x, but the crux is that in the vicinity of any nondifferentiability the local linearization $F'(x)\Delta x$ will not be a good approximation to $F(x+\Delta x)-F(x)$.

Consequently equation solvers and other numerical methods based on local linear models cannot work well, except in the immediate vicinity of a nonsingular solution. Instead we work with the approximation $\Delta y = \Delta F(x; \Delta x)$. Assuming that the evaluation procedure contains s sign value calls with arguments u_i for $i = 1 \ldots s$ we can represent the switching structure by the signature vector

$$\sigma = (\sigma_i)_{i=1\ldots s} \in \{-1, 0, 1\}^s \quad \text{with} \quad \sigma_i \equiv \mathbf{sign}(u_i)$$

Basic properties of piecewise linear approximation

- At any $x \in \mathcal{D}$ the function $\Delta F(x; \Delta x)$ is defined for all $\Delta x \in \mathbb{R}^n$
- $\mathbb{R}^n$ is the disjoint union of at most 3^s nonempty simplices S_σ
- $\Delta F(x; \Delta x)$ is linear on the relative interiors S_σ° with Jacobians $J_\sigma \in \mathbb{R}^{m \times n}$
- For every $\Delta x, v \in \mathbb{R}^n$ we can compute directly from the computational graph the largest $\hat{\tau} \geq 0$ so that $\Delta F(x; \Delta x + \tau v)$ is linear for $0 < \tau < \hat{\tau}$.

In the Lipschitz-continuous case we have furthermore:

- $F(x + \Delta x) - F(x) - \Delta F(x; \Delta x) = O(\|\Delta x\|^2)$
- $\Delta F(z; \Delta x) - \Delta F(x; \Delta x) = O(\|\Delta x\| \, \|z - x\|)$
- If the common facet $S_\sigma \cap S_{\tilde{\sigma}}$ has the maximal dimension $n - 1$ then $J_\sigma - J_{\tilde{\sigma}} = 2ba^\top$, where a is a nonzero normal of the facet.

The structural properties in the Lipschitz continuous case allow the solution of the path equation $\Delta F(x; \Delta x) = -t\, F(x)$ for $t \in [0, 1]$ under the coherent orientation condition of Robinson [2]. The method is similiar to that proposed by Eaves [3]. Similarly, we work on bundle type methods for unconstrained optimization, where the generalized gradients are obtained in a systematic way. We discuss these approaches and demonstrate the surprisingly simple evaluation of piecewise linear models based on ADOL-C.

Bibliography

[1] Andreas Griewank and Andrea Walther: *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation. Second edition.* SIAM, 2008.

[2] Stephen M. Robinson: *Normal Maps Induced by Linear Transformations*, Mathematics of Operations Research, Vol. 17, No. 3, 1992.

[3] B. Curtis Eaves and Herbert E. Scarf: *The Solution of Systems of Piecewise Linear Equations*, Mathematics of Operations Research, Vol. 1, No. 1, 1976. [CFDP 390 and CFP 434]

14 Preconditioners based on Strong Components

Kamer Kaya
(*CERFACS, France)*

Iain Duff
(*RAL, United Kingdom)*

Corresponding Author: **Kamer Kaya** (`kamer@cerfacs.fr`)

We propose a new method for constructing a preconditioning matrix $\mathbf{M}$ to accelerate the solution of the system

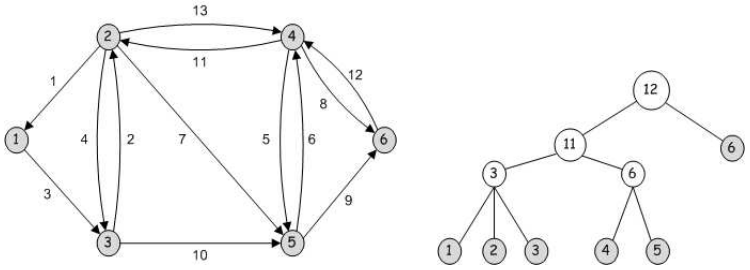
$$\mathbf{Ax} = \mathbf{b},$$

when using Krylov-based iterative methods. The coefficient matrix $\mathbf{A}$ is large and sparse and we assume it is irreducible, that is it cannot be permuted to block triangular form. If it is reducible, then we will apply our algorithms to the irreducible blocks on the diagonal.

Let $G = (V, E)$ be a strongly connected digraph with n vertices and m weighted edges. Given a permutation σ_0 on E, a hierarchical decomposition of G into its strong components can be defined in the following way. For $1 \leq i \leq m$, let $\sigma_0(i)$ be the ith edge in σ_0. Let $G_0 = (V, \emptyset)$ be the graph obtained by removing all the edges from G. Consider that edges are added one by one to G_0 in the order determined by σ_0. Let $G_i = (V, \{\sigma(j) : 1 \leq j \leq i\})$ be the digraph obtained after the addition of the first i edges. Initially in G_0, there are n strong components, one for each vertex, and, during the edge addition process, the strong components gradually coalesce until there is only one. The hierarchical decomposition of G into its strong components with respect to the edge permutation σ_0 shows which strong components are formed in this process hierarchically. Note that a strong component in a hierarchical decomposition is indeed a strong component of some digraph G_i although it is only a strong subgraph of G.

A hierarchical decomposition can be represented with a hierarchical decomposition tree T, whose leaf nodes correspond to the vertices in V, non-leaf nodes correspond to edges in E that create strong components during the process, and subtrees correspond to the decomposition trees of the strong components that form as the process proceeds. An example digraph and the corresponding decomposition tree can be found in Fig. 1.

Given a digraph $G = (V, E)$ and a permutation σ_0, the hierarchical decomposition tree T can be obtained by first constructing G_0 and executing Tarjan's strong component algorithm (`SCC`) for each internal digraph G_i obtained during the edge addition process. This would be an $\mathcal{O}(mn + m^2)$ algorithm since $1 \leq i \leq m$ and the cost of `SCC` is $\mathcal{O}(n+m)$. It would thus be prohibitive for large graphs. To obtain T in a more efficient way, Tarjan first proposed a recursive algorithm of complexity $\mathcal{O}(m\log^2 n)$ that he later improved to have complexity $\mathcal{O}(m \log n)$ [4]. A high level description of Tarjan's hierachical decomposition algorithm, `HD`, is given in Algorithm 1.



(a) A digraph (b) The HD tree

Figure 1: A digraph and its hierarchical decomposition tree with respect to the ordering defined on the edges of the digraph.

1: **if** $|E| - i = 1$ **then**
2: The edge $\sigma(|E|)$ makes the graph strongly connected. Return the tree T containing the vertices in V as leaves and a root.
3: **else**
4: Set $j = \lceil (i + |E|)/2 \rceil$.
5: **if** G_j is strongly connected **then**
6: Call $\texttt{HD}(G_j, \sigma, i)$.
7: **else**
8: For each strong component SC of G_j, containing more than one vertex, call $\texttt{HD}(SC, \sigma_s, i_s)$ where σ_s is the permutation for $E(SC)$ and $i_s = \max(k)$ such that SC_k is known to be acyclic.
9: Create a condensed graph G' from G by condensing each strong component of G_j into one vertex. Call $T = \texttt{HD}(G', \sigma', i')$ where σ' is the permutation for $E(G')$ and $i' = \max(k)$ s.t. G'_k is known to be acyclic.
10: Replace each leaf of T with the subtrees obtained by previous calls. Return T.
11: **end if**
12: **end if**

Algorithm 1: $T = \texttt{HD}(G = (V, E), \sigma, i)$. Here, σ is the permutation of E and G_i is known to be acyclic. For the initial call, $\sigma = \sigma_0$ and $i = 0$.

Given a matrix $\mathbf{A}$, our algorithm hierarchically decomposes the corresponding digraph into its strongly connected subgraphs. Later by using the decomposition, it permutes the rows and columns of the original matrix $\mathbf{A}$ and obtains a block triangular preconditioning matrix containing a subset of the nonzeros of $\mathbf{A}$ where the maximum size of a diagonal block is smaller than a desired value mbs.

We propose some modifications to make `HD` suitable for preconditioning: Our first modification allows us to have non-distinct edge weights, that is matrices with the same value in different positions. To achieve this, we removed the necessity of using weights during the decomposition process and instead use the permutation σ_0. Another modification to the HD algorithm is that we do not want to accept any blocks larger than a predefined value, mbs, and this enables us to stop the recursion earlier than in the original algorithm. As an example, when $mbs = 3$ for the example in Fig. 1, we do not need to decompose the components rooted with 3 and 6 since their sizes are already smaller than mbs. Our last modification increases the chance for obtaining components of size at most mbs by combining smaller components already obtained. To achieve this, during the decomposition process, we delete some edges of the graph that can only be used to obtain components with more than mbs vertices. Further explanation of these modifications is given in our technical report [2].

We use the modified algorithm to obtain a block triangular matrix $\mathbf{M}$ where the strong components correspond to the blocks on the diagonal of $\mathbf{M}$. To the best of our knowledge, this is the first work that uses `HD` for preconditioning purposes. After we obtain the block triangular form from this modified `HD`, we then see whether any blocks can be merged and finally use a greedy algorithm to order the blocks on the diagonal so that most of the entries are in the upper triangular part. Before we use algorithm `HD`, we first scale and permute the matrix using `MC64` [1], which we do also for our experiments on other preconditioners.

We compare our algorithm `SCPRE` with a block preconditioner `XPABLO` [3] and a MATLAB version of the industry-standard `ILUT` on sets of matrices from circuit and device simulations from the University of Florida sparse matrix collection. We show below a very abbreviated table from the results in [2]. In the table, we give the number of iterations (with the least in bold font) and the relative memory requirement (in the second line for each matrix). For `SCPRE`, we use $mbs = 1000$.

For circuit simulation problems (the upper part), `ILUT` and `SCPRE` converge for all matrices in this set. `XPABLO` fails to converge for *bcircuit* and *ckt11752_dc_1* and so `SCPRE` is clearly the best block based preconditioner on this set of matrices. Although `ILUT` requires significantly fewer iterations on *G2_circuit* and *ckt11752_dc_1*, in both cases it requires more memory. However, for *G2_circuit*, if we increase mbs to 5000, the number of iterations drops to 95 and our relative memory requirement increases to 6.10 and, for *ckt11752_dc_1*, by increasing mbs to only 3000 we require only 11 iterations with a relative memory cost of only 1.45. Thus we feel we can we recommend using `SCPRE` for circuit simulation matrices especially when the amount of memory to store the preconditioner is the main concern.

For the device simulation matrices in the bottom part of the table, the block based preconditioners are far more robust on this set with convergence for all the test matrices. We therefore feel that we can recommend `SCPRE` as the preconditioner for the device simulation matrices.

Matrix	XPABLO	ILUT	SCPRE
G2_circuit	727	**89**	642
	1.77	5.47	2.23
circuit_3	572	**3**	14
	1.25	2.16	1.40
bcircuit	-	238	**16**
	1.32	1.10	1.38
ckt11752_dc_1	-	**11**	213
	1.02	2.55	1.32
mult_dcop_01	12	6	**1**
	1.04	22.48	0.86
2D_27628_bjtcai	**45**	-	142
	1.85	2.58	2.22
3D_28984_Tetra	232	-	**98**
	2.82	1.98	2.65
ibm_matrix_2	23	-	**13**
	5.39	23.24	5.12
matrix_9	**182**	-	205
	4.96	37.32	2.27
wang3	100	**20**	79
	2.42	8.02	3.85

To balance these good results, we show in our paper [2] that ILUT outperforms both block approaches on matrices from CFD applications. Further research is needed to understand the effect of structure in determining the best approach.

Bibliography

[1] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22:973–996, 2001.

[2] I. S. Duff and K. Kaya. Preconditioners based on strong components. Technical Report TR/PA/10/97, CERFACS, Toulouse, France, 2010.

[3] D. Fritzsche, A. Frommer, and D. B. Szyld. Extensions of certain graph-based algorithms for preconditioning. *SIAM Journal on Scientific Computing*, 29:2144–2161, 2007.

[4] R. E. Tarjan. An improved algorithm for hierarchical clustering using strong components. *Information Processing Letters*, 17(1):37–41, 1983.

15 Computing Selected Eigenvalues of Banded Symmetric Matrices

Alex Druinsky and Sivan Toledo
(*Tel-Aviv University, Israel)*

Corresponding Author: **Alex Druinsky** (alexdrui@post.tau.ac.il)

We present a new algorithm for computing selected eigenvalues of a banded symmetric matrix A. The algorithm applies the bisection method [1] directly to the banded matrix. To compute the inertia of $A - \sigma I$, we use Kaufman's recent symmetric banded factorization algorithm [10]. The number of arithmetic operations that the algorithm performs on an n-by-n matrix with $2k+1$ nonzero diagonals to compute ℓ eigenvalues is $\Theta(\ell n k^2)$.

The state-of-the-art method for computing selected eigenvalues of a symmetric banded matrix works differently and has a different cost structure. The best existing method, implemented in LAPACK's routine DSBEVX [7], first reduces A to a tridiagonal form $T = UAU^*$ where U is unitary. The desired eigenvalues are then computed using the bisection algorithm on the tridiagonal T. If A is n-by-n and has k nonzero subdiagonals (and superdiagonals), the reduction of A to T requires $\Theta(n^2 k)$ work (arithmetic operations). An eigenvalue is then computed using a constant number of inertia computations on $T - \sigma I$ where σ is a real shift. The constant depends on the precision required, since the inertia computations essentially amount to a binary search that narrows down the interval that contains the sought-after eigenvalue. Each inertia computation factors its tridiagonal input matrix at a cost of $\Theta(n)$ operations. The total cost is $\Theta(n^2 k)$ even if one computes all the eigenvalues.

If the number ℓ of eigenvalues is such that $\ell k/n$ is small, the new algorithm is superior to DSBEVX. For a given matrix, the new algorithm is particularly attractive when the number of required eigenvalues is small.

Computing the inertia of a matrix is the fundamental computation in the bisection algorithm for computing selected eigenvalues. Computing the inertia of a matrix requires $\Theta(n^3)$ work for dense matrices and $\Theta(n)$ work for tridiagonal ones. Until recently, there were no other classes of matrices for which it was known how to compute a symmetric inertia-revealing factorization in $o(n^3)$ work. Sparse symmetric indefinite factorization codes can often compute the inertia dramatically faster (but not always), but there was no theory that guaranteed a $o(n^3)$ running time. The need to pivot can quickly destroy the sparsity in the trailing submatrix.

In 2007, the situation changed, but only for one class of sparse matrices, namely banded ones. Linda Kaufman discovered a symmetric factorization algorithm that requires $\Theta(nk^2)$ arithmetic operations and which computes the inertia as a side effect. Her algorithm is related to a slightly earlier symmetric banded factorization algorithm with the same running-time bound, but which could not be used for inertial computations [9]. Our algorithm exploits Kaufman's discovery.

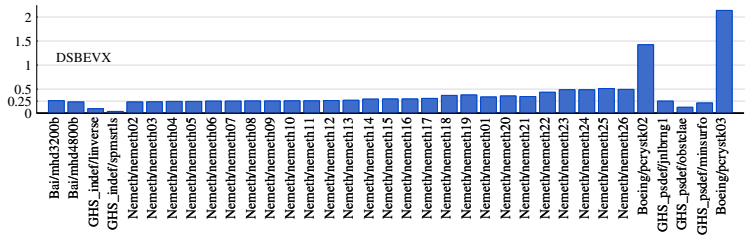


Figure 1: The ratio of the running time of our code and the running time of DSBEVX when both compute the 8 leftmost eigenvalues. Smaller numbers mean that our code is faster. The matrices are ordered by n^2k. The two matrices on which our code is slower have a much larger k/n ratio than the rest.

The inertia can also be computed from the principal minors of a symmetric matrix, which in turn can be computed from an unsymmetric factorization. If the matrix is banded, the band structure is preserved and the inertia computation costs $\Theta(nk^2)$ [4]. This alternative factorization is probably less efficient than Kaufman's, since it does not exploit symmetry. We plan to compare it to Kaufman's method but we have not yet done so.

We have implemented and tested an eigensolver based on bisection applied directly to a banded matrix, using Kaufman's factorization to compute inertia. The implementation uses Kaufman's banded factorization code. We have also generalized another technique by Kaufman [8]. In 2000 she observed that when a bisection eigensolver calls an inertia routine, the inertia is known almost exactly even before the computation begins (most of the time the uncertainty is 1). Therefore, the inertia routine can halt once enough negative or positive eigenvalues have been discovered to resolve the uncertainty, without completing the symmetric factorization process. Kaufman's observation focused on tridiagonal matrices, but it also applies to banded matrices.

We tested our new code on both random matrices and on a selection of 35 matrices from the University of Florida Sparse Matrix Collection. We chose matrices that were in reasonably-narrow banded form. We did not reorder the matrices. On 30 additional matrices from the same collection DSBEVX was too slow (took more than one hour). We ran the experiments on one core of a Core 2 computer running at 2.13 GHz with 2 GB of RAM, using BLAS and LAPACK routines from Intel's MKL version 10.2.

The results are shown in Figures 1 and 2. On most matrices, our code is much faster than LAPACK's when asked to compute 8 eigenvalues (Figure 1). On two of the matrices with a large k/n our code took longer than LAPACK. The scatter plot in Figure 2 shows that the running time ratio between the codes is indeed linearly correlated with $\ell k/n$. There are some outliers; we are currently investigating them.

The performance of our algorithm is somewhat sensitive to the location of the desired eigenvalues, because Kaufman's factorization algorithm is. The difference is at most a factor of 2. We omit a detailed analysis from this abstract.

We also omit from this abstract analysis of the significance of early halting of the factorization [8].

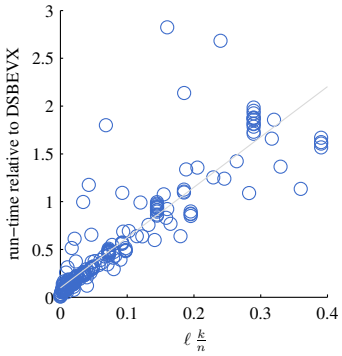


Figure 2: The relationship between the running time ratios of the two algorithms and $\ell k/n$. The plot contains all the matrices from Figure 1 and runs that compute 2^m eigenvalues for integer value of m from 1 up to the point at which our algorithm is slower. The line is the linear regression for the data points.

Other approaches to the computation of eigenvalues of symmetric banded matrices include Wilkinson's banded QR iteration [3, 2, pp. 557–560], which was included in EISPACK (as BQR). This approach seems to have fallen into disrepute and has not been implemented in LAPACK. High-quality Lanczos implementations are also reasonable candidates; we have begun comparisons with ARPACK, but we still do not have definite results [6].

The authors thank Dr. Linda Kaufman for her insightful comments and for providing the code of her retraction algorithm.

Bibliography

[1] GIVENS, WALLACE, *Numerical Computation of the Characteristic Values of a Real Symmetric Matrix*, Oak Ridge National Laboratory, 1954, ORNL-1574, Oak Ridge, Tennessee, Apparently the earliest paper about bisection for eigenvalues.

[2] WILKINSON, J. H., *The Algebraic Eigenvalue Problem*, Clarendon Press, 1965, Oxford

[3] MARTIN, R. S. AND REINSCH, C. AND WILKINSON, J. H. , *The QR Algorithm for Band Symmetric Matrices*, Numerische Mathematik, 1970, 16, 2, 85-92

[4] MARTIN, R. S. AND WILKINSON, J. H. , *Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices*, Numerische Mathematik, 1967, 9, 2, 279-301

[5] SMITH, B. T. AND BOYLE, J. M. AND DONGARRA, J. J. AND GARBOW, B. S. AND IKEBE, Y. AND KLEMA, V. C. AND MOLER, C. B., *Matrix Eigensystem Routines – EISPACK Guide*, Springer-Verlag, Berlin/Heidelberg, 1976, Lecture Notes in Computer Science, 6, Second, 10.1007/3-540-07546-1, 978-3-540-07546-2

[6] Lehoucq, R. B. and Sorensen, D. C. and Yang, C., *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, Society for Industrial and Applied Mathematics, Philadelpia, PA, 1998, Software, Environments, and Tools, 6, 0-89871-407-9

[7] Anderson, E. and Bai, Z. and Bischof, C. and Blackford, S. and Demmel, J. and Dongarra, J. and Du Croz, J. and Greenbaum, A. and Hammarling, S. and McKenney, A. and Sorensen, D., *LAPACK Users' Guide*, Third, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999, Software, Environments, and Tools, 9, 0-89871-447-8 (paperback)

[8] Kaufman, Linda, *An Observation on Bisection Software for the Symmetric Tridiagonal Eigenvalue Problem*, ACM Transactions on Mathematical Software, 2000, 26, 4, 520-526, December, 10.1145/365723.365728

[9] Irony, Dror and Toledo, Sivan, *The Snap-Back Pivoting Method for Symmetric Banded Indefinite Matrices*, SIAM Journal on Matrix Analysis and Applications, 2006, 28, 2, 398-424, 10.1137/040610106

[10] Kaufman, Linda, *The Retraction Algorithm for Factoring Banded Symmetric Matrices*, Numerical Linear Algebra with Applications, 2007, 14, 3, 237-254, April, 10.1002/nla.529

16 Dispersive Lanczos

Haim Avron
(*IBM T. J. Watson Research Center, United States)*

Alex Druinsky and Sivan Toledo
(*Tel-Aviv University, Israel)*

Corresponding Author: **Sivan Toledo** (sivan.toledo@gmail.com)

We describe technique that allows Lanczos's algorithm to accurately and reliably compute all the eigenvalues of a symmetric real matrix A. The main innovation of the new technique is a method to produce a certificate that all the eigenvalues have been found, even if some have multiplicity greater than 1. To the best of our knowledge, no existing Lanczos code includes this feature, nor has it been described in the literature. The certificate also describes the numerical multiplicity of each eigenvalue. The key idea of the new method is to slightly and randomly perturb A so as to disperse multiple eigenvalues; with a high probability, the slight dispersal results in a matrix with no multiple eigenvalues. This provides the algorithm with a provably-effective termination criterion.

The Lanczos algorithm [1] uses matrix-vector multiplications to build a tridiagonal matrix T whose eigenvalues are related to those of A. Let n be the dimension of A and let k be the number of iterations (number of times A has been multiplied by a vector). The matrix T is k-by-k. In exact arithmetic, when t reaches the number of distinct eigenvalues of A, T has the same eigenvalues, each with multiplicity 1, and the Lanczos method breaks down. But rounding errors cause a behavior that is significantly different:

1. Breakdown or even near breakdown (a tiny offdiagonal element in T) is almost never observed.
2. Some eigenvalues of A may not appear in T even after n iterations, where as others may appear with a numerical multiplicity that is greater than that in A.
3. T often has some eigenvalues which are not eigenvalues of A, called spurious eigenvalues.

These behaviors occur because rounding errors quickly destroy the orthogonality of the Lanczos basis of the Krylov subspace, which should in theory be orthogonal. Lanczos knew this and suggested full orthogonalization in every step; this is often too expensive, so Lanczos's algorithm was abandoned as a method for computing accurate eigenvalues (even without orthogonality, Lanczos provides useful information on the spectrum of A, especially extremal ones). This led researchers to develop Arnoldi-like variants that explicitly orthogonalize the basis vectors. Unfortunately, explicit-orthogonalization variants tend to be much more expensive than plain Lanczos, especially when some eigenvalues have large numerical multiplicity. (Selective orthogonalization techniques sometimes reduce the cost, but not always.)

Significant progress was made in the early 1980s, when the *Lanczos phenomenon* was discovered [3]: even if one makes no effort to keep the basis vectors orthogonal, the spectrum of T eventually contains all the eigenvalues of A. A formal proof was given in the 1990's [4, 5]. This led to codes that run for more than n iterations in an attempt to reach this "eventual convergence" point [2, 3]. These Lanczos codes faced two problems. One problem is to classify eigenvalues of T into genuine ones that are also eigenvalues of A and spurious ones. There are several criteria that can be used; for example, every numerically-multiple eigenvalue of T is genuine (also an eigenvalue of A) .

The second problem is more difficult: how to determine that we have found all the eigenvalues of A? If we find n distinct genuine eigenvalues, we can clearly stop. But if we find $n' < n$, is it because A has only n' distinct eigenvalues, or because some eigenvalue of A still does not appear in T? The only attempt we are aware of to address this question is due to Parlett and Reid [2]. Their code assumes that if an interval $[a, b]$ between two genuine eigenvalues a and b does not contain any eigenvalue of T, then there are also no eigenvalues of A in $[a, b]$. This is just a heuristic, and their paper acknowledges that it can fail (they also write that it is quite robust).

Our new code improves on state-of-the-art codes from the 1980s by adding a robust way to decide when to stop and to determine multiplicities. If P is a matrix with $\|P\|_2 = \delta\|A\|_2$, then the eigenvalues of $A + P$ lie within distance δ of the eigenvalues of A. If, in addition, P is random, then the eigenvalues of $A+P$ are distinct. If A has multiple eigenvalues, the corresponding eigenvalues of $A + P$ will be close (closer than δ), but under certain conditions on P, they will not be too close. We choose δ so that the perturbed eigenvalues are close enough to those of A to satisfy the user's requirement (say an absolute error of at most $10^{-8}\|A\|_2$), while still allowing a tridiagonal eigensolver applied to T to separate n distinct eigenvalues of $A+P$. For example, if $\delta = 10^{-9}$, a tridiagonal eigensolver can compute the eigenvalues of T to within, say, $10^{-15}\|A\|_2$, allowing us to find genuine eigenvalues of $A + P$ to within $10^{-14}\|A\|_2$, which is less than the minimum separation we expect to see in the spectrum of $A + P$.

We have implemented the algorithm using double-precision hardware floating point arithmetic (in both Matlab and C, and using LAPACK [6] routines to compute the spectrum of T). We have also implemented the algorithm in C++ using extended precision, using MPACK [8], which internally uses QD [7] to implement double-double precision. The algorithm works well, but we have not performed enough experiments to fully evaluate it. We have also not yet implemented many optimizations that we have in mind (some of which are described below).

We are currently exploring several issues. The most important one is a probabilistic analysis of the minimum eigenvalue gap in $A+P$, which determines the accuracy with which we need to compute the eigenvalues of T (however, using extended-precision packages we can attain very high accuracy, albeit at the cost of performance). The minimum gap depends on the angle between invariant spaces of A and P and the dimensions of these subspaces, but we have not yet completed this analysis.

A second issue that we are working on is finding good ways to generate P, which needs to be cheap to apply. A diagonal P often works well (as we wrote, this depends on the invariant subspaces of A), but we are also exploring alternatives, such as matrices involving randomized fast transforms.

A third topic of research is the use of m different P's concurrently, with the same or different δ. Naively, this requires m times more memory and work, but this is not always the case. First, using several P's exponentially increases the probability that some perturbed matrix will not have small gaps in the spectrum, leading to a quick termination. Second, when using both small and large δ's, the T's corresponding to large δ's quickly give us coarse but correct information about the location of the eigenvalues of A, eliminating unnecessary work on the small perturbations.

The last issue that we are working on is additional criteria for deducing the multiplicity of eigenvalues, again in an attemp to avoid long iterations.

Bibliography

[1] Lanczos, Cornelius, *An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators* Journal of Research of the National Bureau of Standards 1950, 45, 4, 255-282, October

[2] *Parlett, B. N. and Reid, J. K.*, Tracking the Progress of the Lanczos Algorithm for Large Symmetric Eigenproblems, IMA Journal of Numerical Analysis, 1981, 1, 2, 135-155, 10.1093/imanum/1.2.135

[3] Cullum, Jane K. and Willoughby, Ralph A., *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol. 1. Theory.*, Birkhäuser, 1985, 3, Progress in Scientific Computing, Boston, 0-8176-3058-9, 3-7643-3294-9,

[4] Druskin, V. and Knizhnerman, L., *Error Bounds for the Simple Lanczos Procedure for Computing Functions of Symmetric Matrices and Eigenvalues*, Computational Mathematics and Mathematical Physics, 1991, 31, 7, 970-983, This is the Russian-language original that is cited in [5].

[5] Druskin, Vladimir and Knizhnerman, Leonid, *Krylov Subspace Approximation of Eigenpairs and Matrix Functions in Exact and Computer Arithmetic*, Numerical Linear Algebra with Applications, 1995, 2, 3, 205-217, May/June, 10.1002/nla.1680020303, This is an English-language survey paper of a bunch of publications in Russian.

[6] Anderson, E. and Bai, Z. and Bischof, C. and Blackford, S. and Demmel, J. and Dongarra, J. and Du Croz, J. and Greenbaum, A. and Hammarling, S. and McKenney, A. and Sorensen, D., *LAPACK Users' Guide*, Third, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999, Software, Environments, and Tools, 9, 0-89871-447-8 (paperback)

[7] Hida, Y. and Li, X. S. and Bailey, D. H., *Quad-double Arithmetic: Algorithms, Implementation, and Application*, Lawrence Berkeley National Laboratory, 2000, LBNL-46996, Berkeley, CA 94720, October

[8] Nakata, Maho, *The MPACK (MBLAS/MLAPACK); a multiple precision arithmetic version of BLAS and LAPACK*, http://mplapack.sourceforge.net, Version 0.6.6; 2010/08/06

17 Parallel computation of entries of A^{-1}

Patrick Amestoy and François-Henry Rouet
(*Université de Toulouse, INPT(ENSEEIHT)-IRIT, France)*

Iain S. Duff
(*Atlas Centre, RAL, United Kingdom)*

Jean-Yves L'Excellent
(*INRIA and LIP, ENS Lyon, France)*

Bora Uçar
(*CNRS and LIP, ENS Lyon, France)*

Corresponding Author: **François-Henry Rouet** (`frouet@enseeiht.fr`)

We are concerned about the parallel computation of several entries of the inverse of a large sparse matrix. We assume that the matrix has already been factorized by a direct method and that the factors are distributed. There are many applications where the computation of explicit entries of the inverse of a sparse matrix are required: statistical analysis of least-squares computations [1], atomistic level simulation of nanowires [2], computation of short-circuit currents [3], and approximations of condition numbers [4]. In most of these cases, many entries are requested, *e.g.*, all the entries of the diagonal, on which we focus in this abstract for the sake of simplicity.

A distributed sparse factorization can be represented by an assembly tree where each node of the tree corresponds to the partial factorization of a dense submatrix. Each node of the tree is associated with the variables that are eliminated during the partial factorization at that node. The factorization is performed from the leaf nodes to the root (we assume without loss of generality that the matrix is irreducible). As the factorization proceeds towards the root, the submatrices in general become larger so that more than one processor can be used at each node.

To compute column j of the inverse, the equation $Ax = e_j$ can be used, where e_j is the jth column of the identity matrix. One can obtain major savings if the structural zeros of e_j are exploited or if only one entry of the jth column is requested. If we have an LU factorization of A, a_{ij}^{-1}, the (i, j) entry of A^{-1}, is obtained by solving successively the two triangular systems:

$$\begin{cases} y = L^{-1}e_j \\ a_{ij}^{-1} = (U^{-1}y)_i \end{cases} \tag{9}$$

We see from the equations (9) that in the forward substitution phase, the right-hand side (e_j) contains only one nonzero entry and that, in the backward step, only one entry of the solution vector is required. The following result takes advantage of both these observations along with the sparsity of A to provide an efficient computational scheme.

Theorem 1 (Property 8.9 in [5]). *To compute a particular entry a_{ij}^{-1} of A^{-1}, the only nodes of the tree which have to be traversed are on the path from the node j up to the root node, and on the path going back from the root node to the node i.*

Using this result enables us to "prune" the tree by removing all the nodes which do not take part in the computation of an entry. Thus, for a single entry of the inverse, the pruned tree only consists of the root node, the nodes i and j, and all nodes on the unique paths between these nodes and the root. In the common case, when we require many entries of the inverse, we have to solve for many right-hand sides and thus we must compute the entries by blocks as we will not have sufficient memory to compute them all at once. One can apply tree pruning to these blocks of right-hand sides. In this case the pruned tree will be effectively the union of all pruned trees corresponding to each right-hand side in the block. In [6], we considered the combinatorial problem of partitioning the requested entries into blocks to minimize the overall cost in an out-of-core environment. In this work, we address the problem of how to compute such blocks of entries efficiently in parallel.

In order to compute a block of entries in parallel, we solve for several blocks of nb right-hand sides (that we call the "nb-blocks") at the same time. At first glance, this seems embarrassingly parallel. However, in a distributed memory environment, we need to run parallel instances of a linear solver in parallel, each one using the whole set of processors to solve for a block of right-hand sides (because all distributed factors might have to be accessed). This is not possible using MPI without replicating the factors on all processors.

In the computational setting of [5, 6], the right-hand sides are processed following an order which tends to put together nodes which are close in the assembly tree, *e.g.*, a postorder. As a consequence, few processors (probably only one) will be active in the lower part of tree when processing a block of right-hand sides. In order to provide more parallelism over the above setting, it is necessary to develop an *interleaving* strategy where the entries in the block are chosen not just from a postorder but so that every processor will be active when a block of entries is computed. Thus interleaving tends to cancel the benefits of a good permutation for sequential computation; it increases the number of accesses/flops because it puts together activities that correspond to distant branches/parts of the assembly tree. Also, choosing a large block of right-hand side columns to exploit parallelism will result in some processors doing far more flops if we consider the block as being indivisible. We still want to exploit the benefits of `BLAS`/dense computations but not at the expense of too many unneeded arithmetic operations.

We do this by processing at each node only the columns of the nb-block of right-hand sides for which the corresponding inverse entries are associated with the node or with a descendant of the node in the tree. Thus the subblock on which we do our computations is as small as it can be, and so we are as efficient in terms of operation count as is possible if the sparsity of an individual right-hand side were exploited. At the leaf nodes of the pruned tree, the block of computations will normally be quite small corresponding only to entries present at that node. As we progress up the tree, the computational block will increase, with the block at any node being the union of the blocks at the children with any new entries appearing at the node. At the root node (assuming irreducibility)

the block will be of size nb. One very important feature is that because the nb-block of right-hand sides is postordered, the block at any node will always be a contiguous subset of entries from the nb-block so that only the position of the first and last entries need to be passed and the merging process at a node is trivial.

On an earlier version of our algorithm, we requested a minimum size of computational block (that we call nb_{sparse}) so that the block size at any node was a multiple of nb_{sparse} although the contiguity property was still maintained. This had some attractiveness because of specifying minimum computational units for the `BLAS`, however it meant that we were doing unnecessary operations. We illustrate this in Table 1 by simulation runs: more operations are done with $nb_{sparse} = 32$ than with $nb_{sparse} = 1$, which corresponds to the algorithm described above. Without interleaving ("no IL"), few processors are active when processing a block, whereas interleaving ("IL") improves parallelism at the price of an increase in number of operations. The benefit of our approach ("IL, nb_{sparse}=1") is that a low arithmetic cost is obtained while still exploiting parallelism (the numbers in parenthesis give an idea of the amount of parallelism one can expect). By the time of the meeting, we plan to show actual performance results with MUMPS [7].

Strategy	Matrix	
	Purdue$_{201,400}$	11pt$_{125,000}$
no IL no nb_{sparse}	4.59×10^{12} (1.54)	7.85×10^{12} (2.11)
IL no nb_{sparse}	17.56×10^{12} (11.20)	17.35×10^{12} (11.15)
IL $nb_{sparse}=32$	2.59×10^{12} (11.20)	4.58×10^{12} (11.15)
IL $nb_{sparse}=1$	2.14×10^{12} (11.20)	4.19×10^{12} (11.15)

Table 1: Number of flops for the computation of the whole diagonal of the inverse of two matrices on 16 processors; block size is $nb = 512$; the numbers in parentheses indicate the average number of active processors at the leaves. Matrix Purdue$_{201,400}$ comes from [2], and matrix 11pt$_{125,000}$ corresponds to an eleven-point discretization of a $50 \times 50 \times 50$ domain.

Bibliography

[1] L. Bouchet, J. Roques, P. Mandrou, A. Strong, R. Diehl, F. Lebrun, and R. Terrier, *INTEGRAL SPI Observation of the Galactic Central Radian: Contribution of Discrete Sources and Implication for the Diffuse Emission 1*, The Astrophysical Journal, 635 (2005), pp. 1103–1115.

[2] S. Cauley, J. Jain, C. K. Koh, and V. Balakrishnan, *A scalable distributed method for quantum-scale device simulation*, Journal of Applied Physics, 101 (2007), p. 123715.

[3] K. Takahashi, J. Fagan, and M. Chin, *Formation of a sparse bus impedance matrix and its application to short circuit study*, in Proceedings 8th PICA Conference, Minneapolis, Minnesota, 1973.

[4] Å. Björck, *Numerical methods for least squares problems*, SIAM Press, 1996.

[5] Tz. Slavova, *Parallel triangular solution in the out-of-core multifrontal approach for solving large sparse linear systems*, PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 2009.

[6] P. R. Amestoy, I. S. Duff, Y. Robert, F.-H. Rouet and B. Uçar, *On computing inverse entries of a sparse matrix in an out-of-core environment*, SIAM J. Sci. Comput. (submitted) 2010, INPT-IRIT technical report RT-APO-10-06.

[7] P. R. Amestoy, A. Buttari, I. S. Duff, A. Guermouche, J.-Y. L'Excellent, B. Uçar, *MUMPS (MUltifrontal Massively Parallel Solver)*, Encyclopedia of Parallel Computing, David Padua (Eds.), Springer, 2010 (to appear).

18 Approaching optimality for solving SDD linear systems

Gary Miller and Richard Peng
(*Carnegie Mellon University, United States)*

Ioannis Koutis
(*University of Puerto Rico, Rio Piedras at Puerto Rico)*

Corresponding Author: **Gary Miller** (glmiller@cs.cmu.edu)

A linear system $Ax = b$ is symmetric diagonally dominant (SDD) when $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$. Solvers for such systems are emerging as a powerful algorithm primitive. They are the key subroutine in the fastest known algorithms for a multitude of problems that include the max-flow problem [1] and several optimization problems in computer vision [2].

The recent intense interest in algorithms that rely on SDD solvers was motivated by the seminal work of Spielman and Teng who gave the first nearly-linear time algorithm for such systems, running in time $O(m \log^c m \log(1/\epsilon))$, where m is the number of non-zero elements in the system matrix, ϵ is the approximation error, and c is some large constant [4]. The Spielman and Teng solver is a very complicated and impractical algorithm.

We present the fastest known solver for SDD systems. Its running time is $O(m \log^2 m \log(1/\epsilon))$, up to lower order factors. The algorithm is simple, concise, and potentially practical.

The key contribution: Better Graph Sparsification

It is well known that general SDD systems are linear-time reducible to systems $L_A x = b$ where L_A is the Laplacian of a graph A. The Laplacian is a special SDD matrix with non-positive off-diagonal elements and zero row-sums. We can thus focus our attention to systems on Laplacians. This brings the problem into the realm of combinatorial scientific computing and –more specifically– *combinatorial preconditioning* an area of research initiated by P. Vaidya [5].

The key to the design of a fast solver is *graph sparsification.* In the graph sparsification problem one wants to approximate a given graph A with a graph B such that: (i) B has significantly fewer edges than A and (ii) the condition number $\kappa(L_A, L_B)$ is as small as possible. There is a trade-off between the number of edges in B and the condition number between the two Laplacians.

Spielman and Teng proved that if A has n nodes and m edges, it is possible to produce in $O(m \log^c n)$ time, a graph B with $n + m/k$ edges such that $\kappa(L_A, L_B) \leq k \log^c n$. Their algorithm is based on complicated graph decompositions that partition the edges of the graph into sets which can be sparsified by uniform sampling.

Spielman and Srivastava [3] studied the related problem of generating a sparse graph B, under the constraint that $\kappa(L_A, L_B) < (1 + \epsilon)$. They gave a remarkable algorithm that produces a graph B with $O(n \log n/\epsilon^2)$ edges via a simple sampling procedure: the edges of A are sampled and added to B with

probabilities proportional to their effective resistance in the resistive network defined by A. However, computing the effective resistances seems to require the solution of a system.

Our *incremental sparsification* algorithm uses ideas from [3] to show that the edges of A can be sampled with probabilities proportional to *upper bounds* on their effective resistances. We also show that it is possible to compute in $O(m \log^2 n)$ time upper bounds that are strong enough to generate a graph B with $n + m/k$ edges such that $\kappa(L_A, L_B) \leq k \log^2 n$. Properly combined with known techniques, the incremental sparsification algorithm yields our solver.

Acknowledgement This project is partially supported by the National Science Foundation under grant number CCF-1018463.

Bibliography

[1] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. *CoRR*, abs/1010.2921, 2010.

[2] Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. In *International Symposium of Visual Computing*, pages 1067–1078, 2009.

[3] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 563–568, 2008.

[4] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, June 2004.

[5] P.M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. A talk based on this manuscript, October 1991.

19 Parallel Sparse Matrix Indexing and Assignment

Aydın Buluç
(*Lawrence Berkeley National Laboratory, United States)*

John Gilbert
(*University of California, Santa Barbara, United States)*

Corresponding Author: **Aydın Buluç** (abuluc@lbl.gov)

This paper presents distributed-memory parallel algorithms for generalized sparse-matrix indexing and assignment. We show that our algorithms, which use parallel sparse matrix-matrix multiplication as a subroutine, are fast and scalable for the most general case.

Introduction

Given two sparse vectors of indices, I and J, SpRef is the operation of storing a submatrix of a sparse matrix in another sparse matrix ($\mathbf{B} = \mathbf{A}(\mathsf{I}, \mathsf{J})$). It extracts all the $\mathsf{I}(i)$th rows and all the $\mathsf{J}(j)$th columns for $i = 1, ..., length(\mathsf{I})$ and $j = 1, ..., length(\mathsf{J})$, respecting the order of indices. When $\mathbf{A}$ is the sparse adjancency matrix of a graph, SpRef corresponds to subgraph selection.

Simple cases such as row-wise ($\mathbf{A}(i, :)$), column-wise ($\mathbf{A}(:, i)$), and element-wise ($\mathbf{A}(i, j)$) indexing is often handled by special purpose subroutines [5]. A parallel algorithm for the general case, where I and J are arbitrary vectors of indices, does not exist in the literature. We propose an algorithm that uses parallel sparse matrix-matrix multiplication (SpGEMM). Our algorithm is amenable to performance analysis for the general case.

SpAsgn is the operation of assigning a sparse matrix to a submatrix of another sparse matrix ($\mathbf{B}(\mathsf{I}, \mathsf{J}) = \mathbf{A}$). For SpAsgn, we only describe the algorithm and omit performance results and analysis due to lack of space.

Sequential Algorithm

Performing SpRef by a triple sparse-matrix product is illustrated in Figure 1. The algorithm can be decribed concisely in Matlab notation as follows:

```
function B = sprefm(A,I,J)

[m,n] = size(A);
len_i = length(I);
len_j = length(I);
R = sparse(1:len_i,I,1,len_i,m);
Q = sparse(J,1:len_j,1, n,len_j);
B = (R*A)*Q;
```

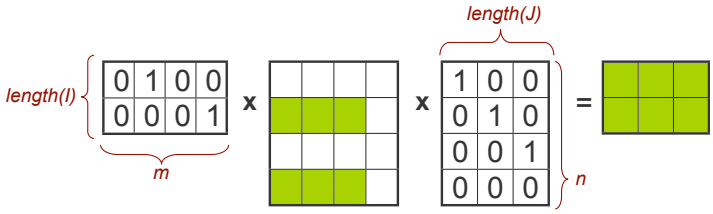


Figure 1: Sparse matrix indexing (SpRef) using mixed-mode sparse matrix-matrix multiplication (SpGEMM). On an m-by-n matrix $\mathbf{A}$, the SpRef operation $\mathbf{A}(\mathsf{I}, \mathsf{J})$ extracts a submatrix of size $length(\mathsf{I})$ x $length(\mathsf{J})$, where I is a vector of row indices and J is a vector of column indices. The example shows the operation for $\mathbf{A}([1,3],[0,1,2])$. It performs two SpGEMM operations between a boolean matrix and a floating point matrix.

The sequential complexity of this algorithm is $\text{flops}(\mathbf{R}\cdot\mathbf{A}) + \text{flops}((\mathbf{RA})\cdot\mathbf{Q})$. Due to the special structure of the permutation matrices, the nonzero operations required to form the product $\mathbf{R}\cdot\mathbf{A}$ is equal to the number of nonzero elements in the product. More precisely, $\text{flops}(\mathbf{R}\cdot\mathbf{A}) = nnz(\mathbf{R}\cdot\mathbf{A}) \leq nnz(\mathbf{A})$. Similarly, $\text{flops}((\mathbf{RA})\cdot\mathbf{Q}) \leq nnz(\mathbf{A})$, making the overall complexity of SpRef $O(nnz(\mathbf{A}))$ for any I and J.

Performing SpAsgn by two triple sparse-matrix products and additions is illustrated in Figure 2. We create two temporary sparse matrices that are of identical dimensions to $\mathbf{A}$. These matrices contain nonzeros only for the $\mathbf{A}(\mathsf{I}, \mathsf{J})$ part, and zeros elsewhere. The first triple product embeds $\mathbf{B}$ into a bigger sparse matrix that we add to $\mathbf{A}$. The second triple product embeds $\mathbf{A}(\mathsf{I}, \mathsf{J})$ into an identically sized sparse matrix so that we can zero out the $\mathbf{A}(\mathsf{I}, \mathsf{J})$ portion by subtracting it from $\mathbf{A}$. The algorithm can be decribed concisely in Matlab notation as follows:

```
function C = spasgnm(A,I,J,B)
% A = spasgnm(A,I,J,B) performs A(I,J) = B

[ma,na] = size(A);
[mb,nb] = size(B);
R = sparse(I,1:mb,1,ma,mb);
Q = sparse(1:nb,J,1,nb,na);
S = sparse(I,I,1,ma,ma);
T = sparse(J,J,1,na,na);
C = A + R*B*Q - S*A*T;
```

Parallel Algorithm

The parallelization of the SpRef algorithm poses multiple challenges. The boolean matrices have only one nonzero per row and column, respectively. For the parallel 2D algorithm to scale well with increasing number of processors, data structures and algorithms should respect hypersparsity [2]. Furthermore, communication should take place over a single processor dimension, lowering

$$\mathbf{A} = \mathbf{A} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & \mathbf{B} & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ 0 & \mathbf{A}(\mathsf{I},\mathsf{J}) & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Figure 2: Conceptual illustration of the SpAsgn ($\mathbf{A}(\mathsf{I},\mathsf{J}) \leftarrow \mathbf{B}$) operation. For simplicity, the vector indices I and J are assumed to be contiguous, which is not required by the algorithm.

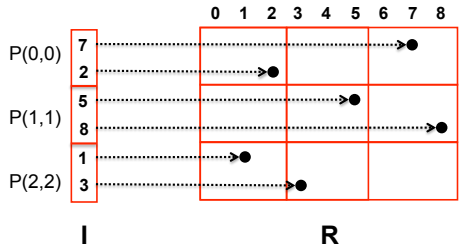


Figure 3: Parallel forming of the left hand side boolean matrix $\mathbf{R}$ from the sparse index vector I on 9 processors, logically forming a 3×3 processor grid. $\mathbf{R}$ will be subsequently multiplied with $\mathbf{A}$ to extract 6 rows out of 9 from $\mathbf{A}$ and order them as $\{7, 2, 5, 8, 1, 3\}$.

costs by a factor of $\sqrt{p}$ over a scheme where communication takes place over all processors.

The communication cost of forming the $\mathbf{R}$ matrix in parallel is the cost of `Scatter` along the processor column. For the sparse vector I distributed to $\sqrt{p}$ diagonal processors, scattering can be implemented with an average communication cost of $\Theta(\alpha \cdot \lg p + \beta \cdot (length(\mathsf{I})/\sqrt{p})$ [7]. This process is illustrated in Figure 3. A similar analysis applies to the construction of the $\mathbf{Q}$ matrix.

The parallel performance of the SpGEMM routine is a complicated function of the nonzero density [1, 4]. Special structure of our matrices, however, make our analysis more precise. We assume that the triple product is evaluated from left to right, $\mathbf{B} = (\mathbf{R} \cdot \mathbf{A}) \cdot \mathbf{Q}$; a similar analysis can be applied to the reverse evaluation. A conservative estimate of $ni(\mathbf{R}, \mathbf{A})$, the number of indices i for which $\mathbf{R}(:, i) \neq \emptyset$ and $\mathbf{A}(i, :) \neq \emptyset$, is $nnz(\mathbf{R}) = length(\mathsf{I})$.

Assuming a uniform distribution of nonzeros, each processor owns nnz/p nonzeros and broadcasts those $\sqrt{p}$ times during SpGEMM execution. We can thus get a rough estimate of the computation and communication costs of the SpGEMM subroutines:

$$T_{comp} \approx \Theta\Big(\frac{nnz(\mathbf{A})}{p} \cdot \lg\big(\frac{length(\mathsf{I})}{p} + \frac{length(\mathsf{J})}{p} + \sqrt{p}\big)\Big)$$

$$T_{comm} = \Theta(\alpha \cdot \sqrt{p} + \beta \cdot \frac{nnz(\mathbf{A})}{\sqrt{p}}).$$

We see that SpGEMM costs dominate the cost of SpRef. The extra $\lg \sqrt{p}$ factor in T_{comp} are due to additions of intermediate triples over $\sqrt{p}$ stages. The bottleneck in scalability is the bandwidth costs, limiting speedup to $\Theta(\sqrt{p})$.

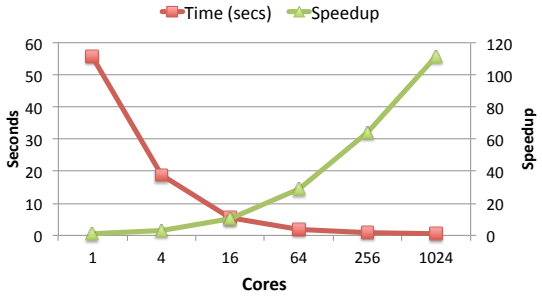


Figure 4: Performance and parallel scaling of applying a random symmetric permutation to an RMAT matrix of scale 22. This operation corresponds to relabeling vertices of a graph. The speedup and timings are plotted on different axes of the same graph.

Experimental Results

We implemented our algorithms within the Combinatorial BLAS framework [3]. We ran benchmarking experiments on NERSC's Franklin system. Our first set of experiments randomly permutes the rows and columns of $\mathbf{A}$, a primitive in parallel matrix computations commonly used for load balancing [8]. In our second set of experiments, we tried to simulate subgraph extraction by generating a random permutation $r = randperm(1 : n)$ and dividing it to $k \ll n$ chunks $r_1, \ldots, r_k$. We then performed k SpRef operations of the form $\mathbf{A}(r_i, r_i)$ one after another (with a barrier in between).

The performance and parallel scaling of the symmetric random permutation is shown in Figure 4. The input is an RMAT matrix [6] of scale 22, having approximately 32 million nonzeros on a matrix with dimensions $2^{22} \times 2^{22}$. We see that scaling is close to linear for small (up to 64) number of processors, and proportional to $\sqrt{p}$ afterwards, in line with our analysis.

The performance of subgraph extraction for $k = 10$ induced subgraphs, each of which has n/k randomly chosen vertices in them, is shown in Figure 5. The algorithm performs similarly well in this case too. The empirical scalability is slightly less than the case of applying a single big permutation, which is expected since we are performing multiple smaller subgraph extractions, increasing span and decreasing available parallelism.

Bibliography

[1] Aydın Buluç and John R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *ICPP'08: Proc. of the Intl. Conf. on Parallel Processing*, pages 503–510, Portland, Oregon, USA, 2008. IEEE Computer Society.

[2] Aydın Buluç and John R. Gilbert. On the representation and multiplication of hypersparse matrices. In *IPDPS'08: Proceedings of the 2008 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–11. IEEE Computer Society, 2008.

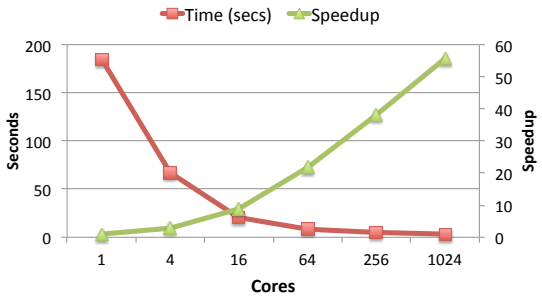


Figure 5: Performance and parallel scaling of extracting 10 induced subgraphs from an RMAT matrix of scale 22. Vertices for each subgraph are chosen randomly. Each subgraph has $n/10$ vertices where n is the matrix dimension. The speedup and timings are plotted on different axes of the same graph.

[3] Aydın Buluç and John R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *The International Journal of High Performance Computing Applications*, to appear, 2010.

[4] Aydın Buluç and John R. Gilbert. Highly parallel sparse matrix-matrix multiplication. Technical Report UCSB-CS-2010-10, Computer Science Department, University of California, Santa Barbara, 2010. `http://arxiv.org/abs/1006.2183`.

[5] Aydın Buluç, John R. Gilbert, and Viral B. Shah. Implementing sparse matrices for graph algorithms. In J. Kepner and J. Gilbert, editors, *Graph Algorithms in the Language of Linear Algebra*. SIAM, Philadelphia. In press.

[6] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *SDM*. SIAM, 2004.

[7] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert A. van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[8] Andrew T. Ogielski and William Aiello. Sparse matrix computations on parallel processor arrays. *SIAM Journal on Scientific Computing*, 14(3):519–530, 1993.

20 The minimum degree ordering with dynamical constraints

Patrick R. Amestoy
(*Université de Toulouse, INPT(ENSEEIHT)-IRIT, France)*

Alfredo Buttari
(*CNRS and INPT(ENSEEIHT)-IRIT, France)*

Abdou Guermouche
(*Université de Bordeaux, France)*

Jean-Yves L'Excellent
(*INRIA and ENS Lyon, France)*

Bora Uçar
(*CNRS and ENS Lyon, France)*

Corresponding Author: **Bora Uçar** (`bora.ucar@ens-lyon.fr`)

We propose a modification of the minimum degree ordering algorithm in which the nodes are constrained to come only after some other nodes. This is close to the minimum degree ordering with constraints (CMMD) algorithm [6]. The difference is that during the course of our algorithm we remove some of the constraints, whereas the constraints are static in the CMMD and its variants. We have modified the AMD [1] algorithm using the code given in [4] to incorporate the dynamical constraints. This AMD code is a fast, state-of-the-art implementation that includes dense variable detection, mass elimination, and supervariable detection. We allow the constrained vertices to be mass eliminated at all times but we do not allow them to form supervariables unless the constraints are removed. Apart from a linear time initialization process, our modifications amounts to visiting the pattern of the original matrix only once throughout the whole ordering process. Therefore, our implementation is also fast and reflects the state of the art.

Such an ordering algorithm can have different applications. We are particularly motivated by the KKT matrices. These matrices have the form $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & O \end{pmatrix}$, where A_{11} is symmetric positive semi-definite, and A_{12} has full rank. Such systems are, in general, factored by performing numerical pivoting using 1×1 and 2×2 pivots. As is well known, the numerical pivoting hampers the performance of the standard factorization algorithms, as the fill-in predicted in the analysis phase turns out to be much less than the amount required in the numerical factorization phase. Therefore, ordering strategies that reduce the fill-in while potentially reducing the need for numerical pivoting are sought. Bridson [3] modifies the minimum degree-based algorithms to order a variable in the second block only after all of its neighbors (which are in the first block) have already been ordered. This has the desirable effect that if A_{11} is definite and A_{12} is full rank then, without any numerical pivoting, an LDL^T decomposition with a diagonal D exists. Scott [7] implements Bridson's method more

efficiently and investigates it. She concludes that Bridson's constraint results in too much fill, and she allows a variable to be eliminated after one of its neighbors in the first block is eliminated.

We implement these two algorithms in a single method by incorporating a threshold test for the eliminated neighbors. Bridson sets this threshold value to the original degree in the matrix; Scott sets the threshold to 1 for all constrained variables. Obviously one needs to strike a balance between these two extremes. Our algorithm performs this using two threshold values. One of them is associated with the number of eliminated neighbors. The other one tries to simulate the numerical factorization and allows a variable to be eliminated if the modification to the associated diagonal entry is deemed to be enough. Clearly the second test is hard to perform accurately, as we do not do the numerical factorization. We therefore propose the following mechanism. Let i be a constrained variable; at the beginning we set $v(i) = \tau_v \sum a_{ij}^2/a_{jj}$ where τ_v is a threshold parameter between 0 and 1. Every time a variable j from the first block is eliminated, we update the $v(\cdot)$ value of its neighbors in the second block, so that $v(i)$ becomes $\tau_v \sum a_{ij}^2/a_{jj} - a_{ij}^2/a_{jj}$. If ever $v(i)$ reduces below zero, we remove the constraint from the variable i. A similar threshold parameter τ_d is also used for the number of eliminated neighbors as well; the constraint on a variable i is removed if $d(i) = \tau_d \times |\{a_{ij} \neq 0\}|$ many neighbors have been eliminated. Therefore, our algorithm becomes that of Bridson's with $\tau_v = \tau_d = 1$. It becomes that of Scott's, with a special τ_d that sets $d(i) = 1$ for all constrained vertices. By playing with τ_v and τ_d one can therefore try to strike a balance between predicted fill-in and the need for numerical pivoting.

Below we present some results with MUMPS [2]. We kept all control variables of MUMPS at default except that we do not allow it to perform any permutation in the analysis phase. We tried both the partial pivoting and static pivoting approaches. In the former case, the number of off-diagonal pivots should be reduced, and the predicted and actual nonzeros in the factors should be close to each other. In the latter case, the number of pivots that are modified should be small as well as the backward error and the number of the iterative refinement steps. We have tried the ordering algorithm with different τ_v and τ_d ($\tau_d = \epsilon$ corresponds to relieving the constraint on a variable as soon as one of its neighbors has been eliminated).

		Partial pivoting		Static pivoting			
matrix	mthd	estNnz	actNnz	nnz	accuracy	it.ref	modPvts
c-69	$\tau_v = 1, \tau_d = 1$	26458644	26458644	26458644	6,29E-11	0	0
	$\tau_v = 1, \tau_d = \epsilon$	3475196	3515889	3475196	4,00E-07	1	141
	$\tau_v = 0.5, \tau_d = 1$	3568779	3599518	3568779	3,64E-07	1	108
stokes128	$\tau_v = 1, \tau_d = 1$	4181157	4346690	4181157	2,74E-05	7	1806
	$\tau_v = 1, \tau_d = \epsilon$	3078088	3526488	3078088	8,14E-10	1	7938
	$\tau_v = 0.5, \tau_d = 1$	3665874	3933954	3665874	3,08E-05	6	3797
ncvxqp5	$\tau_v = 1, \tau_d = 1$	56381037	56544852	56381037	3,54E-08	1	147
	$\tau_v = 1, \tau_d = \epsilon$	49746367	58359444	49746367	6,78E-08	1	1774
	$\tau_v = 0.5, \tau_d = 1$	56735901	58876213	56735901	5,27E-08	1	637

In general, the estimates in the partial pivoting case are tighter as τ_v increases, due to reduced pivoting. For the static pivoting case, smaller number of tiny pivots were replaced as τ_v increases.

We will present more results and try to see the relation with the methods discussed in [5, 8].

Bibliography

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 886–905.

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[3] R. Bridson, *An ordering method for the direct solution of saddle-point matrices*. Preprint.

[4] T. A. Davis, *Direct Methods for Sparse Linear Systems*, no. 2 in Fundamentals of Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[5] A. J. de Niet and F. W. Wubs, *Numerically stable LDL^T-factorization of F-type saddle point matrices*, IMA Journal of Numerical Analysis, 29 (2009), pp. 208–234.

[6] J. W. H. Liu, *The minimum degree ordering with constraints*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 1136–1145.

[7] J. A. Scott, *A note on a simple constrained ordering for saddle-point systems*, Tech. Report RAL-TR-2009-007, RAL, 2009.

[8] M. Tuma, *A note on the LDL^T decomposition of matrices from saddle-point problems*, SIAM Journal on Matrix Analysis and Applications, 23 (2002), pp. 903–925.

21 Optimal Derivative Accumulation Using Integer Programming

Jieqiu Chen and Paul D. Hovland
(*Argonne National Laboratory, United States)*

Robert Luce
(*Technische Universität Berlin, Germany)*

Corresponding Author: **Paul D. Hovland** (hovland@mcs.anl.gov)

Automatic differentiation (AD) is a family of methods for obtaining the derivatives of functions computed by a program [1]. AD couples rule-based differentiation of language intrinsics with derivative accumulation according to the chain rule. The associativity of the chain rule leads to many possible "modes" of combining partial derivatives. The simplest method, the forward mode, combines partial derivatives starting with the independent variables and propagating forward to the dependent variables. The reverse mode combines partial derivatives starting with the dependent variables and propagating back to the independent variables. Exponentially many hybrid, or cross-country, modes are possible. Finding the optimal Jacobian accumulation strategy is NP hard [2]. Therefore, all AD tools employ some sort of heuristic strategy. The most popular heuristics are pure forward mode, pure reverse mode, and a hierarchical strategy using forward mode overall but "preaccumulating" the derivatives of small program units (often statements or basic blocks). Figure 1 gives an example of four strategies for differentiating the simple function segment

```
a = cos(x)
b = sin(y)*y*y
f = exp(a*b)
```

This trivial example is not intended as a complete illustration of automatic differentiation, but rather as an illustration of the differences in accumulation strategies. Real applications have control flow, aliasing, output dependences, intrinsics evaluated at points of nondifferentiability, and thousands or millions of lines of code. AD tools must deal with these realities. Nonetheless, this trivial example illustrates the advantages of hierarchical accumulation strategies. If the number of directional derivatives to be computed (variable `p` in Figure 1) is large, the optimal basic block accumulation strategy ($17 + 3p$ flops) is significantly cheaper than pure forward mode ($9 + 12p$ flops), even for this tiny example.

Algorithms for automatic differentiation are often expressed in terms of the computational graph. Figure 2 shows the computational graph for our simple example. If the edges of the computational graph are assigned weights equal to partial derivatives, then the derivative of a dependent variable v_j with respect to an independent variable v_i is the sum over all paths from v_i to v_j of the product of the edge weights along that path. The optimal Jacobian accumulation problem is reduced to finding an optimal order in which to combine edge weights. A simplified version of this problem is to find an optimal vertex elimination strategy, where a vertex is eliminated by combining all in edges with all out edges (requiring $|\text{in}| \times |\text{out}|$ multiplications).

We have developed an integer linear programming formulation of the optimal vertex elimination problem. This enables us to use a standard integer optimization solver to find an optimal vertex elimination strategy. Our objective is not to replace the elimination heuristics used in AD tools, since finding the optimal elimination strategy for all basic blocks would be prohibitively expensive. Rather, we hope to use the optimization formulation to 1) evaluate the effectiveness of heuristics and 2) find an optimal strategy for certain key computational kernels. In addition to a basic formulation, we have developed some simple lower bounds and symmetry-breaking constraints. Table 1 presents preliminary results for several small test problems and Table 2 shows results for several larger instances.

```
a = cos(x)
  dadx = -sin(x)
  g_a(1:p) = dadx*g_x(1:p)
tmp1 = sin(y)
  d1dy = cos(y)
  g_1(1:p) = d1dy*g_y(1:p)
tmp2 = tmp1*y
  g_2(1:p) = y*g_1(1:p)+tmp1*g_y(1:p)
b = tmp2*y
  g_b(1:p) = y*g_2(1:p)+tmp2*g_y(1:p)
tmp1 = a*b
  g_1(1:p) = b*g_a(1:p)+a*g_b(1:p)
f = exp(tmp1)
  g_f(1:p) = f*g_1(1:p)
```

(a)

```
a = cos(x)
  dadx = -sin(x)
  g_a(1:p) = dadx*g_x(1:p)
tmp1 = sin(y)
  d1dy = cos(y)
tmp2 = tmp1*y
b = tmp2*y
  adjy = y*y*d1dy + y*tmp1 + tmp2
  g_b(1:p) = adjy*g_y(1:p)
f = exp(a*b)
  adja = f*b
  adjb = f*a
  g_f(1:p) = adja*g_a(1:p)+adjb*g_b(1:p)
```

(b)

```
a = cos(x)
  dadx = -sin(x)
  g_a(1:p) = dadx*g_x(1:p)
tmp1 = sin(y)
  d1dy = cos(y)
tmp2 = tmp1*y
b = tmp2*y
  adjy = (tmp1 + d1dy*y)*y + tmp2
  g_b(1:p) = adjy*g_y(1:p)
f = exp(a*b)
  adja = f*b
  adjb = f*a
  g_f(1:p) = adja*g_a(1:p)+adjb*g_b(1:p)
```

(c)

```
a = cos(x)
  dadx = -sin(x)
tmp1 = sin(y)
  d1dy = cos(y)
tmp2 = tmp1 * y
b = tmp2*y
f = exp(a*b)
  adjx = f*a*dadx
  adjy = f*a*(tmp2 + y*(tmp1 + d1dy*y))
  g_f(1:p) = adjx*g_x(1:p)+adjy*g_y(1:p)
```

(d)

Figure 1: Comparison of several strategies for the simple example: (a) pure forward mode, (b) reverse mode statement level preaccumulation, (c) optimal statement level preaccumulation, (d) optimal basic block preaccumulation.

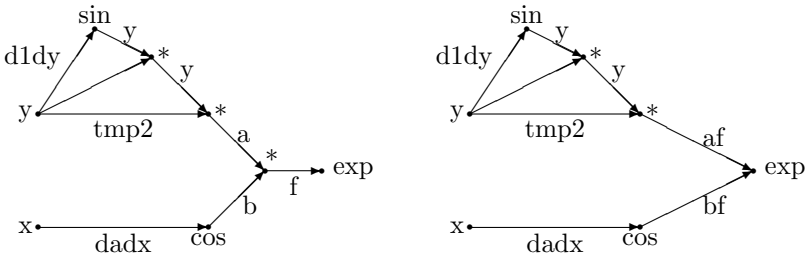


Figure 2: Computational graph (left) for the simple example, (right) after elimination of vertex a*b.

Table 1: Solve times (in seconds) for several small test problems, with and without lower bounds and symmetry-breaking constraints (*** indicates failure to find a provably optimal solution within 10 minutes).

Problem	Indeps	Deps	Total Verts	Opt Cost	Time w/o Consts	Time w/ Consts
fig10.4	4	3	10	18	0.04	0.15
ex10.8	4	3	12	22	0.66	1.07
ex10.8a	4	3	12	30	0.74	1.26
revbound	1	1	12	10	***	8.38
butterfly	4	4	16	48	***	258

Table 2: Solve times (in seconds) for several larger instances.

Problem	Total Verts	Total Edges	Opt Cost	Opt Cost by Solver	Time
seu20_4_1	24	62	58	58	2.34
seu40_4_1	44	124	114	114	37.57
seu80_4_1	84	250	235	235	1531.68
flattenedFDC	89	126	?	124	172.67
flattenedRF	131	204	?	269	24376.72

Bibliography

[1] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, 2000.

[2] U. Naumann, *Optimal Jacobian accumulation is NP-complete*, Math. Prog., 112 (2008), pp. 427–441.

22 Branch and Bound for Optimal Jacobian Accumulation

Viktor Mosenkis, Elmar Peise, and Uwe Naumann
(*RWTH Aachen University, Germany)*

Corresponding Author: **Viktor Mosenkis**
(mosenkis@stce.rwth-aachen.de)

We consider implementations of multivariate vector functions $F : \mathbb{R}^n \to \mathbb{R}^m$ as computer programs mapping a vector of inputs $\mathbf{x} \in \mathbb{R}^n$ onto a vector of outputs $\mathbf{y} \in \mathbb{R}^m$ as $\mathbf{y} = F(\mathbf{x})$. The given implementation of F is assumed to decompose into a *single assignment code* (SAC) as follows:

$$\begin{aligned} &\text{for } j = n+1, \ldots, n+p+m \\ &\quad v_j = \varphi_j(v_i)_{i \prec j} \end{aligned}$$

where $i \prec j$ denotes a direct dependence of v_j on v_i. A directed acyclic graph (DAG) $G = (V, E)$ with integer vertices $V = X \cup Z \cup Y$ and edges $E = \{(i, j) : i, j \in V,\ i \prec j\}$, where $X = \{1, \ldots, n\}$, $Z = \{n+1, \ldots, n+p\}$, $Y = \{n+p+1, \ldots, n+p+m\}$, is induced.

In Algorithmic Differentiation (AD) [2] the DAG G is *linearized* conceptually by attaching the values of local partial derivatives

$$c_{(k,j)} \equiv \frac{\partial \varphi_j}{\partial v_k}(v_i)_{i \prec j}$$

to all edges $(k, j) \in E$ yielding the linearized DAG or l-DAG. The φ_j, $j = n+1, \ldots, n+p+m$, are assumed to be continuously differentiable in a neighborhood of their respective arguments. A graph-based interpretation of the chain rule of differential calculus first presented in [1] yields the following method for accumulating all entries of the Jacobian matrix $A = F'(\mathbf{x}) \in \mathbb{R}^{m \times n}$ of F at some point $\mathbf{x} \in \mathbb{R}^n$. Let $(i, \ldots, j)$ denote a path connecting a vertex i with a vertex j in G and let $A = (a_{j,i})_{i=1,\ldots,n}^{j=n+p+1,\ldots,n+p+m}$. Then

$$a(j, i) = \sum_{(i,\ldots,j)} \prod_{(k,l) \in (i,\ldots,j)} c_{(k,l)} \tag{10}$$

for all sources $i \in \{1, \ldots, n\}$ and sinks $j \in \{n+p+1, \ldots, n+p+m\}$. The Optimal Jacobian Accumulation (OJA) problem aims to minimize the number of fused multiply-add (fma) operations required for the accumulation of the whole Jacobian. This problem is known to be NP-hard [5]. Hence in practice search space reduction heuristics like vertex and edge elimination are used. Both exploit the associativity of the chain rule of differential calculus. Front elimination of an edge (i, j) is defined through the application of Equation (10) to the subgraph induced by (i, j) and all edges emanating from j. New edges (i, j') are introduced for all j' with $j \prec j'$ and $i \not\prec j'$. The new edge labels are set to $c_{(i,j')} = c_{(i,j)} c_{(j,j')}$. If $(i, j') \in E$, then the existing edge

labels are updated according to $c_{(i,j')} = c_{(i,j')} + c_{(i,j)}c_{(j,j')}$ as shown in Fig. 1. Finally (i,j) is removed. If (i,j) is the only in-edge of j (as in Fig. 1) then j is removed as well as all edges emanating from j. Back elimination of (i,j) is

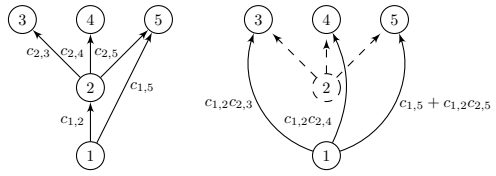
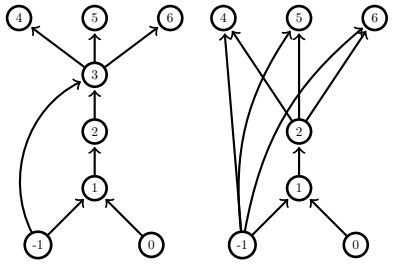


Figure 1: Front elimination of edge $(1,2)$

defined analogously on the subgraph induced by (i,j) and all edges with sink i. The elimination of vertex v is performed by front elimination of all its in- or back elimination of all its out-edges [3]. To accumulate the Jacobian with these elimination techniques one has to perform vertex [edge] eliminations until the resulting graph becomes bipartite. The remaining edge labels correspond to the Jacobian entries. This approach yields the problem of finding a vertex [edge] elimination sequence transforming the l-DAG G into a bipartite graph with minimal number of fmas. We refer to such elimination sequences as optimal. VE [EE] denote the number of fmas performed by an optimal vertex [edge] elimination sequence. For the theoretical investigation of this combinatorial optimisation problems guaranteed results for optimal elimination sequences are essential. To support this research we propose a branch and bound algorithm based on depth first search (DFS). To reduce the search space we use two different ideas. Under certain constraints the permutation of two subsequent eliminations in an elimination sequence does not change the total number of operations. We refer to this technique as *reordering*. Furthermore we exploit lower bounds for the minimum number of operations in order to stop exploring the actual branch of the search tree as soon as possible.



(a) G (b) $G - 3$

Figure 2: DFS with vertex elimination

Consider the example in Fig. 2(a). DFS leads to the search tree in Fig. 3 (edge labels represent the costs of the corresponding vertex elimination). The elimination sequence $[2,1,3]$ yields the smallest costs of 9 fmas. Even without any further bounding strategies there is no need to test the elimination of vertex 1 after the elimination of $[3,2]$ because $\sigma[3,2] = \sigma[2,3,1] = 9$, where σ denotes the cost of an elimination sequence.

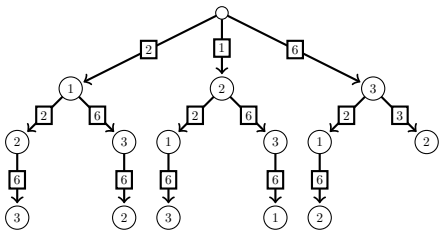


Figure 3: Search tree

Following lower bounds for the minimal cost of vertex and edge elimination sequences are used in the current implementation of the algorithm:

$$VE \geq EE \geq \max\{|Z|, |E^*| - |Z|\} \ , \tag{11}$$

where $E^* = E \setminus \big(X \times Y\big)$ denotes the set of removable edges. In our example the removal of vertex 3 (Fig. 2(b)) involves 6 fmas. Applying Equation (11) delivers a lower bound of 4. Hence none of the elimination sequences starting with the elimination of vertex 3 yield a lower cost than the sequence $[2, 1, 3]$. As a consequence we can stop exploring this branch of the tree. Reordering exploits the fact that for two arbitrary vertices a, b, the identity $G - [a, b] = G - [b, a]$ holds, where $G - [a, b]$ denotes a graph after the application of the elimination sequence $[a, b]$. If a and b are not adjacent then $\sigma[a, b] = \sigma[b, a]$. Hence $[1, 3, 2]$ and $[3, 1, 2]$ have the same costs, implying that we can omit testing the latter. The results of the application of search space reduction techniques to a graph with 15 intermediate nodes are presented in Table 1. We were able to reduce the computation time from ≈600 days to 21,09 minutes. Mostly due to reordering. The currently used lower bound turned out to be not tight enough to speed up the algorithm significantly.

Used Bounding Technique	Time
none (DFS)	≈ 600d
lower bound	≈ 200d
reordering	63,54m
both	21,09m

Table 1: Runtime results

Similar reordering results for edge elimination are more restrictive. The search space for edge elimination is much larger than that of vertex elimination. So far we are able to compute optimal edge elimination sequences only for small problems like the lion or bat graph presented in [4]. Better lower bounds are the subject of ongoing research.

This talk introduces the branch and bound algorithm and presents first runtime results. Moreover we discuss search space reduction techniques for a specific class of DAG's referred as *absorption-free*. Ways to generalize these observations for arbitrary DAG's are presented.

Bibliography

[1] F. L. Bauer. Computational graphs and rounding errors. *SIAM Journal on Numerical Analysis*, 11:87–96, 1974.

[2] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition*. Society for Industrial and Applied Mathematics (SIAM), 2008.

[3] A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markowitz rule. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. SIAM, Philadelphia, PA, 1991.

[4] U. Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Math. Prog.*, 3(99):399–421, 2004.

[5] U. Naumann. Optimal Jacobian accumulation is NP-complete. *Math. Prog.*, 112:427–441, 2006.

23 Cache-oblivious sparse matrix–vector multiplication

Albert-Jan Yzelman and Rob H. Bisseling
(*Utrecht University, Netherlands)*

Corresponding Author: **Albert-Jan Yzelman** (A.N.Yzelman@uu.nl)

In earlier work [3], we presented a one-dimensional (1D) cache-oblivious sparse matrix dense vector multiplication (SpMV) scheme. By reordering unstructured sparse $m \times n$ input matrices A by using only row and column permutations, SpMV $y = Ax$ on the reordered matrix (PAQ, with P and Q permutation matrices) is more efficient in terms of cache utilisation. This reordering is based on a one-dimensional scheme for sparse matrix partitioning, of which the original goal was to efficiently *parallelise* the SpMV; thus specifically for the SpMV, a direct connection between cache misses in the sequential case and communication volume in the parallel case was shown. Partitioners which can be used in the reordering method typically take as parameters (amongst others) the number of processors p and a maximum imbalance parameter $\epsilon \in [0, \infty)$, typically taken small. The 1D method presented is *cache-oblivious*, in the sense that this underlying partitioner is recursive, and that recurring farther than the optimal number of parts p in sequential SpMV ($p \to \infty$) theoretically still retains optimal performance.

At present, parallel applications are more frequently using two-dimensional (2D) partitioning. Extending the work done in the 1D case to the 2D realm is not a trivial task, primarily due to the standard Compressed Row Storage (CRS) datastructure being potentially inefficient when used on a 2D reordered matrix. In recent work [5], we investigate the 2D case; several datastructures are considered and experimented with, leading to block-based datastructures to be theoretically more efficient than CRS or variants of CRS. This block-based datastructure relies on using the 2D reordering method to derive sparse blocks within A, which are then handled in a specific recursive block order, based on the position of the blocks in PAQ. Nonzeroes within each block are then still stored using (a variant of) CRS, although any alternative (auto-)tuned datastructure may also be used.

Hypergraph-based partitioning

We proceed with a short description of the hypergraph-based partitioning methods considered in this work. The sparsity structure of A can be modelled by hypergraphs $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ using various models; examples are the *row-net*, *column-net*, and *fine-grain* [1] models, where respectively matrix columns, rows and nonzeroes are modelled by vertices in $\mathcal{V}$. Matrix rows and columns are modelled by the nets (or hyperedges) in $\mathcal{N}$, when appropiate, and contain a subset of the vertices.

Within all these models, a *partitioning* can be defined as a splitting of $\mathcal{V}$ into $\mathcal{V}_0 \cup \cdots \cup \mathcal{V}_{p-1}$ where all pairs of the p parts $\mathcal{V}_{\{i,j\}}$ are disjoint. Then,

for each net a *connectivity* λ_i over a partitioning $\mathcal{V}_0, \ldots, \mathcal{V}_{p-1}$ of $\mathcal{V}$ can be defined. This enables us to define cut row nets $\mathcal{N}_c^{\text{row}} \subset \mathcal{N}$ as row nets with connectivity larger than one, as well as $\mathcal{N}_c^{\text{column}}$. We consider only recursive bipartitioners which start on the entire set of vertices $\mathcal{V}$. The partitioner in general takes a partitioning of p parts of $\mathcal{V}$, and splits a specific part $\mathcal{V}_i$ thereof into two disjoint sets $\mathcal{V}_i^{\text{new}}, \mathcal{V}_p$. The final goal is to minimise $\sum_{n_i \in \mathcal{N}} (\lambda_i - 1)$ under the constraint $|\mathcal{V}_j| \leq (1 + \epsilon)\text{nz}/p$ for all $j \in [0, p-1]$. The function minimised corresponds exactly to the communication volume in parallel SpMV, and the constraint translates to a load-balance criterion. Note that the fine-grain method is 2D, whereas the row-net and column-net are 1D representations. These 1D representations can be combined during partitioning to obtain another 2D method, which is the Mondriaan partitioning scheme [2].

Cache-oblivious multiplication

For all these hypergraph models we can now describe the connection to matrix reordering in the same uniform way. After each iterative bipartitioning of a vertex set into $\mathcal{V}_{\text{left}}$ and $\mathcal{V}_{\text{right}}$, the hypergraph nets can be divided into several categories: $\mathcal{N}_-^{\text{row}}$, which are nets corresponding to matrix rows containing only nonzeroes from $\mathcal{V}_{\text{left}}$; $\mathcal{N}_c^{\text{row}}$, nets corresponding to matrix rows containing nonzeroes from both $\mathcal{V}_{\text{left}}$ and $\mathcal{V}_{\text{right}}$; and $\mathcal{N}_+^{\text{row}}$, nets corresponding to matrix rows containing only nonzeroes from $\mathcal{V}_{\text{right}}$. Analogously, $\mathcal{N}_c^t extcolumn$ can be defined. The key idea of both 1D and 2D cache-oblivious SpMV is to apply row and column permutations according to these classifications; see Figure 1(left). Note that for the 1D row-net model, the sets $\mathcal{N}_{\{-,c,+\}}^{\text{column}}$ are empty and the figure simplifies to that in [3]. Applying the method in 2D directly according to this earlier work leads to inefficiencies which worsen as p increases, as seen in the middle picture. This is solved in our recent work [5] by applying sparse blocking, as shown in the right picture. Experiments[6] have been performed on an IBM Power6+ architecture, with a 64kB (data) L1 cache, a 4MB L2 cache, and a 32MB L3 cache. Results for large matrices are shown in Table 1.

A different way to obtain a cache-oblivious SpMV is to adapt *only* the sparse matrix storage scheme, and keep the original sparse matrix intact. An idea to use the Hilbert curve to induce an ordering on the nonzeroes during SpMV has been made competitive in our work in [4]. We now seek to combine this method with the 2D reordering, either directly, or within the block datastructure schemes. Another extension is to make use of all data available from partitioning, and perform the cache-oblivious SpMV in parallel, either for distributed or shared-memory.

Bibliography

[1] Ü. V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2D decomposition of sparse matrices. In *Proceedings 8th International Workshop on Solving Irregularly Structured Problems in Parallel*, page 118. IEEE Press, Los Alamitos, CA, 2001.

[6] Reordering was done using the Mondriaan 3.01 software, available at http://www.math.uu.nl/people/bisseling/Mondriaan. SpMVs are performed using the Sparse Library 1.4 software, available at http://www.math.uu.nl/people/yzelman/software.

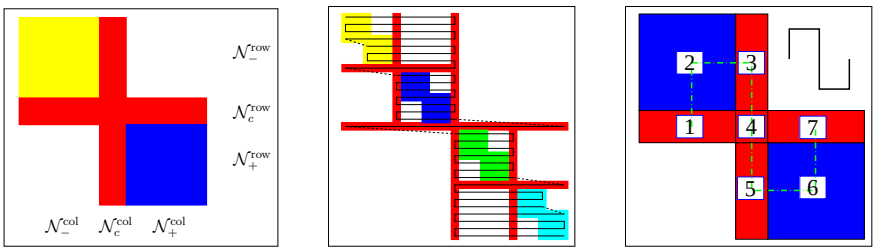


Figure 1: 2D matrix reordering, for $p = 2$ (left). The inefficient nonzero traversal for $p = 4$ using ZZ-CRS ordering (middle) is countered by imposing an ordering on the sparse blocks (right).

Matrix	Original	1D [3]	1D & Blocking
stanford	18.99	9.92 (10)	9.52 (9)
stanford_berkeley	20.93	20.10 (4)	19.26 (9)
cage14	69.36	75.47 (2)	76.48 (2)
wikipedia-2005	248.63	154.93 (10)	142.32 (9)
wikipedia-2006	688.42	378.30 (9)	302.35 (9)

Matrix	2D Mondriaan	2D Fine-grain
stanford	9.35^C (8)	9.73^C (10)
stanford_berkeley	19.18^B (4)	19.41^B (9)
cage14	74.37^C (2)	75.13^C (2)
wikipedia-2005	115.56^C (8)	124.18^B (8)
wikipedia-2006	256.43^C (9)	267.47^B (8)

Table 1: Time of an SpMV multiplication using various schemes. Time is measured in milliseconds. The best running time (averaged over 100 runs) together with the number of parts p resulting in the fastest SpMV is shown. Superscripts indicate whether a block-based (B) or CRS-based (C) datastructure was used (applicable to 2D only, others use CRS variants only).

[2] 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.

[3] A. N. Yzelman and Rob H. Bisseling. Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods. *SIAM Journal on Scientific Computing*, 31(4):3128–3154, 2009.

[4] A. N. Yzelman and Rob H. Bisseling. A cache-oblivious sparse matrix–vector multiplication scheme based on the Hilbert curve. Preprint, 2010.

[5] A. N. Yzelman and Rob H. Bisseling. Two-dimensional cache-oblivious sparse matrix–vector multiplication. Preprint, 2010.

24 Graph Expansion and Communication Costs of Fast Matrix Multiplication

Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz
(*University of California at Berkeley, United States)*

Corresponding Author: **Oded Schwartz** (oded.schwartz@gmail.com)

The communication of an algorithm (e.g., transferring data between the CPU and memory devices, or between parallel processors, a.k.a. I/O-complexity) often costs significantly more time than its arithmetic It is therefore of interest to design and implement algorithms minimizing communication on the one hand, and to obtain lower bounds for the communication needed, on the other hand.

Previous Work. In [2, 3] we generalize the results of [10, 12] regarding matrix multiplication, to attain new I/O-complexity lower bounds for a much wider variety of algorithms (most of the bounds were shown to be tight). This includes algorithms for LU factorization, Cholesky factorization, LDL^T factorization, QR factorization, as well as algorithms for eigenvalues and singular values. Thus we essentially cover all direct methods of linear algebra. The results hold for dense matrix algorithms (most of them are of cubic time), as well as sparse matrix algorithms (whose running time depends on the number of non-zero elements). They apply to sequential and parallel algorithms, to compositions of linear algebra operations (like computing the powers of a matrix), and to certain graph theoretic problems[7].

In [2, 3] we use the approach of [12], based on the Loomis-Whitney geometric theorem [13, 4], by embedding segments of the computation process into a three dimensional cube. This approach, however, is not suitable when distributivity is used, as is the case in Strassen [16] and other fast matrix-multiplication algorithms (e.g., [7, 6]).

The I/O-complexity of classic matrix multiplication and algorithms with similar structure is quite well understood. This is not the case for algorithms of more complex structure. Avoiding the communication of parallel classical matrix multiplication was addressed [5] almost simultaneously with the publication of Strassen's fast matrix-multiplication [16]. Moreover, an I/O-complexity lower bound for the classical matrix-multiplication algorithm is known for almost three decades. Nevertheless, the I/O-complexity of Strassen's fast matrix multiplication and similar algorithms has not yet been resolved.

Communication Cost of Fast Matrix Multiplication.

Upper bound. The I/O-complexity $IO(n)$ of Strassen's algorithm applied to n-by-n matrices on a machine with fast memory of size M, can be bounded above as follows (for actual uses of Strassen's algorithm, see [9, 11, 8]): Run

[7] See [14] for bounds on graph-related problems, and our [3] for a detailed list of previously known and recently designed sequential and parallel algorithms that attain the above mentioned lower bounds.

the recursion until matrices are sufficiently small. Then, read the two input matrices into the fast memory, perform the matrix multiplication inside the fast memory, and write the result into the slow memory. We thus have $IO(n) \leq 7 \cdot IO\left(\frac{n}{2}\right) + O(n^2)$ and $IO\left(\frac{\sqrt{M}}{3}\right) = O(M)$. Thus

$$IO(n) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right). \tag{12}$$

where $\omega_0 = \lg 7$.

Lower bound. In this paper, we obtain a tight lower bound:

Theorem 1. *(Main Theorem) The I/O-complexity $IO(n)$ of Strassen's algorithm on a machine with fast memory of size M is*

$$IO(n) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right). \tag{13}$$

It holds for any implementation and any known variant of Strassen's algorithm. This includes Winograd's $O(n^{\omega_0})$ variant that uses 15 additions instead of 18, which is the most used fast matrix multiplication algorithm in practice [9].

We then extend the lower-bound to a wider class of all stationary (i.e., Strassen-like) and non-stationary but uniform fast matrix-multiplication algorithms, and obtain tight lower bounds for these algorithms as well (where the new value of w_0 corresponds to the relevant algorithm). We also conclude a corresponding lower bound for parallel implementation of these algorithm.

The Expansion Approach. The proof of the main theorem is similar to the one taken by Hong and Kung [10] and is based on estimating the edge expansion of the computation directed acyclic graph *(CDAG)* of an algorithm (where we have a vertex for each input / intermediate / output argument, and edges according to dependencies).

An implementation of an algorithm determines, in the parallel model, which arithmetic operations are performed by which of the p processor. This corresponds to partitioning the corresponding CDAG into p parts. Edges crossing between the various parts, correspond to arguments that are in the possession of one processor, but are needed by other processor, and therefore relate to communication. A corresponding interpretation can be given for the sequential model.

The I/O-complexity is thus tightly connected to the edge expansion properties of this graph. As the graph has a recursive structure, the expansion can be analyzed directly (combinatorially, similarly to what is done in [1]) or by spectral analysis (in the spirit of what was done for the Zig-Zag expanders [15]). There is however, a new technical challenge. While the replacement and Zig-Zag products act similarly on all vertices, this is not the case here: multiplication and addition vertices behave differently when applying a recursive step.

Bibliography

[1] N. Alon, O. Schwartz, and A. Shapira. An elementary construction of constant-degree expanders. *Combinatorics, Probability & Computing*, 17(3):319–327, 2008.

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Communication-optimal Parallel and Sequential Cholesky Decomposition. In *SPAA '09: Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 245–252, New York, NY, USA, 2009. ACM.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing Communication in Linear Algebra. Submitted. Available from http://arxiv.org/abs/0905.2485, 2009.

[4] Y. D. Burago and V. A. Zalgaller. *Geometric Inequalities*, volume 285 of *Grundlehren der Mathematische Wissenschaften*. Springer, Berlin, 1988.

[5] L. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, Bozeman, MN, 1969.

[6] H. Cohn, R. D. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. In *FOCS*, pages 379–388, 2005.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[8] F. Desprez and F. Suter. Impact of mixed-parallelism on parallel implementations of the strassen and winograd matrix multiplication algorithms: Research articles. *Concurrency and Computation: Practice and Experience*, 16(8):771–797, 2004.

[9] C. C. Douglas, M. Heroux, G. Slishman, and R. M. Smith. GEMMW: A portable level 3 BLAS winograd variant of strassen's matrix-matrix multiply algorithm. *Journal of Computational Physics*, 110(1):1–10, 1994.

[10] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, New York, NY, USA, 1981. ACM.

[11] S. Huss-Lederman, E. M. Jacobson, J. R. Johnson, A. Tsao, and T. Turnbull. Implementation of strassen's algorithm for matrix multiplication. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 32, Washington, DC, USA, 1996. IEEE Computer Society.

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

[13] L. H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the AMS*, 55:961–962, 1949.

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

[15] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1):157–187, 2002.

[16] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

25 A Hybrid Parallel Solver Framework for General Sparse Linear Systems

Sivasankaran Rajamanickam, Erik G. Boman, and Michael A. Heroux
(*Sandia National Labs, United States)*

Corresponding Author: **Siva Rajamanickam** (srajama@sandia.gov)

Introduction

As parallelism in a single node increases, computational science applications have to adapt to a hybrid system where each compute node by itself is a shared memory system. This presents new challenges and opportunities for robust sparse linear solvers and preconditioners on the node. For example, domain decomposition based preconditioners can scale better with node level sparse linear solver/preconditioner as the number of subdomains can be limited to the number of nodes (leading to fewer iterations). Even within a node, high performance solvers should account for NUMA based architectures. We present a hybrid solver, ShyLU, for solving general sparse linear systems on the node. ShyLU can also be used as a preconditioner.

Solver Framework

Our approach includes two levels of parallelism, where the top level is based on exploiting bordered block structure. Although such structure sometimes occurs naturally, we use hypergraph partitioning to find such block structure. Solvers that use this idea have been implemented in distributed memory architectures [1]. We use a similar framework for our solver, but limit the subdomains (and MPI tasks) to the NUMA regions in a shared memory node. Each subdomain (diagonal block) can be solved either exactly or ineaxactly. Note that any multithreaded sparse solver can be used here to exploit fine-grain parallelism.

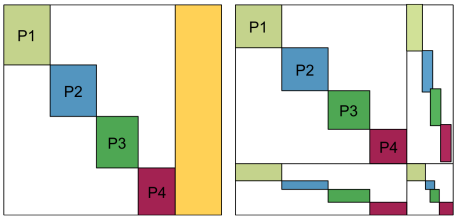


Figure 1: Column ordering and symmetric reordering using hypergraph partitioning.

Figure 1(a) shows a matrix A with column ordering from a hypergraph partition. The hypergraph model in general can handle unsymmetric sparse

matrices. It is also more suitable for partial pivoting on the columns and for a left-looking sparse LU factorization on each of the subdomain (as no row is shared between two subdomains). For symmetric matrices, the column ordering can also be used as a row ordering to get square subdomains, and the reordered matrix will have the structure shown in figure 1(b). Once the subdomains are solved in parallel we can solve for the Schur complement to solve for A. The Schur complement is never explicitly computed. We compute an approximate Schur complement using probing for a predetermined structure and use an inner iteration to solve it. This results in a much smaller system for the iterative methods. We can also replace the solve on the subdomains with incomplete factorization algorithms and use this framework as a node level hybrid preconditioner in a global solve.

Experimental Results

ShyLU is currently implemented as a preconditioner in the Trilinos framework [2] with AztecOO as the iterative solver. The serial solver KLU is used to solve each subdomain, as our multithreaded version is still under development. All the experiments use Zoltan as the hypergraph partitioner to partition for four subdomains (for the NUMA regions/sockets). The tests use symmetric matrices of the size 10K to 20K rows from the UF sparse matrix collection, one matrix from a Sandia application (Tramanto) and a synthetic FEM matrix from MATLAB (wathenLarge).

Trilinos framework has no support for hybrid methods like ShyLU where the iterative method can be used just on the Schur complement. As a result, the experiments iterate over the entire matrix, even when the subdomains were solved exactly using KLU. The results will get better as the software can be adapted to iterate on the Schur complement. Table 1 shows the number of iterations for AztecOO to converge for ShyLU, ML (Algebraic multilevel method) and two incomplete factorizations. A '-' in the number of iterations show that the method did not converge for that test case. ShyLU is as good as or better than the other incomplete factorizations for these tests. The parameters of ILU and ILUT can be modified for better performance than shown here. The experiments shown here uses the default values level of fill zero for ILU and ILUT with level of fill as one and drop tolerance as $1e-12$.

Matrix Name	ShyLU	ML	ILU	ILUT
Cage11	13	14	12	12
cbuckle	101	-	-	-
Lourakis	28	20	42	38
FIDAPex35	16	-	-	-
Oberwolfach	-	27	-	-
fem_3d_thermal	25	23	26	25
Dubkova1	56	55	189	154
Tramanto	112	-	-	-
wathenLarge	35	14	36	37

Observations

Although the hypergraph based nested dissection exposes parallelism for our solver framework, the objective for the partitioning in our problem is different

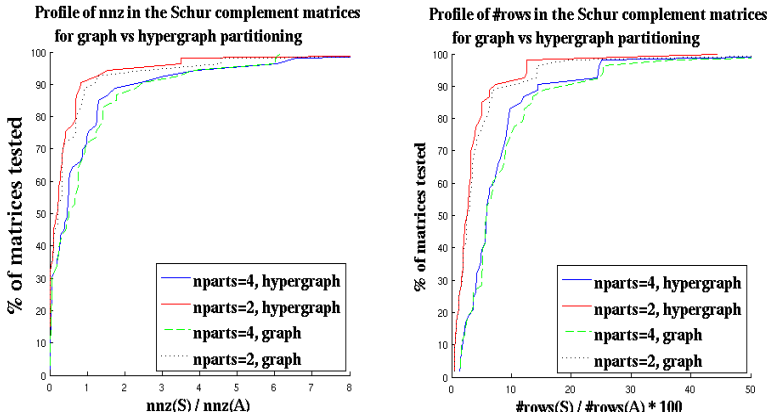


Figure 2: Comparison of the size of the Schur complement for graph and hypergraph partitioning.

than the objective for hypergraph partitioning. The ideal partitioning objective for ShyLU style solver is that subdomains be better balanced, with a small separator and a sparser (or better structured) Schur complement. No partitioners solve this objective directly. However, graph partitioning can be used instead of hypergraph partitioning as the test matrices were symmetric. Figure 2 compares graph (METIS) and hypergraph (Patoh) partitioning with 53 matrices of size 1K to 10K rows from the UF sparse matrix collection. We explicitly compute the Schur complement in MATLAB for these matrices. The improvement in hypergraph partitioner over graph partitioner in terms of the number of nonzeros in the Schur complement is marginal for this set of matrices. However, hypergraph partitioning is best suited for our approach of using a left-looking solver that uses partial pivoting in the subdomains.

Bibliography

[1] I. S. Duff and J. A. Scott. A parallel direct solver for large sparse highly unsymmetric linear systems. *ACM Trans. Math. Softw.*, 30(2):95–117, 2004.

[2] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

26 Combinatorial Problems in a Parallel Hybrid Linear Solver

Ichitaro Yamazaki and Xiaoye Li[†]
(*LBNL, United States)*

François-Henry Rouet
(*ENSEEIHT-IRIT, France)*

Bora Uçar
(*ENS-Lyon. France)*

Corresponding Author: **Ichitaro Yamazaki** (ic.yamazaki@gmail.com)

Introduction. A hybrid linear solver based on the Schur complement method has great potential to be a general purpose solver scalable on tens of thousands of processors. In this method, the original linear system is first partitioned into k subdomain problems using a parallel graph partitioning algorithm. This results in a linear system of the following block structure:

$$\left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \dots & F_k & C \end{array}\right) \left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_k \\ y \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ \vdots \\ f_k \\ g \end{array}\right), \tag{14}$$

where D_ℓ is the ℓ-th subdomain, C consists of the vertex separators, and E_ℓ and F_ℓ are the interfaces between D_ℓ and C. By eliminating the unknowns associated with the subdomains, we obtain the Schur complement system,

$$Sy = \widehat{g}, \tag{15}$$

where $S = C - \sum_{\ell=1}^{k} F_\ell D_\ell^{-1} E_\ell$ and $\widehat{g} = g - \sum_{\ell=1}^{k} F_\ell D_\ell^{-1} f_\ell$. Subsequently, the solution of the linear system (14) can be computed by first solving the Schur complement system (15) for y, then solving the subdomain problem,

$$D_\ell u_\ell = f_\ell - E_\ell y, \tag{16}$$

for u_ℓ. This method provides a framework for developing an effective parallel hybrid solver, where each subdomain problem (16) is solved using a parallel direct solver, while a parallel preconditioned iterative solver is used to solve the Schur complement system (15). However, to achieve high performance, there are several combinatorial problems that need to be resolved. In this talk, we will disucss two such problems.

Partitioning with multiple constraints to extract subdomains. The parallel performance of our hybrid solver is greatly influenced by the initial partitioning to extract the k subdomains. To achieve good load-balance, the subdomains and interfaces need to be balanced (e.g., in terms of their dimensions

or numbers of nonzeros), while the total size of the vertex separators should be small. Even though a nested dissection algorithm is a popular approach for this, the recursive bisections are performed independently from each other, and it often results in an imbalance of the global partition as more subdomains are extracted. To address this, we developed k-way partitioning algorithms, which try to achieve the global constraints directly. Specifically, we first compute a k-way edge partition of the adjacency graph of the global matrix, which typically has well-balanced subdomains. Then, we extract all the edge separators whose end points belong to two different partitions. Finally, vertex separators are extracted within the edge separators by, for example, finding a minimal vertex cover. An important aspect of these algorithms is a heuristic to select the next vertex or edge that is moved from the edge separators to the vertex separators. We found that an effective heuristic is to select the vertex with the largest edge degree from a large subdomain. This allow us to maintain the balance between the subdomains while trying to minimize the size of the vertex separators. We have observed that our k-way partitioning algorihms often obtain partitions that are better balanced than those obtained from the nested dissection algorithm. However, the sizes of the Schur complements are typically increased by factors of about two.

Reordering sparse right-hand sides for triangular solution. To compute a preconditioner for solving (15), our hybrid solver uses SuperLU_DIST [3] to solve two triangular systems $L_\ell G_\ell = E_\ell$ and $U_\ell^T W_\ell = F_\ell^T$ for G_ℓ and W_ℓ, respectively, where $D_\ell = L_\ell U_\ell$ is an LU factorization of the ℓ-th subdomain D_ℓ. For the rest of the discussion, we will focus on the triangular solutions $L_\ell G_\ell = E_\ell$, but the same arguments can be made for $U_\ell^T W_\ell = F_\ell^T$.

To achive high performance, the sparsity of the right-hand sides E_ℓ needs to be exploited. In SuperLU_DIST, the jth columns $E_\ell(:, j)$ and $G_\ell(:, j)$ of E_ℓ and G_ℓ, respectively, are partitioned following the supernode structure of L_ℓ. When fill occurs in a supernodal segment of $G_\ell(:, j)$ during the triangular solution, the segment becomes full from the first nonzero position in this segment to the next supernode boundary. Based on this observation, during the numerical solution, we avoid the operations with explicit zeros by performing the block operations only with the part of the segments under the first nonzeros.

There could be tens to hundreds of thousands of columns in the right-hand sides E_ℓ. To improve the data locality and reduce the message counts, triangular solution is computed for a block of h columns at a time. These h right-hand-side columns within each block must have the same nonzero pattern. Hence, for each supernodal segment of a block, padded zeros are explicitly introduced between the first nonzero position of the entire block and that of the individual column. In order to minimize the number of padded zeros, we developed the following two techniques to reorder the columns of E_ℓ.

1) Reordering based on a postordering of elimination tree We first permute the rows of E_ℓ based on a postorder of the elimination tree of D_ℓ. Then, the columns of E_ℓ are reordered in the increasing order of the row indices of their first nonzeros. The motivation for this reordering is that if the i-th element is the first nonzero of the column $E_\ell(:, j)$, then, according to the Gilbert's path theorem, the fill will be generated in $G_\ell(:, j)$ at the positions corresponding to the nodes on the path from the i-th node to the root of the elimination tree.

Thus, when the right-hand-side columns are reordered based on this postorder, it is likely that the staring nodes of the adjacent columns are close together in the elimination tree, and their paths to the root have large overlap. Even though this technique considers only the first nonzeros of the right-hand sides, we have observed that in comparison to the natural ordering, the number of padded zeros was reduced by a factor of about 1.4 for the matrices arising from accelerator and fusion applications [1, 2].

2) Reordering based on a hypergraph model Consider a partition $\Pi = (\mathcal{V}_1, \ldots, \mathcal{V}_m)$ of the columns of G_ℓ. If r_i is the ith row of G_ℓ, then for a given part $\mathcal{V}_k$, the number of padded zeros in r_i is given by

$$cost(r_i, \mathcal{V}_k) = \begin{cases} |\mathcal{V}_k| - |r_i \cap \mathcal{V}_k| & \text{if } r_i \cap \mathcal{V}_k \neq \emptyset \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

where $|\cdot|$ denotes the cardinality of a set. If Λ_i is the set of the parts that are connected by the row r_i in the partition Π, then the total number of padded zeros is given by

$$cost(\Pi) = \sum_i \sum_{\mathcal{V}_k \in \Lambda_i} (|\mathcal{V}_k| - |r_i \cap \mathcal{V}_k|) = \sum_i (\lambda_i h - |r_i|), \tag{18}$$

where $|\mathcal{V}_k| = h$ for $k = 1, 2, \ldots, m$, and $\lambda_i = |\Lambda_i|$. We do not know any algorithm to minimize (18). However, if we assume that each row r_i contains about h nonzeros, then the cost function (18) is approximated by

$$cost(\Pi) \approx \sum_i h(\lambda_i - 1) \; . \tag{19}$$

This function (19) can be minimized using a row-net hypergraph model. In comparison to postordering, we observed that this hypergraph model reduces the number of padded zeros by a factor of about 1.2.

Bibliography

[1] Community Petascale Project for Accelerator Science and Simulation (ComPASS). https://compass.fnal.gov.

[2] Center for Extended MHD Modeling (CEMM). URL: http://w3.pppl.gov/cemm/.

[3] X. Li and J. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, 2003.

27 Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix

Haim Avron
(*IBM T. J. Watson Research Center, United States)*

Sivan Toledo
(*Tel-Aviv University, Israel)*

Corresponding Author: **Haim Avron** (haimav@us.ibm.com)

Introduction

Finding the trace of an explicit matrix is a simple operation. But there are application areas where one needs to compute the trace of an implicit matrix, that is, a matrix represented as a function. For example, in lattice Quantum Chromodynamics, one often needs to compute the trace of a function of a large matrix, $\texttt{trace}(f(A))$. Explicitly computing $f(A)$ for large matrices is not practical, but computing the bilinear form $x^T f(A)x$ for an arbitrary x is feasible [4, 3]. Other examples include the regularized solution of least-squares problems using the Generalized Cross-Validation approach (see [6]), estimating $\|A\|_F$ and $\left\|A^{-1}\right\|_F$ (using the well-known identity $\|A\|_F^2 = \texttt{trace}(A^*A)$)), and computing the number of triangles in a graph [1].

The standard approach for computing the trace of an implicit function is Monte-Carlo simulation, where the trace is estimated by $\frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i$, where the z_i are random vectors. The original method is due to Hutchinson [6]. Although this method has been improved over the years ([5, 7, 8]), no paper to date has presented a theoretical bound on the number of samples required to achieve an ϵ-approximation of the trace; only the variance of estimators has been analyzed.

We describe four significant contributions to this area:

1. We provide rigorous bounds on the number of Monte-Carlo samples required to achieve a maximum error ϵ with probability at least $1 - \delta$ in several trace estimators. The bounds are surprising: the method with the best bound is not the method with the smallest variance.

2. We provide specialized bounds for the case of projection matrices, which are important in certain applications.

3. We propose a new trace estimator in which the z_is are random columns of a unitary matrix with entries that are small in magnitude. This estimator converges slower than known ones, but it also uses fewer random bits.

4. We experimentally evaluate the convergence of the three methods on a few interesting matrices.

Theoretical results

We study five different trace estimators. They differ in the way random vectors z_i are generated. The question arises: how do we compare the estimators? The easiest way to analyze the quality of trace estimators is to analyze their variance. For any Monte-Carlo estimator R_M we have $Var(R_M) = Var(R_1)/M$ so we only need to analyze the variance of a single sample. This type of analysis usually does not reveal much about the estimator, because the variance is usually too large to apply Chebyshev's inequality effectively.

A better way to analyze an estimator is to bound the number of samples required to guarantee that the probability of the relative error exceeding ϵ is at most δ; this is what we focus on.

Definition 1. *Let A be a symmetric positive semi-definite matrix. A randomized trace estimator T is an* (ϵ, δ)-approximator *of* $\texttt{trace}(A)$ *if*

$$\Pr\left(|T - \texttt{trace}(A)| \leq \epsilon\, \texttt{trace}(A)\right) \geq 1 - \delta\,.$$

A third important metric is the number of random bits used by the algorithm, i.e. the randomness of the algorithm. The trace estimators are highly parallel; each Rayleigh quotient can be computed by a separate processor. If the number of random bits is small, they can be precomputed by a sequential random number generator. If the number is large (e.g., $O(n)$ per Rayleigh quotient), the implementation will require a parallel random-number generator. This concern is common to all Monte-Carlo methods.

Table 1 summarizes the results of our analysis. The table also shows which estimators and analyses are new. The definition of the estimators and the proof of the bounds appear in the full version of the paper. The estimators are also evaluated experimentally (see full version).

In some special cases it is possible to prove better bounds, or even the exact trace. For example, we show that using a Gaussian trace estimator we can compute the rank of a projection matrix (i.e., a matrix with only 0 and 1 eigenvalues) using only $O(\text{rank}(A)\log(2/\delta))$ samples (where δ is a probability of failure; there is no dependence on ϵ). See full version of the paper [2].

Estimator	Variance of one sample	Bound on # samples for an (ϵ, δ)-approx (new)	Random bits per sample (new)
Gaussian	$2\|A\|_F$	$20\epsilon^{-2}\ln(2/\delta)$	infinite; $\Theta(n)$ in floating point
Normalized Rayleigh-quotient (new)	-	$\frac{1}{2}\epsilon^{-2}n^{-2}\operatorname{rank}^2(A)\ln(2/\delta)\kappa_f^2(A)$	-
Hutchinson's	$2\left(\|A\|_F^2 - \sum_{i=1}^n A_{ii}^2\right)$	$6\epsilon^{-2}\ln(2\operatorname{rank}(A)/\delta)$	$\Theta(n)$
Unit Vector (new)	$n\sum_{i=1}^n A_{ii}^2 - \texttt{trace}^2(A)$	$\frac{1}{2}\epsilon^{-2}\ln(2/\delta)r_D^2(A)$ $r_D(A) = \frac{n\cdot\max_i A_{ii}}{\texttt{trace}(A)}$	$\Theta(\log n)$
Mixed Unit Vector (new)	-	$8\epsilon^{-2}\ln\left(4n^2/\delta\right)\ln(4/\delta)$	$\Theta(\log n)$

Table 1: Summary of results: quality of the estimators under different metrics. The proofs appear in the full version of the paper.

Bibliography

[1] Haim Avron. Counting triangles in large graphs using randomized matrix trace estimation. In *Proceedings of KDD-LAvronToledo11DMTA'10*, 2010.

[2] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM*, 59(2), 2011.

[3] Z. Bai, M. Fahey, G. Golub, M. Menon, and E. Richter. Computing partial eigenvalue sum in electronic structure calculations. Technical Report SCCM-98-03, Stanford University, Jan 1998.

[4] Zhaojun Bai, Mark Fahey, and Gene Golub. Some large scale matrix computation problems. *J. Comput. Appl. Math*, 74:71–89, 1996.

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

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

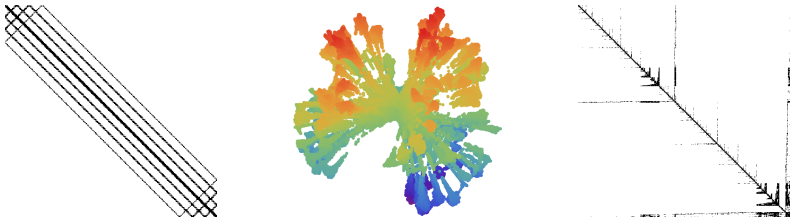
[7] Toshiaki Iitaka and Toshikazu Ebisuzaki. Random phase vector for calculating the trace of a large matrix. *Physical Review E*, 69:057701–1–057701–4, 2004.

[8] Mei Ning Wong, F. J. Hickernell, and Kwong Ip Liu. Computing the trace of a function of a sparse matrix via Hadamard-like sampling. 2004.

28 A Geometric Approach to Matrix Ordering

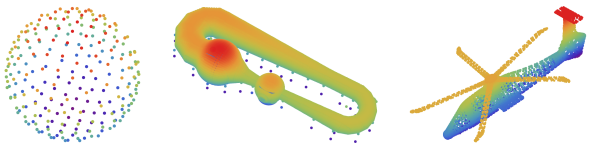
B. O. Fagginger Auer and R. H. Bisseling
(*Utrecht University, Netherlands)*

Corresponding Author: **Bas Fagginger Auer** (`B.O.FaggingerAuer@uu.nl`)



With the increased development and availability of many-core processors (both CPUs and GPUs) it is important to have algorithms that can make use of these architectures. To this end, we present a new recursive hypergraph partitioning algorithm that uses the underlying geometry of the hypergraph to generate the partitioning, largely done using shared-memory parallelism. Hypergraph geometry may either be provided from the problem at hand or generated by the partitioning software. The above illustrates this process for the matrix `twotone` (left): we first generate `twotone`'s geometry (middle), and then use this to permute the matrix (right).

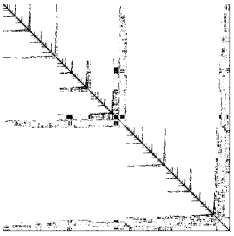
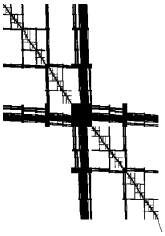
As an example of immediately available geometry we consider the `pothen` collection of matrices, available from [7], which consists of NASA structural engineering matrices collected by A. Pothen. The square pattern matrices from this collection have a natural visual representation: each row/column corresponds to a vertex (the coordinates of which are supplied in a separate file) and each nonzero to an edge between the vertices corresponding to the row and column to which the nonzero belongs. These vertices give an immediate visual representation, as illustrated for `sphere3`, `pwt`, and `commanche_dual` below.



However, not all matrices have an immediate geometric origin (e.g. `twotone`), so we would like to be able to create such a visual representation ourselves if it cannot be provided directly. We do this by generalizing the graph visualization method described in [1] to hypergraphs. For a hypergraph $G = (\{1, \ldots, k\}, \{e_1, \ldots, e_l\})$ (usually a representation of a sparse matrix) with vertex weights $w_1, \ldots, w_k > 0$ and hyperedge costs $c_1, \ldots, c_l > 0$, we determine a visual representation of G by minimizing the energy function f in eqn (20) to obtain positions $x_1, \ldots, x_k \in \mathbf{R}^d$ in d-dimensional space ($d \geq 3$) for all vertices of G. A minimum energy configuration exists if and only if G is connected,

and is approximated by a multilevel gradient stepping method of complexity $\mathcal{O}(k \log(k) + l)$, suitable for shared-memory parallelism [8]. The energy function is chosen in such a way that vertices which are very much interconnected will remain closely together, while loosely connected groups of vertices drift apart. This permits us to isolate clusters of interconnected vertices by applying the `k-means++` algorithm [2] in parallel.

By recursively dividing the vertices of G into two clusters we obtain a recursive bipartitioning of the vertices of the hypergraph G. This can be used to permute sparse matrices to Bordered Block Diagonal (BBD, left for `rhpentium`) or Separated Block Diagonal (SBD, right for `wikipedia-20070206`) form.

$$f(x_1, \ldots, x_k) = \frac{1}{2} \sum_{j=1}^{l} c_j \sum_{i \in e_j} \|x_i - z_j\|^2 + \frac{1}{2} \sum_{j=1}^{k} \sum_{\substack{i=1 \\ i \neq j}}^{k} \frac{w_i \, w_j}{\|x_i - x_j\|^{d-2}}, \qquad (20)$$

$$z_j := \frac{1}{|e_j|} \sum_{i \in e_j} x_i.$$

The quality of the generated bipartitionings is measured in two ways. Firstly, by using the bipartitionings to bring the matrix into recursive SBD form for cache-oblivious sparse matrix–vector multiplication [3, 4]. We compare the permutations generated by the Mondriaan 3.0 hypergraph partitioner to our visual matrix orderings (VMOs) in this context in the top-right table. Here we find the original sparse matrix–vector multiplication time (Orig.) in milliseconds, the best result from [4], which used Mondriaan 3.0, and VMO. On average, generating VMOs was 21.6 times faster than generating permutations using Mondriaan, while the improvement over the original sparse matrix–vector multiplication time is comparable to that of Mondriaan, except for `lp_nug30` and `cage14`. Generating geometry for `lp_nug30` with our multilevel scheme failed to bring out the structure of the matrix and the original layout of `cage14` was already optimized for sparse matrix–vector multiplication (Mondriaan also fails to improve).

Secondly, we used VMO to generate a nested-dissection (recursive BBD) layout for the purpose of LU decomposition [6, 5]. Here, we measure the fill-in (defined as $(nz(L) + nz(U) - nz(I))/nz(A)$ for $A = L\,U$) obtained by SuperLU 4.1 for the VMO, as well as SuperLU's built-in COLAMD, MMD($A^T + A$), and MMD($A^T\,A$) orderings. The results are shown in the table on the bottom-right: on average the VMO fill-in is 1.52 times the lowest fill-in of the other methods. SuperLU's orderings are outperformed by VMO in 8 of the 28 cases. The recursive BBD layout is furthermore suitable for parallelizing the LU decomposition [8].

These examples show that the partitionings generated by this geometrical method are of sufficient quality, with the additional advantage that they can be

Matrix	Orig.	[4]	VMO
ex37	0.116	0.113	0.113
memplus	0.308	0.300	0.280
rhpentium	0.645	0.515	0.646
lhr34	1.37	1.34	1.34
lp_nug30	5.35	4.85	9.15
s3dkt3m2	7.81	7.27	7.80
tbdlinux	6.43	2.36	5.66
stanford	19.0	9.35	5.88
stanford_berkeley	20.9	19.2	22.5
wikipedia-20051105	249	116	128
cage14	69.4	74.4	99.0
wikipedia-20060925	688	256	264

Table 1: Sparse matrix–vector multiplication timings (ms).

Matrix	VMO	CMD	MMD+	MMD×
swang1	**6.2**	7.7	6.7	8.2
lns_3937	**15.0**	17.5	132.2	17.5
poli_large	**1.6**	1.6	1.6	1.6
mark3jac020	68.1	45.6	121.3	**43.9**
fd18	**21.9**	24.1	302.0	25.5
lhr04	6.0	**4.1**	20.6	4.3
raefsky6	**2.7**	3.4	4.5	3.1
shermanACb	19.0	45.3	**14.3**	57.2
bayer04	10.2	4.2	41.8	**4.2**
Zhao2	158.1	115.1	1280.1	**107.0**
mult_dcop_03	3.1	**2.0**	3.4	5.9
jan99jac120sc	71.8	**15.9**	52.4	19.7
bayer01	7.5	**5.4**	47.6	5.6
sinc12	37.8	44.7	**36.3**	45.3
onetone1	32.1	14.4	149.0	**14.2**
mark3jac140sc	**111.0**	125.7	4435.0	152.0
af23560	**24.8**	25.0	82.7	26.9
e40r0100	9.2	9.2	137.5	**8.4**
sinc15	56.3	58.0	**48.7**	57.2
Zd_Jac2_db	9.6	**5.1**	32.1	5.7
lhr34c	7.0	4.7	50.5	**4.7**
sinc18	**65.7**	67.8	68.2	72.3
torso2	10.2	16.8	**8.2**	14.5
twotone	35.8	**15.2**	1448.1	17.0
lhr71c	6.7	4.8	66.4	**4.7**
av41092	64.6	26.0	177.6	**23.8**
bbmat	32.0	**26.7**	1000.6	26.8

Table 2: LU decomposition fill-in.

generated quickly (a 21.6 times speedup over Mondriaan in the first experiment), using methods suitable for shared-memory parallelism.

Bibliography

[1] Y. F. Hu, *Efficient and high quality force-directed graph drawing*, The Mathematica Journal, 2005, 10, 37–71

[2] Arthur, D. and Vassilvitskii, S., *k-means++: the advantages of careful seeding*, SODA '07: Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms, 2007, 978-0-898716-24-5, 1027–1035, New Orleans, Louisiana, SIAM, Philadelphia, PA

[3] A. N. Yzelman and R. H. Bisseling, *Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods*, SIAM J. Sci. Comput., 31, 4, 3128-3154, 2009, 10.1137/080733243

[4] A. N. Yzelman and R. H. Bisseling, *Two-dimensional cache-oblivious sparse matrix–vector multiplication*, 2010, http://www.math.uu.nl/people/yzelman/publications/yzelman10_pre.pdf, Preprint

[5] *L. Grigori and E. G. Boman and S. Donfack and T. A. Davis*, *Hypergraph-Based Unsymmetric Nested Dissection Ordering for Sparse LU Factorization*, SIAM, 2010, SIAM J. Sci. Comput., 32, 6, 3426-3446

[6] *C. Aykanat and A. Pinar and U. V. Çatalyürek*, *Permuting Sparse Rectangular Matrices into Block-Diagonal Form*, SIAM, 2004, SIAM J. Sci. Comput., 25, 6, 1860-1879, http://link.aip.org/link/?SCE/25/1860/1, 10.1137/S1064827502401953

[7] *T. A. Davis and Y. F. Hu*, *The University of Florida Sparse Matrix Collection*, 2010, ACM Trans. Math. Software (to appear), http://www.cise.ufl.edu/ davis/techreports/matrices.pdf

[8] *B. O. Fagginger Auer and R. H. Bisseling*, *A Geometric Approach to Matrix Ordering*, 2011, http://www.staff.science.uu.nl/ faggi101/vmo.pdf, Preprint

29 Reversing the Data Flow of Computer Programs

Johannes Lotz and Uwe Naumann
(*RWTH Aachen University, Germany)*

Corresponding Author: **Johannes Lotz** (lotz@stce.rwth-aachen.de)

Introduction

Adjoint codes play an increasingly important role in scientific computing [2, 3, 4]. Data flow reversal is a fundamental ingredient of adjoint codes and very memory consuming. We aim to find an optimal subroutine argument checkpointing scheme. This combinatorial optimization problem is referred to as the Call Tree Reversal (CTR) problem and is shown to be NP-complete in [5]. For a given amount of persistent memory the objective is to checkpoint arguments of selected subroutine calls such that a reasonable approximate solution of the underlying DAG Reversal problem is obtained. Motivated by the NP-completeness of CTR heuristics are developed and test results are presented.

Background

We consider implementations of multivariate vector functions

$$F : \mathbb{R}^n \to \mathbb{R}^m$$

as computer programs mapping a vector of inputs $\mathbf{x} \in \mathbb{R}^n$ onto a vector of outputs $\mathbf{y} \in \mathbb{R}^m$ as $\mathbf{y} = F(\mathbf{x})$. The given implementation of F is assumed to decompose into a *single assignment code* (SAC) as follows:

$$\begin{aligned} &\text{for } j = n+1, \ldots, n+p+m \\ &\quad v_j = \varphi_j(v_i)_{i \prec j} \end{aligned}$$

where $i \prec j$ denotes a direct dependence of v_j on v_i. The result of each elemental function φ_j is assigned to a unique auxiliary variable v_j. The n inputs $x_i = v_i$, for $i = 1, \ldots, n$, are mapped onto m outputs $y_j = v_{n+p+j}$, for $j = 1, \ldots, m$. This mapping involves the computation of the values of p *intermediate variables* v_k, for $k = n+1, \ldots, n+p$.

The data flow is reversed, that is, the computed values of the intermediate and output SAC variables are accessed in reverse order during the computation of the local partial derivatives of the φ_j in the adjoint code:

$$\begin{aligned} \text{for } i \quad &\prec \quad j \text{ and } j = n+p+m, \ldots, n+1 \\ \bar{v}_i \quad &= \quad \bar{v}_i + \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}(v_k)_{k \prec j}. \end{aligned}$$

The $\bar{v}_j$ are assumed to be initialized to $\bar{y}_j$ for $j = n+p+1, \ldots, n+p+m$ and to zero for $j = 1, \ldots, n+p$. The correctness of this approach follows immediately from the associativity of the chain rule.

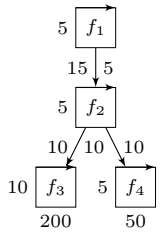
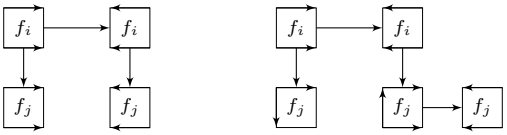


Figure 1: Annotated call tree: $|x_{f_1}| = 5$, $|x_{f_2}| = 5$, $|x_{f_3}| = 10$ and $|x_{f_4}| = 5$; $|SAC^{f_1}| = [15, 5]$, $|SAC^{f_2}| = [10, 10, 10]$, $|SAC^{f_3}| = [200]$ and $|SAC^{f_4}| = [50]$.



(a) $c_{i,j} = 0$ (b) $c_{i,j} = 1$

Figure 2: Call reversal modes

The data section of the program is assumed to be non-persistent and SAC values cannot be expected to be available after the execution of F due to possible overwriting. Therefore all required variables must be stored on a stack (*taped*). SACs induce directed acyclic graphs (DAG). The corresponding DAG REVERSAL problem is shown to be NP-complete in [6].

Call Tree Reversal

For a given implementation of F the SAC is grouped into several subroutines inducing a call tree $\mathcal{T}$ at runtime. For each subroutine f the memory consumption $|x_f|$ of their argument checkpoints and the number of floating-point operations (flops) executed before ($|SAC_0^f|$), inbetween ($|SAC_i^f|$,$i = 1, ..., k-1$) and after ($|SAC_k^f|$) k internal subroutine calls are assumed to be known. The number of flops is assumed to be identical to the memory required to store all intermediate and output SAC variables. This information can be attached to a graphical representation of $\mathcal{T}$ as shown in FIG. 1. Each node in $\mathcal{T}$ corresponds uniquely to a subroutine call. For reverting the data flow in a call tree a subroutine argument checkpointing scheme $\mathcal{C} : E \to \{0, 1\}$ is applied to the set of all edges E. If for an edge $e_{ij} \in E$ connecting f_i and f_j, $c_{i,j} = \mathcal{C}(e_{ij}) = 0$, then the SAC of f_j is taped as part of the tape of f_i. If $c_{i,j} = 1$, then the arguments of f_j are stored on the stack and f_j is executed without taping followed by taping the remainder of f_i. The checkpoint enables an out-of-context evaluation and reversal of f_j when reaching the argument checkpoint during the reversal of f_i (see FIG. 2). The graphical notation for the possible execution modes of a subroutine is given in FIG. 3.

The CTR problem is to find an argument checkpointing scheme $\mathcal{C}$ for $\mathcal{T}$ for a given upper bound $\hat{M}$ on the available persistent memory such that the corresponding reversal tree $\overline{\mathcal{T}}$ is feasible, that is, $M(\overline{\mathcal{T}}) \leq \hat{M}$, and there is no reversal tree $\overline{\mathcal{T}}'$ with a lower operations count $O(\overline{\mathcal{T}}')$ than $\overline{\mathcal{T}}$.

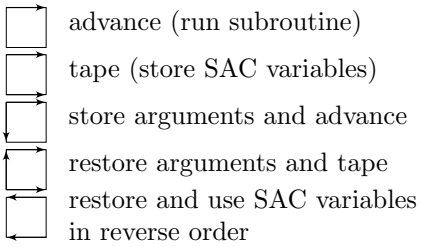


Figure 3: Calling modes for call tree reversal.

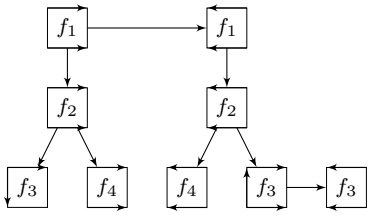


Figure 4: Optimal LMI Solution

Heuristics

We propose two greedy heuristics for CTR. Both start with a reversal scheme having minimal memory consumption, that is (assuming $|x_f| \leq |SAC^f| \ \forall \ f$), $c_{i,j} = 1 \ \forall i, j : e_{ij} \in E$. Step by step $c_{i,j}$ is set to 0 for as long as $M(\overline{\mathcal{T}}) \leq \hat{M}$. The Lowest Memory Increase First heuristic (LMI) prioritizes edges that yield the lowest increase in memory requirement. The Highest Memory Increase First heuristic (HMI) prioritizes edges that yield the highest increase in memory requirement. A solution for the example in Fig. 1 ($\hat{M} = 250$) produced by LMI is given in Fig. 4 ($c_{1,2} = 0$, $c_{2,3} = 1$, $c_{2,4} = 0$). It requires a persistent memory of size $M = 225$ and yields an operation count of $O = 500$.

In Table 1 results of both heuristics are compared on a set of randomly generated call trees. Near-optimal use of the available memory resources gives improvements of 20% to 55% in the operations count.

The formal analysis of both heuristics as well as the development of further heuristics is the subject of ongoing research.

$\|N\|$	$\hat{M}$	LMI	HMI	$\mathcal{C} \equiv 1$
4	800	$M = 713$ $O = \mathbf{1367}$	$M = 774$ $O = 1378$	$M = 687$ $O = 2991$
6	800	$M = 585$ $O = 2101$	$M = 784$ $O = \mathbf{1353}$	$M = 480$ $O = 2562$
8	800	$M = 794$ $O = \mathbf{2300}$	$M = 777$ $O = 2429$	$M = 681$ $O = 3998$
10	1000	$M = 968$ $O = \mathbf{4029}$	$M = 994$ $O = 4156$	$M = 710$ $O = 5083$
12	1000	$M = 898$ $O = \mathbf{3542}$	$M = 976$ $O = 3613$	$M = 559$ $O = 5547$
14	1000	$M = 978$ $O = \mathbf{4530}$	$M = 989$ $O = 4569$	$M = 812$ $O = 8023$

Table 1: Test Results on Random Call Trees

Bibliography

[1] C. Bischof, M. Bücker, P. Hovland, U. Naumann, and J. Utke, editors. *Advances in Automatic Differentiation*, number 64 in LNCSE, Berlin, 2008. Springer.

[2] M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Tools*, number 50 in LNCSE. Springer, 2006.

[3] G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors. *Automatic Differentiation of Algorithms: From Simulation to Optimization*, Computer and Information Science, New York, NY, 2002. Springer.

[4] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition.* Society for Industrial and Applied Mathematics (SIAM), 2008.

[5] U. Naumann. Call tree reversal is NP-complete. In *[1]*, pages 13–22. Springer, 2008.

[6] U. Naumann. DAG reversal is NP-complete. *J. Discr. Alg.*, 7:402–410, 2009.

30 High-Performance Simulation of Isotope Labeling Experiments for Metabolic Flux Estimation

Michael Weitzel, Sebastian Niedenführ, Katharina Nöh, and Wolfgang Wiechert
(*Research Centre Jülich, Germany)*

Corresponding Author: **Michael Weitzel** (`m.weitzel@fz-juelich.de`)

Background. Metabolic Engineering aims to optimize the production capabilities of biotechnologically used organisms. In this context, the term Metabolic Flux Analysis (MFA) covers a family of computational techniques for the *in vivo*-quantification of reaction rates (*fluxes*) in the enzyme-driven, biochemical reaction chains (*metabolic pathways*) of microorganisms and higher cells. Because the result of MFA, the *metabolic flux map*, allows to evaluate the physiological effects of genetic manipulations, MFA became an invaluable tool for Metabolic Engineering.

Currently, the most widely used and reliable variants of MFA are based on carbon labeling experiments. In these experiments a (non-radioactive) ^{13}C labeling substrate $\mathbf{y}_{input}$, e.g. provided by specifically labeled glucose molecules, forms a characteristic labeling imprint in the intermediate products of the metabolism (*metabolites*). The fractional abundances of the metabolites' different labeling patterns is then measured using mass spectrometry (MS) or nuclear magnetic resonance (NMR) devices. Since the details of the ^{13}C enrichment are determined by the enzymes' reaction rates, the measured labeling data can be used to estimate the underlying *in-vivo* reaction rates $\mathbf{v}$.

Methods. ^{13}C-MFA requires the solution of a high-dimensional inverse problem. In particular, an optimization algorithm manages to estimate the reaction rates $\mathbf{v}$ based on a simulation of the labeling imprint $\mathbf{x} = ({}^0\mathbf{x}, {}^1\mathbf{x}, \dots)$ and a subsequent simulation of the MS or NMR measurement equipment, resulting in synthetic measurement values $\mathbf{y}(\mathbf{x})$. Depending on the type of measurements, the simulation of $\mathbf{x}$ is based on either the *Cumomer Method* [1], or the *EMU Method* [2]. Both methods share the same mathematical model which is given by a number of *cascaded* linear equation systems:

$$\begin{aligned} {}^0\mathbf{x} &= \mathbf{1} \\ \mathbf{0} &= {}^k\mathbf{A}(\mathbf{v}) \cdot {}^k\mathbf{x} + {}^k\mathbf{b}\left(\mathbf{v}, \mathbf{y}_{input}, {}^1\mathbf{x}, \dots, {}^{k-1}\mathbf{x}\right). \\ &\quad \text{for } \; k = 1, 2, \dots \end{aligned} \tag{21}$$

These systems are obtained by serializing the relations described by the Cumomer / EMU hyper-graph (a flow network) into balance equations. In particular, the coefficients in the compartmental matrices ${}^k\mathbf{A}$ encode the stoichiometric relations of the reaction network model, whereas the nonlinear terms in ${}^k\mathbf{b}$ formulate the influence of labeling substrate $\mathbf{y}_{input}$. The hyper-graph's nodes are

represented by the unknowns $\mathbf{x} = ({}^0\mathbf{x}, {}^1\mathbf{x}, \dots)$. Each node ${}^k x_i \in {}^k\mathbf{x}$ corresponds to a ^{13}C-labeled molecule fragment consisting of k carbon atoms. Hence, in total, for a metabolite consisting of n carbon atoms 2^n possible fragments exist, from which $\binom{n}{k}$ are balanced in the equation system (21) on cascade level k. Clearly, the dimensions of the equation systems (21) grow exponentially with the number of carbon atoms in the modeled metabolites.

The currently largest metabolic reaction network modeled this way comprises more than 300 different metabolites, which are connected by about 400 biochemical reactions. Because the largest metabolite in this network has 57 carbon atoms, the 57 cascaded equation systems *would* balance around $1.44{\cdot}10^{17}$ different fragments (unknowns), whereas the largest equation system would have dimension $1.5{\cdot}10^{16}$. The corresponding computational problem is clearly beyond the possibilities of today's computers.

The previously used approach for handling this complexity was to restrict the network models to the most important reactions of the central metabolism, containing only small metabolites with few carbon atoms. While the overall performance was still bad, these simplifications limited the quality of the results and, after all, even the applicability of the method. For example, in order to reveal the reliability of an obtained flux estimation a subsequent statistical analysis is necessary. Ideally, this analysis should be based on accurate nonlinear statistics generated by a Monte Carlo bootstrap procedure. However, such a Monte Carlo technique requires thousands of flux estimations, each based on several hundred or thousand simulation runs, in turn.

Results. Newly developed simulation algorithms allow to handle the combinatorial explosion of the system dimensions. Now, without any additional simplifications, even the largest available metabolic reaction network model (including a number of MS measurements) can be simulated in less than a second on a normal desktop computer. This success is based on the combination of two algorithmic techniques [3, 4]:

- Instead of constructing the full Cumomer / EMU network graph only a highly *reduced* network graph is built. These reduced graphs model only those molecule fragments which are both affected by the substrate labeling and required for the simulation of the measurements.

- Reflecting the adjacency of the underlying metabolic network, the resulting balance equation systems (21) are highly sparse. A topological analysis of the network graphs revealed that the sparsity of ${}^k\mathbf{A}$ has to increase with increasing cascade level k. Furthermore the network graphs contain isomorphic subgraphs to great extent. The resulting equation systems (21) are compartmental and therefore diagonal dominant. The observed (sub-)graph isomorphism results in redundancy and isomorphic linear subsystems. A specialized linear solver was implemented which allows to use all these properties.

As a result, the size of the overall computational problem reduces significantly. For the currently desirable measurement configurations and network sizes it was possible to increase the speed of the simulator by four to five orders of magnitude. Based on this progress it is now possible to evaluate nonlinear

statistics and to perform unsupervised flux estimations based on large high-throughput labeling data sets, generated from hundreds of labeling experiments run in parallel.

Bibliography

[1] Wolfgang Wiechert and Michael Wurzel. Metabolic isotopomer labeling systems. Part I: Global dynamic behaviour. *Mathematical Biosciences*, 169(2):173–205, 2001.

[2] Maciek R. Antoniewicz, Joanne K. Kelleher, and Gregory N. Stephanopoulos. Elementary metabolite units (EMU): A novel framework for modeling isotopic distributions. *Metabolic Engineering*, 9(1):68–86, January 2007.

[3] Michael Weitzel. *High Performance Algorithms for Metabolic Flux Analysis*. PhD thesis, University of Siegen, Germany, March 2009. Institute of Systems Engineering, Simulation Group, Department of Electrical Engineering and Computer Science.

[4] Michael Weitzel, Wolfgang Wiechert, and Katharina Nöh. The topology of metabolic isotope labeling networks. *BMC Bioinformatics*, 8(315), 2007.

31 Sampling Graphs with a Prescribed Joint Degree Distribution Using Markov Chains

Ali Pinar
(*Sandia National Labs, United States)*

Isabelle Stanton
(*University of California Berkeley, United States)*

Corresponding Author: **Ali Pinar** (apinar@sandia.gov)

There have been numerous studies about the topological structures of real-world networks, from the Internet to social, biological and technological networks. One common result is the existence of power-law or log-normal distributions over many quantities and in particular the degree distribution: the number of nodes of degree k is proportional to $k^{-\alpha}$. The ubiquity of this distribution has been a motivator for many different generative models, like preferential attachment, the copying model, the Barabasi hierarchical model, forest-fire model, the Kronecker graph model and geometric preferential attachment. Many of these models also match other observed features, such as small diameter or densification. However, recent studies comparing the generative models with real networks on metrics like conductance show that the models do not match other important features of the networks.

Intuitively, if the degree distribution (DD) of a graph describes the probability that a vertex selected uniformly at random will be of degree k then the joint degree distribution (JDD) is the probability that a randomly selected *edge* will be between nodes of degree k and l. Graphs with the same degree distribution may have very different joint degree distributions. For example, the assortativity of a network measures whether nodes prefer to attach to other similar or dissimilar nodes. When similarity is defined in terms of a node's degree, it is a sufficient statistic of the joint degree distribution and measures how different the joint degree distribution is from one where all of the edges are between nodes of the same degree. Studies of the assortativity of networks show that social networks tend to be assortative, while biological and technological networks like the internet tend to be dissortative.

Before attempting to use the joint degree distribution as a metric for designing generative models, it is important to know how tractable it is to work with. Given a joint degree distribution and an integer n, is it possible to construct a graph of size n with that joint degree distribution? Is it possible to construct or generate a *uniformly random* graph with that same joint degree distribution? We address both of these problems in this paper.

Contributions. In this work, we first analyzed the necessary and sufficient conditions for a given joint degree vector to be graphical. We proved that these conditions are sufficient by providing a new constructive algorithm that can generate a graph with any feasible prescribed degree distribution. Next, we introduce a new configuration model for the joint degree matrix problem which is

a natural extension of the configuration model for the degree sequence problem. This new reconfiguration model locally rearranges edges of a graph, such that a new graph is generated that retains the specified joint degree distribution. Finally, using this configuration model, we develop Markov Chains for sampling both pseudographs and simple graphs with a fixed joint degree vector. We prove correctness of both chains and mixing time proofs for the pseudograph chain. The connectedness of the Markov Chain implies that we will be able to generate any graph with the given joint degree distribution, starting from a graph with the same joint degree distribution using our reconfiguration model, and thus we will eventually generate all graphs with the given joint degree distribution, if we keep running of Markov Chain.

The remaining question is how fast the Markov chain mixes, i.e., how fast do we reach a random graph using our reconfiguration model. We have performed an empirical study to observe how fast our Markov chain mixes. For our detailed studies, we restricted ourselves to only small graphs on the order of 100 vertices. Note that even if the graph is small, the size of the Markov chain can be enormous. Our experiments showed that the following results.

- We checked the auto correlation value for each edge. Intuitively, an inherent problem with a Markov Chain method is that successive states generated by the chain may be highly correlated. If we were able to draw independent samples from the stationary distribution, then the autocorrelation of that set of samples with itself would go 0 as the number of samples increased. The autocorrelation time is capturing the size of the gaps between sampled states of the chain needed before the autocorrelation of this 'thinned' chain is very small. If the thinned chain has 0 autocorrelation, then it must be exactly sampled from the stationary distribution. In practice, when estimating the autocorrelation from a

  nite number of samples, we do not expect it to go to exactly 0, but we do expect it to 'die away' as the number of samples and gap increases.

  Our experiments showed that the autocorrelations times dies away exponentially, which shows the Markov Chain mixes very fast.

- Given a joint degree distribution, we can compute the mean of edge, i.e., the ratio of number of graphs with this edge to all graphs with the same degree distribution. We used the mean of an edge as another metric to test the mixing time of the Markov chain. Our experiments showed that the sample mean approached the real mean, and this happened very rapidly.

- As the final set of experiments, we investigated the relationship between the real mean of an edge and the autocorrelation times. Our results showed that the two metrics were related, and the autocorrelation were smaller as the means were closer to 0 or 1, and larger when the mean as around 0.5. To better observe this relationship, we designed a graph that has edges whose means cover the [0,1] interval. Experiments on this graph confirmed our initial observations that the auto correlations are inversely proportional to the absolute value of (edge mean- 0.5). We are currently working on how to exploit this observation for faster mixing times.

32 On fill and flops in symmetric Gaussian elimination

Robert Luce
(*TU Berlin, Germany)*

Corresponding Author: **Robert Luce** (luce@math.tu-berlin.de)

Let $A \in \mathbb{R}^{n,n}$ be a sparse symmetric positive definite matrix and consider the Cholesky factorization $LL^T = PAP^T$ where P is a permutation matrix and L a unit lower triangular matrix. The nonzero pattern of L is a superset of the nonzero pattern of the lower triangular part of A, and the number of additional nonzero elements, called *fill elements*, depends only on the choice of P. Another metric of interest is the number of floating point operations (flops) needed to construct the Cholesky factor L; this number depends only on the permutation matrix P, too. In this work we investigate the relationship of these two metrics.

The symmetric sparse elimination process can be described by the well-known graph theoretic model developed by Rose [2], from which we adopt terminolgy here. Let $G := G_A = (V, E)$ be the graph of A, $\alpha : \{1, \ldots, n\} \to V$ an elimination ordering and $d(\alpha(i))$ the degree of $\alpha(i)$ at the time of its elimination. We set P to be the permutation matrix correspoding to α. If $LL^T = PAP^T$ is the Cholesky factorization of PAP^T and l_i denotes the number of nonzeros in the i-th column, the two optimization problems from the metrics above can be described as

$$\text{minfill}(G) := \arg\min_{\alpha} \sum_{i=1}^{n} d(\alpha(i)) + 1 = \sum_{i=1}^{n} l_i \tag{22}$$

$$\text{minflop}(G) := \arg\min_{\alpha} \sum_{i=1}^{n} (d(\alpha(i)) + 1)^2 = \sum_{i=1}^{n} l_i^2. \tag{23}$$

In (22) we minimize over the total number of nonzeros in L instead of the fill elements only, which is an equivalent optimization formulation (the two problems are *not* equivalent in an approximation sense, though).

For chordal graphs, the two optimization problems are equivalent and

$$\text{minfill}(G) = \text{minflop}(G) = \{\alpha \mid \alpha \text{ is a PEO}\}. \tag{24}$$

whereas not much seems to be known about the relation of the orderings that attain the minima in the two optimizations above.

In figure 1 a very simple example is given where $\text{minflop}(G) \subsetneq \text{minfill}(G)$. The graph is not chordal, but after the first elimination step, the remaining graph is chordal, so after having chosen $\alpha(1)$, we can always extend α by a PEO. Since all vertices of $\{v_1, v_2, v_3\}$ and all of $\{s_1, s_2, s_3\}$ have the same neighborhoods, we only need to analyze the cases where w, v_1 and s_1 are the first nodes to be eliminated. No matter which one we choose, the resulting number of nonzeros in L is always 25, whereas the minimum number of flops (105) is only attained if we start at a node s_1 (or s_2, s_3).

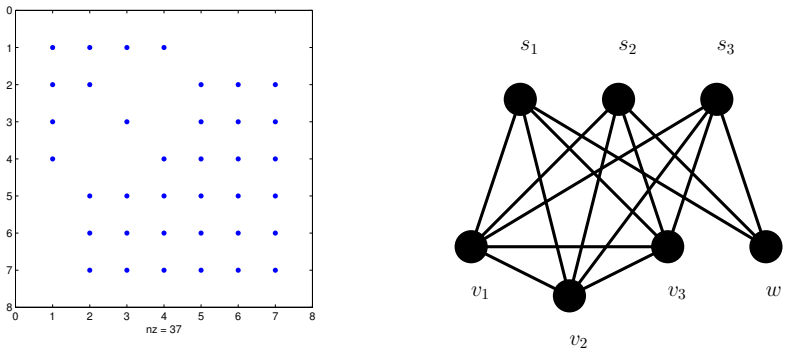


Figure 1: A matrix and its graph G for which $\text{minflop}(G) \subsetneq \text{minfill}(G)$

Analyzing a class of graphs given by Kloks [1] to show that minimum fill and minimum tree width are different, we show that the sets of optima in (22) and (23) may in fact be disjoint.

Bibliography

[1] Ton Kloks. *Treewidth*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994. Computations and approximations.

[2] Donald J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970.

33 On Kronecker Canonical Form of Mixed Matrix Pencils

Satoru Iwata
(*Kyoto University, Japan)*

Mizuyo Takamatsu
(*Chuo University, Japan)*

Corresponding Author: **Mizuyo Takamatsu**
(takamatsu@ise.chuo-u.ac.jp)

Introduction

A *matrix pencil* is a polynomial matrix in which the degree of each entry is at most one. We express a matrix pencil as $D(s) = sX + Y$ by a pair of constant matrices X and Y. A matrix pencil $D(s)$ can be brought into its *Kronecker canonical form* by an equivalence transformation with constant nonsingular matrices. The Kronecker canonical form plays an important role in many applications such as control theory and differential-algebraic equations.

Matrix pencils arising in practice are often very sparse, and it is tempting to exploit their combinatorial structures. The Kronecker canonical form is a block diagonal matrix which consists of nilpotent blocks, rectangular blocks, and a residual square block. Among them, nilpotent blocks admit two combinatorial characterizations. The first one utilizes the highest degree of subdeterminants. The second characterization is based on the ranks of larger constant matrices, called *expanded matrices*. Under the genericity assumption that the set of nonzero coefficients is algebraically independent, it is shown in [3] that the rank of the expanded matrix coincides with the maximum weight of a matching in a bipartite graph. In this paper, we extend these results to *mixed matrix pencils* by using matroid theory.

The genericity assumption is justified by the fact that physical characteristics in engineering systems are not precise in values because of measurement noise. However, it is not always valid in practical situations. In fact, exact numbers do arise, for instance, in conservation laws. This observation led Murota and Iri to introduce the notion of a *mixed matrix* [4].

A mixed matrix is a constant matrix that consists of two kinds of numbers as follows.

Accurate Numbers Numbers that account for conservation laws are precise in values. These numbers should be treated numerically.

Inaccurate Numbers Numbers that represent physical characteristics are not precise in values. These numbers should be treated combinatorially as nonzero parameters without reference to their nominal values. Since each such nonzero entry often comes from a single physical device, the parameters are assumed to be independent.

In order to deal with dynamical systems, it is natural to consider the matrix pencil version, which is called a *mixed matrix pencil*.

In this paper, we prove that the computation of the ranks of expanded matrices for mixed matrix pencils reduces to solving *independent matching problems*. This leads to an algorithm for determining nilpotent blocks of a mixed matrix pencil. Since the independent matching problem in this case is equivalent to a linear matroid intersection problem, we can use the efficient algorithms proposed in [1, 2].

Kronecker Canonical Form

A matrix pencil is known to be strictly equivalent to its Kronecker canonical form

where J_ν is a $\nu \times \nu$ constant matrix, N_μ and L_ϵ are $\mu \times \mu$ and $\epsilon \times (\epsilon + 1)$ matrix pencils defined by

$$\text{block-diag}(sI_\nu + J_\nu, N_{\mu_1}, \ldots, N_{\mu_d}, L_{\epsilon_1}, \ldots, L_{\epsilon_p}, L_{\eta_1}^\top, \ldots, L_{\eta_q}^\top, O),$$

$$N_\mu = \begin{pmatrix} 1 & s & 0 & \cdots & 0 \\ 0 & 1 & s & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & s \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}, \; L_\epsilon = \begin{pmatrix} s & 1 & 0 & \cdots & 0 \\ 0 & s & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & s & 1 \end{pmatrix},$$

and block-diag$(D_1, \ldots, D_b)$ denotes the block-diagonal matrix pencil with diagonal blocks $D_1, \ldots, D_b$. The numbers ν, d, p, q, $\mu_1, \ldots, \mu_d$, $\epsilon_1, \ldots, \epsilon_p$, $\eta_1, \ldots, \eta_q$ are uniquely determined. The matrices $N_{\mu_1}, \ldots, N_{\mu_d}$ are called the *nilpotent blocks*, and the numbers $\mu_1, \ldots, \mu_d$ $(\mu_1 \geq \cdots \geq \mu_d > 0)$ are called the *indices of nilpotency*.

For an $m \times n$ matrix pencil $D(s) = sX + Y$, we consider a $km \times kn$ matrix $\Theta_k(D)$ defined by

$$\Theta_k(D) = \begin{pmatrix} X & O & \cdots & \cdots & O \\ Y & X & \ddots & & \vdots \\ O & Y & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & X & O \\ O & \cdots & O & Y & X \end{pmatrix}.$$

The rank of $\Theta_k(D)$ is expressed by

$$\operatorname{rank} \Theta_k(D) = rk - \sum_{i=1}^{d} \min\{k, \mu_i\},$$

where r denotes the rank of $D(s)$. Therefore, the indices $\mu_1, \ldots, \mu_d$ are determined by the ranks of $\Theta_k(D)$ for $k = 1, \ldots, r$. In this paper, we analyze $\operatorname{rank} \Theta_k(D)$ for a mixed matrix pencil $D(s)$ in order to obtain $\mu_1, \ldots, \mu_d$.

Ranks of Expanded Matrices

A matrix pencil $D(s)$ is called a *mixed matrix pencil* if $D(s)$ is given by $D(s) = (sX_Q + Y_Q) + (sX_T + Y_T)$ that satisfy the following two conditions.

$$\Theta_3(D) = \begin{pmatrix} \bar{Q} \\ \bar{T} \end{pmatrix} = \begin{array}{c|ccc} & C_1 & C_2 & C_3 \\ \hline & X_Q & & \\ & Y_Q & X_Q & \\ & & Y_Q & X_Q \\ \hline R_1^T & X_T & & \\ R_2^T & Y_T & X_T & \\ R_3^T & & Y_T & X_T \end{array}$$

Figure 1: An expanded matrix $\Theta_3(D)$.

(MP-Q) X_Q and Y_Q are constant matrices.

(MP-T) Nonzero entries in X_T and Y_T are independent parameters.

For the sake of simplicity, we focus on a mixed matrix pencil given by $D(s) = \binom{sX_Q+Y_Q}{sX_T+Y_T}$. The general case can be reduced to this special case.

By using $\bar{Q} = \Theta_k(sX_Q + Y_Q)$ and $\bar{T} = \Theta_k(sX_T + Y_T)$, we denote $\Theta_k(D)$ by $\binom{\bar{Q}}{\bar{T}}$. Let us denote the hth column set of $\Theta_k(D)$ by C_h, and the hth row set of $\bar{T}$ by R_h^T for $h = 1, \ldots, k$. We denote the copy of C_h by C_h^Q. These notations are summarized in Figure 1.

We define a bipartite graph $G(\Theta_k(D)) = (\bar{V}^+, \bar{V}^-; \bar{E})$ with

$$\bar{V}^+ = \bigcup_{h=1}^{k} C_h^Q \cup \bigcup_{h=1}^{k} R_h^T, \quad \bar{V}^- = \bigcup_{h=1}^{k} C_h,$$

$$\bar{E} = \bigcup_{h=1}^{k} E_h^Q \cup \bigcup_{h=1}^{k} E_h^X \cup \bigcup_{h=1}^{k-1} E_h^Y,$$

where $E_h^Q = \{(i_h^Q, i_h) \mid i_h^Q \in C_h^Q, i_h \in C_h\}$, and E_h^X and E_h^Y correspond to the set of nonzero entries in X_T and Y_T. Figure 2 illustrates $G(\Theta_3(D))$ for

$$D(s) = \left(\begin{array}{cccc} s & 0 & 0 & 1 \\ 0 & 1 & s & s \\ \hline t_1 s & 0 & 0 & t_2 s \\ t_3 & t_4 & 0 & 0 \\ 0 & 0 & t_5 s & t_6 \end{array}\right)$$

with independent parameters $t_1, \ldots, t_6$.

Let $\mathbf{M}^+ = (\bar{V}^+, \mathcal{I}^+)$ be the linear matroid represented by $\bar{Q}$. We consider the following independent matching problem.

[IMP($\Theta_k(D)$)]
Find a matching $M \subseteq \bar{E}$ that maximizes $|M|$ subject to $\partial^+ M \in \mathcal{I}^+$, where $\partial^+ M$ denotes the set of vertices in $\bar{V}^+$ incident to M.

Our main result is as follows.

Theorem 1. *For a mixed matrix pencil $D(s) = \binom{sX_Q+Y_Q}{sX_T+Y_T}$, the rank of $\Theta_k(D)$ coincides with the optimal value of* **IMP($\Theta_k(D)$)**.

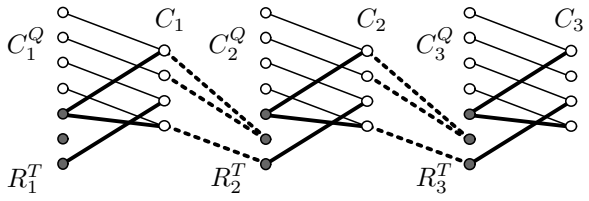


Figure 2: A graph $G(\Theta_3(D))$ with E_h^Q (solid line), E_h^X (heavy line), and E_h^Y (dotted line).

Bibliography

[1] H. N. Gabow and Y. Xu, *Efficient theoretic and practical algorithms for linear matroid intersection problems*, J. Comput. Syst. Sci., 53 (1996), pp. 129–147.

[2] N. J. A. Harvey, *Algebraic algorithms for matching and matroid problems*, SIAM J. Comput., 39 (2009), pp. 679–702.

[3] S. Iwata and R. Shimizu, *Combinatorial analysis of singular matrix pencils*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 245–259.

[4] K. Murota, *Matrices and Matroids for Systems Analysis*, Springer-Verlag, Berlin, 2000.

34 Overlapping clusters for distributed computation

David F. Gleich
(*Sandia National Labs, United States)*

Reid Andersen
(*Microsoft, United States)*

Vahab Mirrokni
(*Google, United States)*

Corresponding Author: **David F. Gleich** (dfgleic@sandia.gov)

We want to design scalable algorithms that address communication problems endemic in distributed computations on graphs using a set of *overlapping clusters*, or vertex partitions that intersect. With overlapping clusters, we store more of the graph than required. The framework then affords the ease of implementation of vertex partitioning, and by removing the minimum storage constraint, our hope is that this technique allows us to reduce communication in a distributed computation. In our experiments, we observe a drastic reduction in communication volume for geometric graphs and some reduction in communication volume for information networks.

Related work

Our proposal for overlapping clusters is novel in its implementation, but not so in concept. These have been studied for a long time in the field of domain decomposition. There, overlapping domains are used to solve a partial differential equation (PDE) using a Schwarz method. These ideas have been generalized to solve many linear systems $Ax = b$ in either an *additive* or *multiplicative* Schwarz method [5]. We investigate using an *additive* Schwarz method method to solve a linear system with our overlapping clusters. These techniques have been applied to solving PageRank [2]. Recently [3] investigated generating overlap from an existing graph partition. Another related idea is the notion of a communication avoiding algorithm. These substitute local computation for communication. For example, [4] create overlap among the vertices managed by each processor to reduce the communication required for k consecutive sparse matrix-vector products. A key difference between our work and the two previous projects is that they start with a partitioning and add overlap; instead, we build a set of overlapping clusters and add a mapping from vertices to clusters.

Theoretical support for overlap

Let $G = (V, E)$ be a graph. An overlapping clustering $(\mathcal{C}, \tau)$ consists of a collection $\mathcal{C}$ of *clusters*, which are subsets of V, and a *mapping* $\tau : V \to \mathcal{C}$ that associates each vertex $v \in V$ with a single cluster $\tau(v) \in \mathcal{C}$. We require that

the collection $\mathcal{C}$ cover the graph, and that each vertex v be contained in its associated cluster $\tau(v)$.

Given a maximum volume MaxVol as an upper bound for the volume of a graph subset stored on each machine (the sum of degrees of all the vertices), we need to divide a large graph into several clusters and store each cluster on one machine. Define a random walk process, $X_0, \ldots$ in the usual way. We can simulate the random walk process in a graph G using a collection $(\mathcal{C}, \tau)$ of overlapping clusters. For each time step t let C_t be the current active cluster. The current vertex X_t is some vertex in V, and the current active cluster C_t is some cluster in $\mathcal{C}$ that contains X_t. Initially, X_0 is a specified starting vertex, and $X_0 = \tau(X_0)$. To advance to the next time step, the active cluster C_t chooses the next vertex X_{t+1} uniformly at random from the neighbors of X_t. If the new vertex X_{t+1} is contained in C_t, then the current cluster remains the active cluster. If the new vertex X_{t+1} is not contained in C_t, then the cluster $\tau(X_{t+1})$ becomes the active cluster. Our first goal is to minimize the number of times the active cluster C_t must be changed during the simulation of the walk. A good collection $\mathcal{C}$ will allow us to simulate a random walk without constantly switching between clusters. We define $\text{Swaps}(X_0, \ldots, X_T)$ to be the number of times the active cluster C_t changes during the walk $X_0, \ldots, X_T$, and we define ρ_∞ as the mean of the expected limiting swapping probability over all vertices. This setup allows us to prove the following theorem.

Theorem 1. *Consider a large cycle C_n of $n = M\ell$ nodes for a large number $M > 0$, and let the maximum volume of a cluster* MaxVol *be ℓ. Let P be the optimal partitioning of G to non-overlapping clusters of size at most* MaxVol *and ρ^*_∞ be the swapping probability of P. There exists an overlapping cover with* TotalVol *of 2*Vol(G) *whose swapping probability ρ'_∞ is less than $\frac{\rho^*_\infty}{\Omega(\text{MaxVol})}$.*

Also, we show that finding a set of clusters to minimize ρ_∞ is NP-hard and also hard to approximate. This follows via a reduction from the minimum bisection problem.

Finding good clusters

We detail a heuristic algorithm to find a good set of overlapping clusters. The algorithm roughly works as follows.

Identify candidate clusters. At first, we find clusters with maximum volume MaxVol and small conductance with a PageRank clustering heuristic [1] or METIS.

Compute well-contained sets. For each cluster, we compute a containment score for each vertex. The vertices with highest containment are assigned as *cores* of the cluster. A random walk from a core vertex should take a while to leave the cluster.

Cover with cluster cores. We now find a subset of the candidates based on three measures: (i) the total volume shouldn't be too large; (ii) the sum of the cut sizes should be small; and (iii) for each node v, we should pick a cluster C such that v is in the core of C. This will yield a set-cover problem, which we use a greedy approximation algorithm to solve.

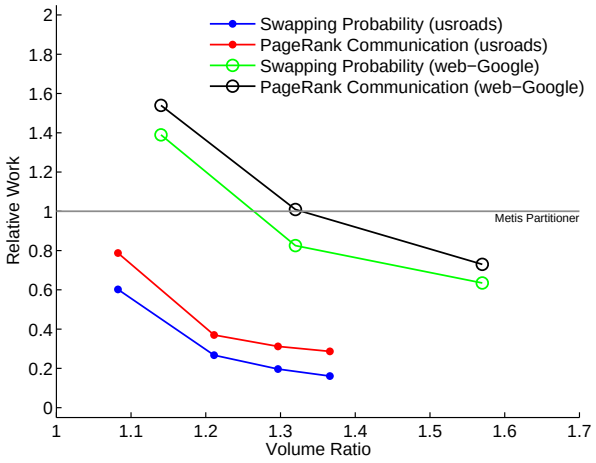


Figure 1: Overlapping performance

Combine clusters. Finally, we combine any small clusters until the final size of each is about MaxVol.

Results

In our experiments, we evaluate clusters from our algorithms on two measures: (i) the swapping probability of a random walk, computed via simulation; and (ii) the communication required for solving PageRank via an additive Schwarz method. At right, we present results from two networks with around 150,000 vertices (us-roads) and 800,000 vertices (web-Google). The horizontal axis is the amount of extra storage required and the vertical axis is the relative work compared to the METIS partitioner. This exemplifies how overlap can help a distributed computation; although we do not always get such great results for information networks. Let us explicitly note that we only have a proof of concept demonstration that overlap can help, which only simulates communication. Because of this nature and that many of our results appear to depend strongly on the individual properties of the graph, we make our codes and experiments available for others to use and reproduce: `https://dgleich.com/projects/overlapping-clusters`

Bibliography

[1] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using PageRank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006.

[2] Rafael Bru, Francisco Pedroche, and Daniel B. Szyld. Cálculo del vector PageRank de google mediante el método iterativo de schwarz (computation of google's pagerank vector with the schwarz iterative method). In J.L. Pérez Aparicio et al., editor, *Congreso de Métodos Numéricos en Ingeniería (Proceedings of the Congress on Numerical Methods in Engineering)*, Granada, Spain, 2005. In Spanish.

[3] David Fritzsche. Fast graph partitioning and applications to preconditioning. Presentation at SIAM Combinatorial Scientific Computing 2009, October 2009.

[4] Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. Minimizing communication in sparse matrix solvers. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, November 2009. ACM.

[5] Daniel B. Szyld and Andreas Frommer. Weighted max norms, splittings, and overlapping additive schwarz iterations. *Numerische Mathematik*, 83:259–278, 1999.

35 Multiscale approach for network compression-friendly ordering

Ilya Safro
(*Argonne National Laboratory, United States)*

Corresponding Author: **Ilya Safro** (safro@mcs.anl.gov)

Finding a suitable compressed representation of large-scale networks has been intensively studied in both practical and theoretical branches of data mining [6, 1, 5, 3, 10]. In particular, the success of applying some of the recently proposed compression schemes [5, 3, 2] strongly depends on the "compression-friendly" arrangement of network nodes. Usually, the goal of these arrangements is to order the nodes such that the endpoints of network links (edges) are located as close as possible. Doing so leads to a more compact representation of links and allows a better performance of compression schemes and network element access operations. In [5], Chierichetti et al. propose a combinatorial optimization problem, namely, *the minimum logarithmic arrangement problem* (MLogA), that minimizes the number of bits in gap encodings of edges stretched between their endpoints.

A network is described by a weighted directed graph $G = (V,\ E)$, where $V = \{1, 2, ..., n\}$ is the set of nodes and E is the set of directed edges. If $ij \in E$, then there exists an edge $i \to j$. Denote by w_{ij} the nonnegative weight of the directed edge ij between nodes i and j; if $ij \notin E$, then $w_{ij} = 0$. We generalize MLogA and formulate the link- and node-weighted version of MLogA (GMLogA); each vertex i is assigned with a *volume*, denoted v_i. The task is to minimize the cost $c(G, x)$, namely,

$$\begin{aligned} \min_{\pi} \quad & \mathsf{c}(G,x) = \sum_{ij \in E} w_{ij} \lg |x_i - x_j| \\ \text{such that} \quad & x_i = \frac{v_i}{2} + \sum_{k, \pi(k) < \pi(i)} v_k \end{aligned} \tag{25}$$

over all possible permutations π.

Coarsening. We present a multiscale framework in which coarsening is interpreted as a modified process of weighted aggregation reinforced by the algebraic distance couplings for GMLogA. Algebraic distance-based coupling is a measure of connectivity strength between two nodes connected by an edge [9, 4]. Given the weighted Laplacian of a graph, denoted by $L = D - W$, we define an iteration matrix H for Jacobi overrelaxation as $H = (1-\omega)I + \omega D^{-1}W$, where $0 \leq \omega \leq 1$. The algebraic distance coupling ρ_{ij} is defined as

$$\rho_{ij} = 1/\big(\sum_{r=1}^{R} \lg |\chi_i^{(k,r)} - \chi_j^{(k,r)}|\big)\ ,$$

where $\chi^{(k,r)} = H^k \chi^{(0,r)}$ is a relaxed randomly initialized test vector, R is the number of initial test vectors, and k is the number of iterations. Considering

ρ_{ij} as an edge strength measure, one can construct a coarse graph by defining a classical AMG interpolation operator $\uparrow_c^f$ that will project the fine graph to the coarse graph. The projection is represented as $L_c \leftarrow \uparrow_c^f L_f(\uparrow_c^f)^T$, in which the centers of coarse aggregates are selected by traversing all nodes and leaving those nodes i in F that satisfy

$$\sum_{j\in C} \rho_{ij} / \sum_{j\in V_f} \rho_{ij} \geq \Theta_1 \quad \text{and} \quad \sum_{j\in C} w_{ij} / \sum_{j\in V_f} w_{ij} \geq \Theta_2 \ . \tag{26}$$

After identifying coarse nodes, we define for each fine i its coarse neighborhood N_i^c that contains a limited set of coarse nodes to which i is connected. The criterion for choosing C-nodes to N_i^c is also based on ρ_{ij}.

Uncoarsening. The uncoarsening contains the following four components: initial interpolation, Gauss-Seidel and compatible relaxations and refinement. Minimization of the contribution of a single node is a central question of interpolation and relaxations. Given an initial ordering, denote by N_i the set of ith neighbors with already assigned coordinates $\tilde{x}_j$. To minimize the local contribution of i to the total energy (25), we assign to it a coordinate x_i that minimizes

$$\sum_{j\in N_i} w_{ij} \lg |x_i - \tilde{x}_j| \ . \tag{27}$$

Since for every $j \in N_i$, $x_i = \tilde{x}_j$ implies that the sum (27) is minus infinity, we resolve this ambiguity by setting

$$x_i = \tilde{x}_t \iff t = \arg\min_{k\in N_i} \sum_{k\neq j\in N_i} w_{kj} \lg |\tilde{x}_k - \tilde{x}_j| \ . \tag{28}$$

The trivial exact solution has a running time $O(|N_i|^2)$, as it requires computing $|N_i|$ sums, each one with $|N_i|-1$ terms. Thus, to preserve the linear complexity of the entire algorithm, one can use the trivial solution for nodes with small $|N_i|$ only. We approximate (28) using a heuristic that seeks the nearly minimum sum in the point of maximal density. Consider set $\{\tilde{x}_j : j \in N_i\}$ as independent and identically distributed samples of a random variable with unknown distribution and w_{ij} as a posteriori probability of a sample $\tilde{x}_j$. Assuming that, we have to choose a point where the estimated probability density is maximized. We do so using the kernel density estimation, where the density at point x is estimated using the Gaussian kernel

$$\hat{d}(x) = \frac{1}{|N_i|h} \sum_{j\in N_i} w_{ij} 2^{-|x-\tilde{x}_j|/h} \ .$$

We will present a linear time algorithm for the density estimation.

Computational results. We compare our numerical results with several recent state-of-the-art methods: random-based, lexicographical, Gray, (double) shingle, spectral, and LayeredLPA orderings [3, 5]. Our benchmark consists of 100 graphs of different nature and size (most of them are taken from [7] and [8]). Evaluation of our method (with and without further compression) shows that algebraic distance based AMG framework significantly outperforms the existing methods, allowing network compression by up to 5 times better without losing good scalability.

Bibliography

[1] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *DCC '01: Proceedings of the Data Compression Conference*, page 203, Washington, DC, USA, 2001. IEEE Computer Society.

[2] A. Apostolico and G. Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[3] P. Boldi and S. Vigna. The Webgraph framework I: compression techniques. In *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, pages 595–602, New York, NY, USA, 2004. ACM.

[4] J. Chen and I. Safro. Algebraic distance on graphs. Technical Report ANL/MCS-P1696-1009 (Preprint), Argonne National Laboratory, 2009.

[5] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD '09*, pages 219–228, New York, NY, USA, 2009. ACM.

[6] Y. Choi and W. Szpankowski. Compression of graphical structures. In *Proceedings of the 2009 IEEE Symposium on Information Theory*, pages 364–368. IEEE Press, 2009.

[7] T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 97(23), 1997.

[8] J. Leskovec. Stanford Network Analysis Package (SNAP).

[9] D. Ron, I. Safro, and A. Brandt. Relaxation-based coarsening and multiscale graph organization. Technical Report ANL/MCS-P1741-0410 (Preprint), Argonne National Laboratory, 2010.

[10] J. Sun, E. Bollt, and D. Ben-Avraham. Graph compression-save information by exploiting redundancy. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(06):P06001, 2008.

36 A collection of web-based educational modules for interactively exploring algorithms in combinatorial scientific computing

H. Martin Bücker, Simon R. Leßenich, and Michael Lülfesmann
(*RWTH Aachen University, Germany)*

Corresponding Author: **H. Martin Bücker** (buecker@sc.rwth-aachen.de)

Despite the increasing importance of discrete and geometric algorithms in computational science and engineering, combinatorial scientific computing (CSC) is currently not among the standard courses taught at university level. Based on experiences with teaching a CSC course within the curriculum of computer science at RWTH Aachen University, we began the design and implementation of educational modules aimed at better illustrating elements of CSC. In this talk, we introduce a collection of web-based educational modules for teaching in the area of CSC. We refer to this collection as EXPLAIN (EXPLore Algorithms INteractively). The general philosophy behind EXPLAIN is not to replace the rich dynamic of verbal communication in a face-to-face learning environment by a comprehensive e-learning environment. Rather, the intention is to add—to the traditional classroom paradigm—interactive resources for exploring graph and the corresponding matrix transformations.

The common layout of the educational modules reflects this intention by showing a graph and its corresponding matrix representation next to each other. A typical layout of an EXPLAIN module is depicted in Figure 1. This module illustrates the symbolic Cholesky factorization of a sparse matrix. More precisely, the students use this module to interactively explore the phenomenon of fill-in, i.e., nonzero elements generated at positions, where the matrix to which the Cholesky factorization is applied, contains zero elements. Symbolic matrix factorizations of sparse matrices are commonly modeled by sequences of graphs obtained from eliminating vertices one after another obeying certain rules that also effect the edges [1, 2, 3, 7, 8]. EXPLAIN visualizes and allows to explore these sequences in terms of both elimination graphs and corresponding matrix representation.

In Figure 1, for instance, the elimination graph of a sparse 6×6 matrix is shown after the vertices 3 and 1 have already been eliminated during the previous steps of the Cholesky factorization. The order of the vertex elimination is depicted in that figure by the indices of the rows/columns. Dynamic user interaction is possible by selecting a vertex for the next factorization step using the computer mouse or by going backward and forward in the elimination ordering. The educational module also offers the opportunity to open an additional window (not shown here) to visualize, for a single factorization step, the elimination of edges one after another. Fill-in edges and corresponding matrix entries are highlighted by red color. EXPLAIN also offers support for uploading matrices so that students can experiment with their own matrices.

The EXPLAIN collection is written in Python and is accessible via a web interface to minimize the effort needed for the instructor to make the collec-

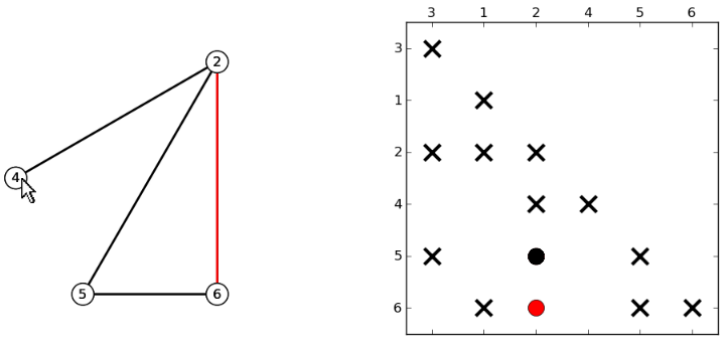


Figure 1: A typical layout of a module showing an instance of a graph $G(A)$ along with the nonzero pattern of a corresponding matrix A. Transformations carried out by interactively clicking on the graph are simultaneously carried out and visualized on the matrix.

tion available to the students. Several different software packages are combined, among others *NetworkX* [4] for providing graph data structures and *matplotlib* [5] for visualizing graphs and matrices. A predecessor of EXPLAIN is the standalone implementation of a Cholesky factorization module presented in [6]. In contrast to EXPLAIN, this educational module is not available using a web browser. Furthermore, EXPLAIN is not restricted to the symbolic Cholesky factorization and is designed to allow seamless integration of various other CSC algorithms, e.g., graph coloring and vertex elimination algorithms occurring in automatic differentiation.

Bibliography

[1] A. Davis. *Direct Methods for Sparse Linear Systems*. Number 2 in Fundamentals of Algorithms. SIAM, Philadelphia, PA, USA, 2006.

[2] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

[3] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA, USA, August 2008.

[5] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[6] M. Lülfesmann, S. R. Leßenich, and H. M. Bücker. Interactively exploring elimination orderings in symbolic sparse Cholesky factorization. In *International Conference on Computational Science, ICCS 2010*, volume 1(1) of *Procedia Computer Science*, pages 867–874. Elsevier, 2010.

[7] S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3(2):119–130, 1961.

[8] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

37 Simulation of particle-filled hollow spheres

Tobias Steinle, Jadran Vrabec, and Andrea Walther
(*University of Paderborn, Germany)*

Corresponding Author: **Tobias Steinle**
(tobias.steinle@uni-paderborn.de)

Motivation and Industrial Applications

Limited fossil fuel supply and increasing awareness of climate change has led to a raise in demand of highly efficient cars and industrial machines. These can be constructed to be more efficient, for example, by reducing their mass. We deal with the concept of a new kind of lightweight material that uses particle filled hollow spheres which are embedded within technical structures. When the structure is vibrating, for example, caused by an engine, the particles, e.g. a ceramic powder, can help to suppress these vibrations quickly by converting kinetic energy to heat through friction arising from collisions of the particles with each other and with the hull of the sphere. This new material could be deployed in machine casings to reduce noise, wear and tear that is caused by vibration.

Computational methods

Modern molecular dynamics methods are capable to simulate large numbers of particles. However, the modelling used there is not realistic for the behavior we want to simulate. For instance, in molecular dynamics usually periodic boundary conditions are considered for the simulation volume. In our case, the particles must interact with the boundary of the simulation box to allow for energy transfer by vibration and particle impact. Additionally, most implementations only allow for a limited number of shapes for the simulation volume. Therefore, several adjustments need to be made. We based our algorithm on a molecular dynamics code developed at the Department of Engineering, University of Paderborn. Newton's equations of motion of the particles are numerically solved by the leapfrog method that computes particle positions, velocities and forces between the particles alternatingly (therefore "leaping").

In the first part of the leapfrog scheme, at time t_i, the velocities are updated based on the forces of the previous step. These velocities are then used to calculate the new positions of the particles. Next, at time $t_{i+\frac{1}{2}}$, the moments and the (impact-) forces between the new sites of the particles are evaluated. The forces are based on the Lennard-Jones (LJ) potential, a pair potential, which is also widely used in molecular dynamics. This is then repeated at time t_{i+1}.
The Lennard-Jones potential U_{ij} is the source of the force F_{ij} acting between

two particles i and j, whose distance is r_{ij}. It is given as

$$U_{ij} = 4\varepsilon \left\{ \left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right\}$$

and the force acting on the particles i and j is

$$F_{ij} = -\frac{\partial U_{ij}}{\partial r_{ij}}.$$

Here ε signifies the depth of the potential energy well and σ is the size parameter which defines the range of the repulsive force. For our application, the LJ-potential needs to be cut off at a certain distance as otherwise the particles would keep interacting even when they are far away from each other. This is like a gravitational effect that needs to be taken into account in molecular dynamics but is not realistic when simulating granular particles. When particles collide with each other, the LJ-potential causes a repulsive force between the particles that will have an effect on the velocities of those particles in the next step of the leapfrog algorithm. Besides just evaluating the translational motion, rotational motion is also tracked. A quaternion representation is used for the rotation of the particles and Fincham's explicit quaternion algorithm is applied to solve the rotational equations of motion. The simulation is executed inside a spherical volume. The sphere itself can be excited to vibrate, giving an additional effect. The hull carries its own Lennard-Jones potential and acts as a reflective boundary condition. When a particle collides with the wall, it is reflected.
In this talk, we present first results with respect to a moderate number of particles inside the hollow sphere.

38 A study of the influence of sparse matrices reordering algorithms for ILU(p) preconditioner on the GMRES method

Kamila Ghidetti, Lucia Catabriga, Maria Claudia Boeres, and Maria Cristina Rangel
(*Universidade Federal do Espírito Santo, Brazil)*

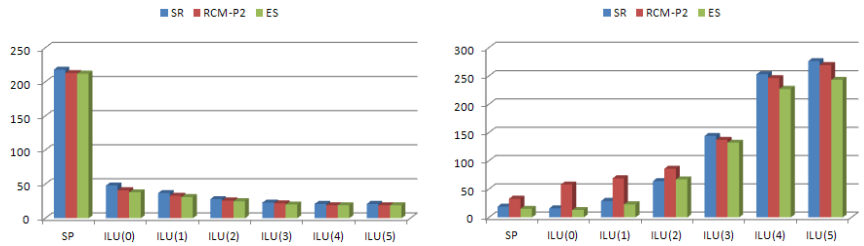
Corresponding Author: **Lucia Catabriga** (luciac@inf.ufes.br)

A significant portion of scientific problems include the solution of sparse and large linear systems. In these cases, the minimizing of the bandwidth and reducing of the envelope are ways to simplify the solution of these systems. These pre-processing methods consist of performing permutation between rows and columns. In the context of the solution systems via direct methods, minimizing the bandwidth reduces the filling that occurs in LU decomposition. Large linear systems, however, are usually resolved by non-stationary iterative methods. These methods do not change the sparsity of the matrix but require convergence criteria.Generally, a process to accelerate convergence, called preconditioning, is necessary. The preconditioners, based on incomplete LU decomposition, are widely used to sharply accelerate the convergence rate. Such operations alter the sparseness of the matrix and consequently the effectiveness of the preconditioners depends on matrix reordering.

This work analyzes the influence of matrices reordering algorithms on solving linear systems using preconditioned Krylov-type iterative methods considering the incomplete LU factorization preconditioners. The algorithms referenced most often in the literature for the reordering of matrices are Reverse Cuthill-McKee (RCM) ([1, 2]), Gibbs-Poole-Stockmeyer (GPS) ([3]) and Spectral (ES) ([4]). We analyze some of these algorithms and propose modifications comparing their solution qualities and performance in term of processing time when they are using as a pre-processing in the ILU(p) preconditioner for the GMRES method.

Those algorithms relate the problem of reordering matrices with the problem of relabeling in graphs. The RCM and GPS apply a breadth-first search procedure on the graph associated to the sparse matrix and return a new labeling order of its vertexes, which corresponds to the permutation of rows and columns of the matrix. The algorithm ES is based on the use of eigenvalues and eigenvectors of the Laplacian matrix of the corresponding graph, sorting a specific eigenvector associated to the graph vertexes and applying this sort as the permutation of rows and columns of the sparse matrix.

Recently, we proposed some modifications to those algorithms, in order to improve the quality of the solution and reduce the processing time, [5]. For the RCM algorithm we proposed to use the level structure of the vertex obtained by the heuristic proposed by [2], performing the new labeling of the vertexes from their positions in levels and increasing order of degrees. This algorithm, called RCM-P2, needs a smaller processing time than the RCM, with a similar



(a) Number of iterations (b) Processing time (seg)

Figure 1: Number of iterations and processing time for the GMRES method.

quality solution. The reduction in processing time may be explained by no long needing to execute the breadth-first search of the RCM algorithm.

The calculation of the eigenvalue and eigenvector associated required in the ES algorithm is highly complex. For this, we chosen in our implementation the Chaco library due to optimized storage matrix and low processing time. We utilized the Multilevel Symmlq/RQI algorithm, that is suitable for large-sized matrices. This algorithm considers the RQI (Rayleigh Quotient Iteration) iterative method, which solves a linear system using the LQ decomposition in each iteration. It is worth noting that the quality of solution obtained depends strongly on the tolerance considered for the iterative process. We also point out that we have changed the source code for Chaco. To reduce the processing time we calculate only the eigenvector associated with the second smallest eigenvalue, that is needed on the ES algorithm.

The matrices used in the computational tests are in the Matrix Market format available on the websites `http://math.nist.gov/MatrixMarket/` and `http://www.cise.ufl.edu/research/sparse/matrices/`. These matrices originate from a variety of application field such as computational fluid dynamics (CFD), electromagnetism, chemistry, and so forth. In this abstract we are to comment only the behavior of a finite element resulting matrix (FEM-3D-t2), that is come from of the CFD application area, but a complete study can be found in [6].

Figure 1 shows the number of iterations and processing time for GMRES without preconditioning and with ILU(p), p= $0, 1, \ldots, 5$ preconditioner for the FEM-3D-t2 matrix. This matrix has 147900 rows, 3489300 non-zero elements, that represents a sparsity percentage of $99,98\%$. In Fig. 1, SP means without preconditioning and SR means without reordering. The number of GMRES iterations decreases drastically when an ILU(p) preconditioner is used and continue decreasing when p increases, as we can see in Fig. 1(a). The use of the reordering strategy reduces discretely the number of GMRES iterations. For FEM-3D-t2 matrix the ES reordering algorithm leads a smallest number of iterations for all considerations (SP and ILU(p), p=$0, 1, \ldots, 5$).

However, the processing time has a different configuration, as we can see in Fig. 1(b). In general, when p increases the processing time also increases. But, using ES reordering and ILU(0) preconditioner we need a smallest processing time for FEM-3D-t2 matrix. For this matrix, the RCM-P2 does not reduce the processing time without preconditioning (SP) and for ILU(p), p= $0, 1, 2$ when compared with SR and ES.

Using a set of sparse matrices do not show here for compactness, we conclude that the reordering of matrices, in most cases, reduces the number of iterations in the GMRES method, but that reducing the processing time depends on the size and conditioning of the matrix.

Bibliography

[1] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices* Proceedings of the 24th ACM national conference, 1969, 157–172, ACM Press, New York, NY, USA, http://portal.acm.org/citation.cfm?id=805928

[2] A. George and W. Liu *ACM Transactions on Mathematical Software* 236–250, An implementation of a pseudoperipheral node finder, 5, 1979

[3] N. E. Gibbs and W. G. Poole and P. K. Stockemeyer, *An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix*, SIAM Journal of Numerical Analysis, 13, 2, 236–250, 1976

[4] S.T. Barnad and A. Photen and H.D. Simon, *Numerical Linear Algebra with Aplications*, 317–334, A spectral algorithm for envelope reduction of sparse matrices, 3, 1995

[5] Kamila Ghidetti and Maria Claudia Boeres and Lucia Catabriga, *A comparative study of the sparse matrices heuristics reordering algorithms (in portuguese)* , Proceedings of the 43th Brazilian Symposium of the Operation Research, 2010

[6] Kamila Ghidetti and Lucia Catabriga and Maria Claudia Boeres and Maria Cristina Rangel, *A study of the influence of sparse matrices reordering algorithms on Krylov-type preconditioned iterative methods*, Proceedings of the 31th Iberian Latin American Congress on Computational Methods in Engineering-CILAMCE, 2010,

39 A Simulator for Large-scale Parallel Computer Architectures

Curtis L. Janssen, Helgi Adalsteinsson, Scott Cranford, Joseph P. Kenny, Ali Pinar, David A. Evensky, and Jackson Mayo
(*Sandia National Labs, United States)*

Corresponding Author: **Ali Pinar** (apinar@sandia.gov)

The degree of parallelism that must be exposed to efficiently utilize modern large-scale parallel computing systems is intimidating. Because individual processor performance gains are currently achieved primarily through multiple cores on a chip and multiple threads of execution in a core, the rate at which parallelism must be exposed by an application will increase as a function of overall machine performance relative to historical trends. This results in greater design complexity for both machine architects and application software developers. The use of simulation, however, can aid both in their efforts to obtain high utilization from future computing platforms.

Simulation is already used extensively in the design of computing systems for both functional verification and timing estimation. As an example of the range of capabilities available, including just a few examples of open-source timing simulators, there are processor simulators (Binkert et al., 2006; M5Sim), memory simulators (Jacob; Wang et al., 2005), and network ns-3 (ns-3).

Several simulators have been developed to generate performance estimates for high-performance computing architectures. These range from high-fidelity and computationally expensive simulators for measuring performance between two nodes (Rodrigues et al., 2003; Underwood, Levenhagen, & Rodrigues, 2007) to lower-fidelity and lower-cost simulators that can estimate performance on large-scale machines. These lower-fidelity simulators use a variety of approaches to generate the applications processor and network workload including tracing, direct execution, and the use of skeleton applications. Additionally, the flow of data through the network is modeled with varying fidelity. In the present paper we are concerned with lower-fidelity and lower-costsimulation techniques to enable simulation at very large scales.

In the present work we describe a macroscale simulator for estimating the performance of large-scale parallel machines. The goals of the simulator are to assist in system design and application development. The simulator is modular, permitting multiple computation and communication models to be employed. This will allow the study of architectures at a variety of fidelities so we can trade off the computational cost of doing a simulation against the accuracy of the result. The simulator will be distributed under an open-source license to maximize its usefulness to the high-performance computing community. We focus on an extremely lightweight implementation, rather than enabling parallelism in the simulator itself. Parallelism can be easily introduced when performing independent simulations of architecture variants. We also provide a detailed MPI model that converts the high-level MPI events into the necessary communication operations. Because the MPI capability is implemented to be modular, it is

simple to investigate the relative performance of various MPI algorithms. The simulator is designed to allow the use of alternative programming models, as well.

We believe that our simulator can be a critical enabling technology for the combinatorial scientific computing (CSC) community. Load balancing among processors and minimizing the effect of communication has been at the heart of many research activities in the CSC community. While the literature produced is very rich, the results are commonly presented in symbolic metrics, such load imbalance ratios, total communication volumes, or total communication pairs, etc. While these results are important to show the effectiveness of proposed solution methods, they can fail short while trying to convince potential users of these results to invest the time to adopt these novel techniques. For that reason, nothing will be as effective as actual timings on a real machine, but timings on a real machine is hard, since i) they cannot be available to everyone ii) it is hard to isolate the effects of the proposed techniques from everything else in the system iii) and basic research typically targets machines of the future. Our simulator can avoid all these problems. First, it is publicly available and can be run on almost any platform. Secondly, the simulation not only provides a nice abstraction of a real machine, but also can provide more than merely a runtime, as information about the performance bottlenecks is available within the simulator. Finally, the driving force for this simulator is to be able to simulate and design future machines.

The simulation can be driven either by traces collected from a real application, or by skeleton application that mimics a real application without any of the details of the application. In this talk, we will explain the machinery behind our simulator, and explain how it can used, and how one can generate a skeleton application to observe the effects of a proposed technique.

Our work is done in the context of a larger project to develop a parallel multiscale simulator that permits users of the simulator to select the desired level of fidelity for each component of the machine. This larger project is an outgrowth of the Structural Simulation Toolkit (SST) (Rodrigues et al., 2003; Underwood et al., 2007) and the macroscale components described herein will be referred to as SST/macro to distinguish them from the existing microscale SST components.

40 A Scalable Parallel Framework for Graph Similarity

Madan Sathe and Olaf Schenk
(*University of Basel, Switzerland)*

Giorgos Kollias and Ananth Grama
(*Purdue University, USA)*

Corresponding Author: **Madan Sathe** (`madan.sathe@unibas.ch`)

With widespread availability of graph-structured data from sources ranging from social networks to biochemical processes, there is increasing impetus for efficient and scalable graph analyses techniques. An important problem in analyses of graph databases is the computation of node-wise similarity across graphs (or within the same graph). These node similarity scores can be used to quantify aggregate similarity, or as seeds for identifying similar (conserved) sub-graphs.

The similarity of two nodes can be recursively quantified in terms of the similarity of their neighbors. Best matching pairs of nodes in the two graphs can be identified through a maximum weighted bipartite matching algorithm.

To reduce the computational cost of these two expensive steps, we present similarity computation and matching algorithms, along with their highly scalable parallel formulations. Our similarity computations uncouple and decompose the similarity matrix to significantly reduce operation count, as well as enhance inherent concurrency. Our matching algorithm is based on a parallel implementation of the auction algorithm, which provides excellent performance and scaling, in conjunction with our similarity computations.

Numerical experiments on the Cray XE6 demonstrate that:

1. our similarity computation algorithm is at least an order of magnitude faster than a state-of-the-art implementation in terms of serial performance,

2. scales almost linearly on up to $3,072$ compute cores, and

3. the integration of this similarity computation with auction-based matching enables the similarity analysis of networks of sizes at least three orders of magnitude larger than currently possible (millions of nodes, tens of millions of edges).

Bibliography

[1] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. *Technical report*, Stanford University, 1998 2008.

[2] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren. A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching. *SIAM Rev.*, 46(4):647–666, 2004.

[3] R. Singh, J. Xu, B. Berger. Pairwise Global Alignment of Protein Interaction Networks by Matching Neighborhood Topology. *RECOMB 2007*, 16–31, 2007.

41 Parallelization of ADOL-C for MPI-parallel function evaluations

Benjamin Letschert, Kshitij Kulshreshtha, and Andrea Walther
(*Universität Paderborn, Germany)*

Corresponding Author: **Benjamin Letschert** (benny.letschert@web.de)

ADOL-C is an open-source package for the automatic differentiation of C and C++ programs. For this purpose ADOL-C uses operator overloading to generate an internal representation of the function to be differentiated, the so-called *tape*. Subsequently, the tape is used to compute for first and higher-order derivatives. Up to now, ADOL-C can be used to differentiate serial programs and programs that are parallelized using a small subset of OpenMP.

This project extends the ADOL-C package such that the differentiation of MPI-parallel simulation codes becomes possible. Since the user decides which calculation is done by which process an appropriate parallelization of the taping process is required, where each process generates a separate tape. Hence, the originally purely sequential taping is splitted into the generation multiple subtapes and the information of a subtape only is used by the corresponding process. Consequently, the coupling of the several processes also has to be incorporated in the taping.

Currently the MPI-parallel simulation to be differentiated with ADOL-C can include the MPI routines *Send, Receive, Barrier, Broadcast, Reduce*. For the implementation of these MPI routines each MPI-function has its own wrapper containing appropriate algorithms for sending/receiving data during the taping. Hence, each process transfers calculated data to other processes which are waiting for the data in order to continue the calculation.

For the subsequent derivative calculation, additional send- and receive operations were added to take different modes like *first order scalar, higher order vector , ...* into account. Furthermore, strategies to compute all function calls in the corresponding order were included for the handling of the differentiation in *forward* or *reverse mode*.

Putting everything together, the frequently used ADOL-C drivers gradient(..), hessian(..), and jacobian(..) now can handle MPI-parallel simulations. Furthermore, also sparsity can be taken into account since also the detection of sparsity structures was extended correspondingly. Therefore, even sparse Hessians and Jacobians can be computed in parallel. For this purpose, the coloring is still done in a serial fashion. An appropriate parallelization of this part will be the subject of future joint work with the developers of the *ColPack* package.

The poster will also contain an illustration of the runtime and communication behavior for an example originating from an optimal power flow problem.

42 Site-Based Partitioning and Repartitioning Techniques for Parallel PageRank Computation

Ali Cevahir
(*Tokyo Institute of Technology, Japan)*

Cevdet Aykanat and Ata Turk
(*Bilkent University, Turkey)*

B. Barla Cambazoglu
(*Yahoo! Research, Spain)*

Corresponding Author: **Ali Cevahir** (ali@matsulab.is.titech.ac.jp)

The PageRank algorithm is an important component in effective Web search. At the core of this algorithm are repeated sparse matrix-vector multiplications (SpMxV) where the involved Web matrices grow in parallel with the growth of the Web and are stored in a distributed manner due to space, network and time limitations. Hence, the PageRank computation, which is frequently repeated, must be performed in parallel with high efficiency and low preprocessing overhead while considering the initial distributed nature of the Web matrices.

Sparse matrix partitioning models are shown to be quite successful in load balancing and minimizing the total volume of communication during the repeated parallel SpMxV in PageRank computations [2]. However, the vast sizes of the Web matrices and the high preprocessing overhead incurred by these models render this solution infeasible in practice. In [5], we first focus on reducing the above-mentioned partitioning overhead. For this purpose, we propose two different Web matrix compression schemes, namely 1D and 2D compression, by exploiting the site information inherently available in page links. The 1D scheme compresses the $n \times n$ Web matrix along only one dimension, i.e., either along rows or columns, thus obtaining an $m \times n$ or $n \times m$ matrix, where n is the number of pages and m is the number of sites. 1D rowwise and 1D columnwise partitioning models are discussed under this 1D compression scheme. Only hypergraph partitioning (HP) models are considered for partitioning 1D-compressed Web matrices since the graph partitioning (GP) models are not suitable for partitioning rectangular matrices. The 2D scheme compresses the matrix in both dimensions, obtaining an $m \times m$ matrix. Since 2D-compressed Web matrices are square, both HP and GP models can be used for partitioning. However, even though the weights of the nonzeros of hypergraph models for these 2D-compressed Web matrices correctly summarize the computational requirements of both row-parallel and column-parallel SpMxVs, the sparsity patterns of them do not correctly summarize the communication requirements. Hence, as the superiority of the HP models depends on the correct modeling of the communication volume, it is not meaningful to use HP and we utilize GP models for these matrices. 1D rowwise and 1D columnwise partitioning models are formulated and discussed under this 2D compression scheme. These partitioning

models significantly decrease the preprocessing overhead of partitioning the $n \times n$ matrix, without sacrificing the parallel efficiency.

Partitioning models discussed in the literature generally assume availability of a global Web graph, possibly stored as a single file or dataset in a host machine. However, in a real-world scenario, this assumption may not be valid since the initial Web dataset is likely to be distributed among many processors. In such a setup, the data has to be redistributed among processors for efficient parallel PageRank computations. Hence, partitioning models should encapsulate the initial data redistribution overhead as well as the communication overhead that will be incurred during the parallel PageRank computations. This problem constitutes a typical instance of the repartitioning (remapping) problem. In [5], we adopt the recently proposed repartitioning models [1, 3, 4], which are based on HP and GP with fixed vertices, and apply them on top of our above-mentioned site-based models in order to encapsulate the initial redistribution overhead in parallel PageRank computations.

Moreover, in [5], we propose a simple yet effective method to handle pages with no in-links. This method avoids the SpMxVs associated with the submatrices corresponding to the pages with no in-links throughout the iterations by only performing two SpMxVs at the beginning. All of our contributions are presented in the context of a state-of-the-art sequential PageRank algorithm proposed by Ipsen and Selee [7], whereas our contributions can be easily extended to other iterative PageRank algorithms. This power-method-based algorithm [7] utilizes the lumping method to handle the dangling pages efficiently via applying the power method only to the smaller lumped matrix, where the convergence rate remains the same as that of the power method applied to the full matrix. It also has the advantage of allowing the dangling node vectors and personalization vectors to be different, thus enabling the implementation of TrustRank [6]. This algorithm is parallelized and tested on two PC clusters with 40 and 64 processors in order to verify the validity of the proposed techniques. PageRank computations conducted on eight well-known, large Web datasets (the largest of which has 133 million pages and 5.5 billion links) indicate the effectiveness of the proposed techniques. These techniques result in considerably high speedups while incurring a preprocessing overhead of several iterations (for some instances even less than a single iteration) of the underlying sequential PageRank algorithm.

Bibliography

[1] C. Aykanat, B.B. Cambazoglu, and B. Ucar, "Multilevel Hypergraph Partitioning with Multiple Constraints and Fixed Vertices", *J. Parallel and Distributed Computing*, vol. 68, pp. 609–625, 2008.

[2] J.T. Bradley, D.V. De Jager, W.J. Knottenbelt, and A. Trifunovic, "Hypergraph Partitioning for Faster Parallel PageRank Computation", *Lecture Notes in Computer Science*, vol. 3670, pp. 155–171, 2005.

[3] B.B. Cambazoglu and C. Aykanat, "Hypergraph-Partitioning-Based Remapping Models for Image-Space-Parallel Direct Volume Rendering of Unstructured Grids," *IEEE Trans. Parallel and Distributed Systems*, vol. 18, no. 1, pp. 114, 2007.

[4] U. V. Çatalyürek, E.G. Boman, K.D. Devine, D. Bozdag, R.T. Heaphy, and L.A. Riesen, "Dynamic Load Balancing for Adaptive Scientific Computations via Hypergraph Partitioning", *J. Parallel and Distributed Computing*, vol. 69, no: 8, pp. 711–724, 2009.

[5] A. Cevahir, C. Aykanat, A. Turk, and B. Barla Cambazoglu, "Site-Based Partitioning and Repartitioning Techniques for Parallel PageRank Computation," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no: 5, pp. 786–802, May 2011.

[6] Z. Gyongyi, H. Garcia-Molina and J. Pedersen, "Combating Web Spam with TrustRank", in *Proc. 30th VLDB Conf.*, vol. 1, pp. 257–263, 2004.

[7] I.C.F. Ipsen and T.M. Selee, "PageRank Computation, with Special Attention to Dangling Nodes", *SIAM J. Matrix Anal. Appl.*, vol. 29, pp. 1281–1296, 2007.

43 Algorithmic Differentiation and Nonlinear Optimization for an Inverse Medium Problem

Johannes Huber and Olaf Schenk
(*University of Basel, Switzerland)*

Uwe Naumann and Ebadollah Varnik
(*RWTH Aachen University)*

Andreas Wächter
(*IBM Research, USA)*

Corresponding Author: **Johannes Huber** (johannes.huber@unibas.ch)

Using the example of an inverse medium problem [1] we demonstrate how combinatorial techniques play a significant role for the efficient solution of large-scale optimization problems. In particular, the fast computation of sparse derivative matrices using algorithmic differentiation is made possible by coloring algorithms, and the efficient solution of sparse linear systems requires weighted graph matching algorithms.

Bibliography

[1] J. Huber, U. Naumann, O. Schenk, E. Varnik, and A. Wächter. Algorithmic Differentiation and Nonlinear Optimization for an Inverse Medium Problem. *Book Chapter in Combinatorial Scientific Computing*, Edited by U. Naumann and O. Schenk, Chapman-Hall CRC Computational Science.

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2008-01 *	Fachgruppe Informatik: Jahresbericht 2007
2008-02	Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04	Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05	Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06	Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07	Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08	George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09	George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10	George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11	George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13	Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14	Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15	Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves
2008-16	Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17	Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18	Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems

2008-19 Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems

2009-02 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications

2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices

2009-04 Daniel Klünder: Entwurf eingebetteter Software mit abstrakten Zustandsmaschinen und Business Object Notation

2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs

2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time

2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme

2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme

2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata

2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms

2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting

2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs

2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut

2010-11 Martin Zimmermann: Parametric LTL Games

2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut

2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems

2010-14 Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination

2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode

2010-16 Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles

2010-17 Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten

2010-18 Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit

2010-19 Felix Reidl: Experimental Evaluation of an Independent Set Algorithm

2010-20 Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games

2011-02 Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting

2011-03 Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems

2011-04 Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars

2011-07 Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV

2011-08 Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog

2011-11 Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains

2011-12 Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems

2011-13 Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP

2011-14 Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games

2011-16 Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM

2011-18 Kamal Barakat: Introducing Timers to pi-Calculus

2011-19 Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.