Combinatorial Algorithms to Enable Computational Science and Engineering: Work from the CSCAPES Institute

Erik G. Boman[1], **Umit V. Catalyurek**[2],
Cedric Chevalier[5], **Karen D. Devine**[1],
Assefaw H. Gebremedhin[3], **Paul D. Hovland**[4],
Alex Pothen[3], **Sivasankaran Rajamanickam**[1],
Ilya Safro[4], **Michael M. Wolf**[6], **and Min Zhou**[7]

[1] Scalable Algorithms Department, Sandia National Laboratories
[2] Biomedical Informatics, and Electrical and Computer Engineering, Ohio State University
[3] Computer Science, Purdue University
[4] Mathematics and Computer Science Division, Argonne National Laboratory
[5] Commissariat à l'énergie atomique
[6] MIT Lincoln Laboratory
[7] Scientific Computation Research Center, Rensselaer Polytechnic Institute

1. Improved parallel mesh partitioning

Load-balancing, partitioning and matrix ordering are, perhaps, the most well-known examples of combinatorial problems in massively parallel computing. CSCAPES researchers have developed new techniques and software for these problems, including matrix ordering for non-symmetric systems [26] and for parallel hybrid solvers [6], and hypergraph repartitioning [9, 10]. Mesh partitioning is an area where hypergraph partitioning can have a significant benefit, impacting a very wide array of applications.

Meshes play an important role in many scientific simulations, especially the numerical solution of partial differential equations by finite element, finite volume and finite difference methods. In parallel computations, high-quality mesh partitioning – dividing a mesh among parallel processors – is crucial to ensure memory use and computational work are balanced and interprocessor communication cost is low. A typical partition distributes mesh elements equally among processors, enabling the matrix assembly or equation formation operations to be well balanced. Mesh nodes are assigned to the processors owning their adjacent elements, with "ghost nodes" (mesh node copies) replicating nodal data along processor subdomain boundaries. The simulation's degrees of freedom are typically associated with the mesh nodes, so the amount of duplication via ghost nodes affects both the linear solver time and total memory usage.

The most commonly used approach to mesh partitioning [42] is to model the mesh using the dual graph, where graph vertices represent mesh elements, and graph edges connect each element with its adjacent elements (Figure 1, middle). Graph partitioners (e.g., Chaco [29], ParMETIS [31], Scotch [37]) then partition this graph (and, thus, the mesh) so that each part has an equal number of vertices (elements) while attempting to minimize the number of edges (adjacencies) cut by processor boundaries. This model can be represented more accurately (with respect to total communication volume) by a hypergraph – an extension of a graph that allows

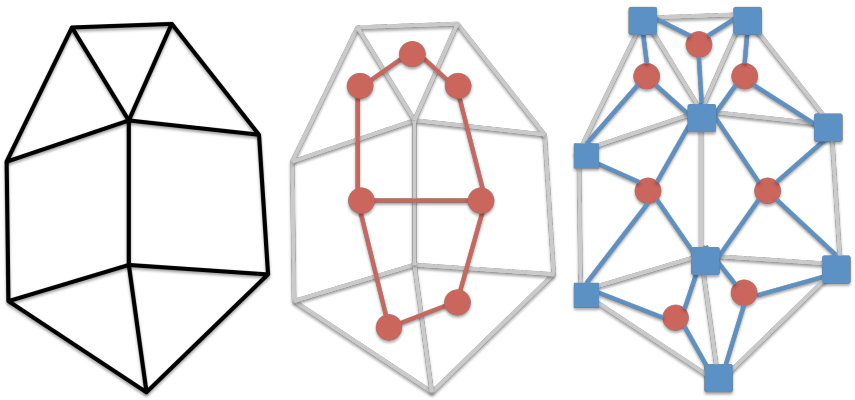

Figure 1. (Left) An example mesh. (Middle) The dual-graph model (shown in red) typically used in graph partitioning; graph vertices (circles) represent mesh elements, while graph edges connect adjacent elements. (Right) New hypergraph mesh-partitioning model; each hypergraph vertex (red circle) represents an element, while each hyperedge (blue square) represents a mesh node, connecting all elements incident to it.

edges to contain two or more vertices; each hypergraph vertex represents a mesh element, and each hyperedge connects a given mesh element and all elements sharing faces with that element. However, this model does not control the number of ghost nodes needed along subdomain boundaries.

CSCAPES researchers at Sandia National Laboratories developed a new hypergraph model for partitioning computational meshes in parallel simulations [14]. The new model accurately represents both interprocessor communication cost and the memory cost of ghosting mesh entities. As in the dual-graph model, the new model partitions mesh elements. But the new model uses hyperedges to additionally represent the dependence between mesh nodes and their adjacent elements. Thus, hypergraph vertices represent mesh elements, and hyperedges represent mesh nodes, with each hyperedge (mesh node) connecting the mesh elements incident to that node (Figure 1, right). A hypergraph partitioner then assigns an equal number of vertices (elements) to each part, while attempting to minimize the number of processors over which hyperedges extend (the number of processors sharing mesh nodes). Unlike the dual-graph model, this new model provides an exact measure of the communication volume associated with boundary data exchanges for shared mesh nodes along processor boundaries.

In collaboration with members of DOE's SciDAC Interoperable Technologies for Advanced Petascale Simulations (ITAPS) Center at Rensselaer Polytechnic Institute, we tested this new mesh-partitioning model on a 1.07 billion element mesh in an implicit finite element fluid-flow simulation of blood flow in abdominal aortic aneurysms (see Figure 2). We used the Zoltan toolkit [18], a widely used partitioning library that includes geometric, graph-based and hypergraph-based partitioners. Zoltan's parallel hypergraph partitioner [17] was used to generate partitions for both models up to 131K processors. The resulting application run times on the Intrepid BG/L are shown in Table 1. The time reductions were achieved by changing only the partitioning model; no changes to the application code were needed. Because the solver phase of the computation is node-based, it benefits most from the new model. Also shown in Table 1, our new mesh-partitioning model reduces the average number of mesh nodes per processor (including ghost nodes), reducing load imbalance in the solver, the total work done by the solver, and memory used by the solver. Moreover, in this experiment, the magnitude of these reductions grows as the number of processors is increased – an important feature in exascale

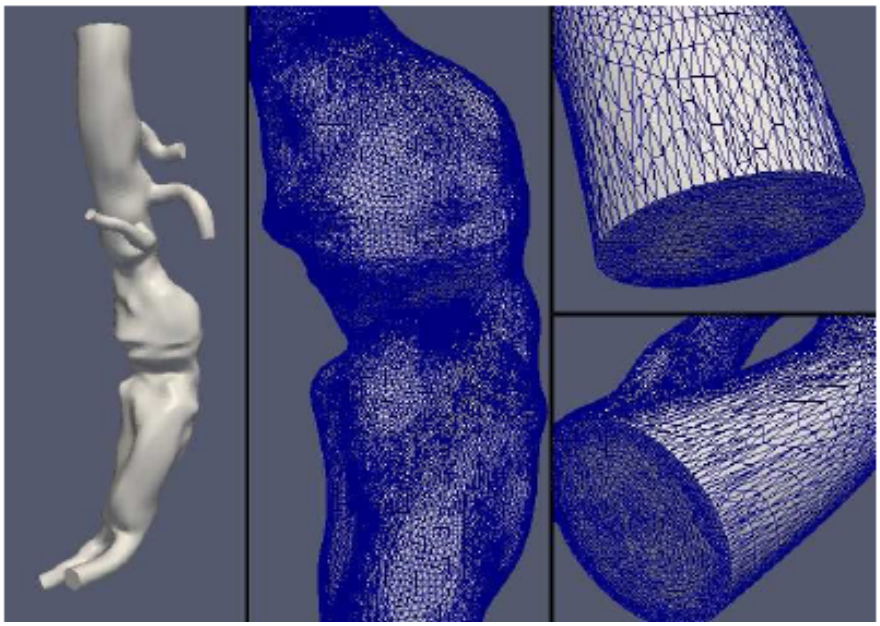

Figure 2. An example tetrahedral finite element mesh for modeling blood flow in abdominal aortic aneurysms. (Image courtesy of the Scientific Computation Research Center at Rensselaer Polytechnic Institute.)

	Dual-graph Model			New Mesh-Partitioning Model			Reduction due to New Model		
Number of Processors	Avg. nodes per proc.	Application solver time	Application total time	Avg. nodes per proc.	Application solver time	Application total time	Avg. nodes per proc.	Application solver time	Application total time
4,096	48,054	564.8 s.	966.2 s.	47,714	531.1 s.	935.3 s.	0.7%	6.0%	2.3%
32,768	12,822	77.3 s.	128.9 s.	12,495	73.8 s.	125.6 s.	3.7%	4.6%	2.6%
131,072	1,908	21.1 s.	34.1 s.	1,781	19.6 s.	32.6 s.	6.7%	6.7%	4.5%

Table 1. Application execution times and average number of mesh nodes per processor for the implicit finite element code PHASTA [30] using a 1.07B element mesh. Our new mesh-partitioning model decreases application execution time and the total number of mesh nodes compared to a dual-graph model.

where millions of subdomains will be used. These experiments were performed on tetrahedral meshes; we conjecture that the improvement may be greater for mixed-element meshes for which the accuracy of the partitioning model is more important. Based on these promising results, CSCAPES mesh-partitioning model has been adopted by ITAPS in its iZoltan mesh-partitioning service.

1.1. Graph Coloring

Graph coloring is an abstraction for partitioning a set of binary-related objects (modeled as a graph) into few groups of "independent" objects. Besides being an archetypal problem in discrete mathematics, coloring, in some variant, is a critical computational enabler in many contexts in computer science, scientific computing, high-performance computing etc. For example, it is used to maximize exploitable parallelism in preconditioned iterative methods for sparse linear systems, to quickly and approximately compute Schur complements, to determine task scheduling in concurrent computation, to compute sparse Jacobian and Hessian matrices efficiently, etc.

Matrix	1d partition	2d partition	Recovery
Jacobian	*distance-2 coloring*	*star bicoloring*	Direct
Hessian	*star coloring*	NA	Direct
Jacobian	NA	*acyclic bicoloring*	Substitution
Hessian	*acyclic coloring*	NA	Substitution

Table 2. Overview of graph coloring models in computation of derivative matrices. The Jacobian is represented by its bipartite graph, and the Hessian by its adjacency graph. NA stands for not applicable.

1.1.1. New Algorithms for Jacobian and Hessian Computation In CSCAPES, we have been developing algorithms and software for exploiting the inherent sparsity available in large-scale Jacobian and Hessian matrices, to make their computation via automatic differentiation (or finite differencing) efficient in terms of execution time, memory, and storage space requirements.

The framework we developed and employed for such a computation involves four steps: (i) automatic sparsity pattern detection, (ii) matrix compression via graph coloring, (iii) computation of the numerical values in the compressed matrix, and (iv) recovery of the numerical values in the derivative matrix from the compressed representation. Each step involves problems that come in several variations depending on whether the derivative matrix to be computed is a Jacobian (nonsymmetric matrix) or a Hessian (symmetric matrix) and on the specifics of the computational techniques used [20]. Table 2 provides an overview of the coloring problems associated with the second step.

We have developed (and deployed) new algorithms for problems occurring in each of the four steps of the framework outlined above, but we highlight only a few of them here.

We developed novel star and acyclic coloring algorithms, useful for Hessian computation [23]. The acyclic coloring algorithm is the first ever practical algorithm to be designed and implemented for the problem. The crucial idea in both algorithms is the exploitation of the structure of two-colored induced subgraphs, a collection of stars and trees, respectively. Taking the advantages these structures offer further, we developed efficient recovery algorithms for the computation of Hessians using direct as well as substitution-based methods [21]. We employed the new coloring and recovery algorithms within an AD tool to compute sparse Hessians to solve an optimization problem [21]. We showed, analytically as well as experimentally, that an acyclic coloring-based recovery via substitution is numerically stable, in contrast to previously known substitution methods where numerical stability was a major concern. We also provided new analytical results on star and acyclic coloring of chordal graphs and cographs, important classes of graphs with a wide range of applications in scientific computing [21, 33].

1.1.2. Ordering for Coloring and More Every one of the coloring problems listed Table 2 is NP-hard to solve optimally. The algorithms we have developed for solving them are fast, yet effective, greedy heuristics. They are greedy in the sense that vertices are colored sequentially one at a time and the color assigned to a vertex is never changed. The *order* in which vertices are processed in a greedy heuristic determines the number of colors used by the heuristic. We have developed and implemented various vertex ordering techniques that are effective in lowering the number of colors used by a greedy algorithm [24]. We characterized the ordering techniques solely in terms of relative degrees in a manner independent of a coloring algorithm that could use them. The decoupling helps achieve better software design, enables efficient implementation, and makes it easier to discover applications outside coloring in which the orderings are useful. All three of these advantages have been realized in our efforts.

1.1.3. ColPack Sofware We have assembled implementations of our coloring, ordering, matrix recovery and other supporting algorithms for Jacobian and Hessian computation in a software

package called `ColPack` [24]. The package is written in an objected-oriented fashion in C++ using the Standard Template Library. It is designed to be efficient, modular and extensible. The first version of `ColPack` was released for free public use under the GNU Lesser General Public License in October 2008. Several improved and updated versions had been released since then; the most recent release is dated March 2011. Download and other information is available at `http://www.cscapes.org/coloringpage`.

In collaboration with Andrea Walther at Paderborn University, Germany, `ColPack` has been interfaced with the operator overloading based automatic differentiation tool ADOL-C [25]. The latest version of ADOL-C has acquired new and more efficient sparsity pattern detection capabilities (based on propagation of index domains) for Jacobian as well as Hessian matrices, and we contributed to the development of these techniques [21, 22]. In a recent CSCAPES effort [34], `ColPack` has been interfaced with the newly developed source transformation based AD tool ADIC2 [35], a tool housed at Argonne. To make the interface with `ColPack` possible, new automatic sparsity detection capabilities were added to ADIC2 [34].

1.1.4. Impact `ColPack` coupled with ADOL-C or ADIC2 has enabled a number of applications and generated long-term collaborations. One example of an application enabled by a ColPack—AD toolkit is a Simulated Moving Bed (SMB) process, a purification technique widely used in the chemical, food, and pharmaceutical industry to separate liquid chemicals that are thermally unstable or have high boiling points [32]. We considered a case where the objective was to maximize throughput, which was modeled as an optimization problem with constraints given by partial differential algebraic equations. These were solved by discretizing in both space and time, which gave rise to Jacobians that are large and sparse. Using ColPack and ADOL-C, we showed that the computation of such Jacobians (which have fairly complex structures) could be reduced by several orders of magnitude relative to a computation that does not exploit sparsity [22]. Recently, the SMB application was enabled in a similar manner in the context of source-transofrmation based derivative computation via the ADIC2—ColPack integration [34].

Another example of application enabled by ColPack and ADOL-C involves the optimization of electric power flow in a network. The network consists of observable parts (where measured data is available) and unobservable parts (where data is estimated). The requirements here were formulated as an unconstrained optimization problem and solved using an interior point method, which relies on the provision of the Hessian of a Lagrange function. We showed that the use of star and acyclic coloring in the Hessian computation reduces overall runtime by several orders of magnitude [21]. This work on power optimization using ColPack and ADOL-C has been used by a power agency in France for some of its routine operations.

Our work on the SMB application inspired a collaboration with Prof. Larry Biegler (Carnegie Mellon University) with whom we have an ongoing conversation to extend the work on SMB to the related Pressure Swing Adsorption process, which is used in purification of gaseous mixtures (e.g., removing carbon dioxide from flue gases). The work on `ColPack` and its distribution has spurred a number of activities by various researchers internationally, including a PhD thesis at Aachen University in Germany. The package has also been used by practitioners in industry, including an energy firm in Portugal.

1.2. Distributed-memory Parallel Coloring Algorithms and Software

Development of parallel algorithms and software has been a central focus of our work on coloring in CSCAPES. We developed a framework for parallelizing greedy distance-1 coloring algorithms on distributed-memory computers [8]. The techniques employed in the framework include: careful exploitation of features of the initial data distribution; speculation—maximizing concurrency by tentatively tolerating inconsistencies and then detecting and resolving them; randomization; and infrequent, coarse-grain communication among processors as opposed to

frequent, fine-grained communication. In [8], several specialized algorithms designed using the framework have been presented, evaluated, and shown to be superior to previously known parallel coloring algorithms. Experiments carried out on clusters consisting of a few hundred processors demonstrated good scalability. In a recent work [11], the algorithms have been further optimized and were shown to be scalable up to tens of thousands of processors of the IBM Blue/P.

In a related work [7], we extended the framework developed in [8] to design parallel algorithms for coloring problems in sparse derivative computation. Specifically, we developed algorithms for distance-2 and restricted star coloring of general graphs (for Hessian computation) and partial distance-2 coloring of bipartite graphs (for Jacobian computation). No previous distributed-memory parallel algorithms were known for these problems. In a distributed-memory parallel distance-2 coloring algorithm, an efficient means for exchanging information between processors hosting a pair of vertices two edges away from each other needs to be devised. Direct communication between such a pair of processors would incur unduly high communication cost and duplicating distance-2 neighborhood information would require unduly large storage space and memory. Instead, we employed a strategy in which information is relayed via a third processor (the processor owning a mutual neighbor of vertices two edges away from each other) as needed. Experiments we run on various clusters showed that the resulting parallel distance-2 coloring algorithm scales well and gives a solution very close to the sequential variant, which is often close to optimal.

MPI implementations of the distance-2 coloring algorithm as well as the distance-1 coloring algorithm have been incorporated into and deployed via Zoltan. Zoltan interfaces for the partial distance-2 coloring and restricted coloring implementations have been created.

1.3. Coloring on Multicore and Multithreaded Architectures

Our latest work on coloring has focused on the emerging and rapidly growing multicore and massively multithreaded architectures. We have designed and implemented a set of efficient multithreaded algorithms for distance-1 coloring on a collection of platforms with varying degrees of multithreading capabilities [12]. The platforms we considered include a 128-processor Cray XMT (supporting more than 10,000-way parallelism via multithreading), a 16-core Sun Niagara 2, and an 8-core Intel Nehalem system. We found 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. In another recent work [36], we have developed multithreaded algorithms for a collection of vertex ordering problems and the distance-2 coloring problem.

2. Graph Matching

Background. Matching is a archetypical combinatorial problem for which we proposed to develop parallel algorithms. When we began our work, there was no practical parallel algorithm for matching, since the approach taken by most matching algorithms is inherently serial.

We have made progress in designing optimal and approximation algorithms for vertex-weighted matchings (VWM), a variant of the matching problem that has not been studied much. We have also developed a parallel 1/2-approximation algorithm for edge weighted matching (EWM), and implemented it on both shared memory and distributed memory multiprocessors. We have also begun to implement exact algorithms for computing maximum cardinality and maximum edge-weight matchings on many-core processors.

Optimal and Approximation Algorithms. We have discovered a property of optimal vertex-weighted matchings called the reachability property that leads to the design of optimal algorithms for VWM. We have also designed new 2/3-approximation algorithms for VWM on bipartite graphs [19, 27] that are faster than the optimal algorithms, while guaranteeing a weight for the approximate matching that has at least 2/3-rds of the weight of an optimal matching. The

approximation algorithm reveals further properties of the VWM problem. We have implemented the approximation algorithms for VWM and EWM, and show that the approximation algorithms can be faster by several orders of magnitude relative to the optimal algorithms. Computationally the weights of the computed matchings are higher than 95 percent of the weight of the optimal matchings on a large set of test problems.

Parallel Approximation Algorithms. Approximation algorithms for matching are much more amenable to parallelization since they search for short augmenting paths in the graph to increase the size of the matching. We describe results from a parallel 1/2-approximation algorithm for the EWM problem implemented on [11, 27]. The algorithm finds locally dominating edges, i.e., edges which are heavier than other edges incident on their endpoints, adds them to the matching, removes the matched edge and its neighboring edges, and iterates. We have implemented this algorithm on several thousands of processors of a Cray XT-4 at NERSC and an IBM Blue Gene/P at Argonne's Advanced Leadership Computing Facility. On these machines, we have computed matchings for graphs with hundreds of millions of edges and demonstrated strong scaling for graphs with good edge separators.

Matching on Multithreaded Processors. We have implemented the parallel half-approximation algorithm on multithreaded architectures such as the Intel Nehalem, the Sun Niagara, the massively multithreaded Cray XMT, and the Nvidia Fermi GPU [28]. A dataflow algorithm that exploited the low cost hardware-based synchronization gave the best results for the XMT. We obtained good strong and weak scaling for several classes of graphs on all these architectures.

Exact algorithms for maximum cardinality and maximum weighted matchings are amenable to parallelization on multithreaded processors. We are currently designing and implementing such algorithms. A maximum cardinality matching also enables the computation of the block (upper or lower) triangular decomposition (BTF) of sparse matrices, which has been used by Rob Hoekstra, David Day and colleagues at Sandia to solve circuit modeling problems in their Xyce tool for modeling circuits under extreme conditions. In one of the instances, BTF enabled the solution of a problem 200 times faster; it also enabled solution of this ill-conditioned problem (with condition number about 10^{30}) to be computed more accurately.

Our matching software is being interfaced with Trilinos, and is being used by our colleagues at Sandia, and the University of Bergen in Norway.

3. Multiscale graph algorithms

The Multiscale method is a class of algorithmic techniques for solving efficiently and effectively large-scale computational and optimization problems. This method was originally invented for solving elliptic partial differential equations and up to now it represents the most effective class of numerical algorithms for them. Whereas the variety of continuous systems' multiscale algorithms turned into a separate field of applied mathematics, for combinatorial optimization problems they still have not reached an advanced stage of development, consisting in practice of a very limited number of multiscale techniques.

The main objective of a multiscale algorithm is to create a hierarchy of problems (coarsening), each representing the original problem, but with fewer degrees of freedom. For the graph modeled problems, this hierarchy may be viewed as a process of learning of a graph topology prior to applying any approximation method. The construction of hierarchies at different scales ends up at the level with a very small number of degrees of freedom (coarsest level) that allows to get a first approximation to the original problem at very large scale within an insignificant running time (even for exact algorithm) in comparison to the size of original problem. Then, the obtained approximation is sequentially projected along all levels of the hierarchy (interpolation or projection) until it reaches the original problem with some approximation for it. The projection stage can be reinforced at each level by refinement algorithms that improve the

quality of approximation before further projection. The projection reinforced by a refinement method is called uncoarsening. Multiscale representation of a manifold on which a combinatorial optimization problem is formulated allows to apply different approximation or exact methods at all scales. Adapting these methods for working at local domains only can improve the overall complexity significantly while applying them at different scales implies the preservation of large-scale solution features inherited from coarser scales.

The multiscale framework has two key advantages that make it attractive for applying on modern large-scale instances: it can exhibit a linear complexity, and it can be relatively easily parallelized and implemented by using standard matrix-vector operations. Another advantage of the multiscale framework is its heterogeneity, expressed in the ability to incorporate external appropriate optimization algorithms (as a refinement) in the framework at different scales. Below we present examples of problems and multiscale techniques designed by CSCAPES members.

3.1. Relaxation-based coarsening of graphs

At each level of coarsening one needs to define the set of coarse variables and the equations (or relations) that they should satisfy. Each coarse variable is defined in terms of the next-finer-level variables and each coarse level equation is based on the set of next-finer-level equations. Determining the set of coarse variables and equations is the central task of any multiscale scheme as the quality of approximation (and, thus, of the solution) strongly depends on them. In the process of defining the set of coarse variables and in constructing an explicit interpolation, it is important to know how close two given fine-level variables are to each other at the stage of switching to the coarse level. We need to know, in other words, to what extent the value after the refinement of one variable implies the value of the other. If they are sufficiently close, they can, for example, be aggregated to form a coarse variable. In [39] and [13] we introduced and analyzed a measure of closeness between two nodes in a given graph. More generally, we defined the distance of one variable x from a small subset S of several variables, in order to measure how well x can be interpolated from S. We demonstrated the effectiveness of the measure in multiscale frameworks (for (hyper)graph partitioning and linear ordering) and several well-known greedy algorithms on graphs where a "greedy decision" depends on the strength of connection between two nodes. The proposed measure can be easily calculated and parallelized.

3.2. Network compression-friendly ordering

Finding a suitable compressed representation of large-scale networks (or matrices) has been intensively studied in both practical and theoretical branches of data mining [16, 2, 15, 5, 43]. In particular, the success of applying some of the recently proposed compression schemes [15, 5, 3] 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 better performance of compression schemes and network element access operations. Our multiscale algorithm [41] for "compression-friendly" arrangement is a linear solver for the minimum logarithmic arrangement problem [15]. It is based on the algebraic multigrid methodology for graph linear ordering problems [40] and local kernel density estimation used as a relaxation. We demonstrate significant improvement for minimization of logarithmic arrangement for various families of networks, including social networks and other (ir)regular instances. The comparison was performed with several methods recently introduced in [1, 5, 4, 15]. In almost all cases, our solver exhibited better numerical results than previous best-known results (up to 80% of imptovement).

3.3. Optimal location of two-dimensional objects

The optimization problem we addressed in this work [38] is to find an optimal location of two-dimensional (2D) objects such that (a) the total length of the given connections between these

objects will be minimal, (b) the overlapping between objects will be as little as possible, and (c) the 2D space will be efficiently utilized. This class of problems can be modeled by a graph in which every vertex has a shape and area and each edge has a weight. In many theoretical and applied fields, this class of problems is often addressed and actually poses a computational bottleneck. Examples include problems such as graph visualization, facility location, wireless network coverage, and optimal VLSI placement. The corresponding optimization problem consists of quadratic utility function (other functionals can be used via quadratization) and linear inequality constraints. It is succesively approximated by the system of equations and solved by a combination of two multigrid techniques, namely, the correction scheme for the utility minimization and the full approximation scheme for the inequality constraints dened over the 2D squares. The entire algorithm solves the nonlinear minimization problem by applying successive steps of corrections, each using a linearized system of equations to define a "pseudo-Lagrangian".

4. Education and Outreach

4.1. Education and Training

Along with performing research and enabling applications in petascale simulations, CSCAPES had a goal of educating the next generation of researchers in combinatorial scientific computing. Toward that end, CSCAPES has sponsored four post-doctoral researchers. Cedric Chevalier (Sandia) performed research in sparse matrix ordering, graph coarsening, and mesh partitioning; he is now a staff scientist at Commissariat à l'énergie atomique (CEA) in France. Michael Wolf (Sandia) studied two-dimensional matrix partitioning and multicore solvers; he is now employed by MIT Lincoln Laboratories. Sivasankaran Rajamanickam is Sandia's current CSCAPES postdoc, studying sparse hybrid solvers. Ilya Safro is currently Argonne's CSCAPES postdoc, where he studies the application of algebraic coarsening to graphs. CSCAPES has also supported five summer interns at Sandia and Argonne, working on matrix partitioning, hypergraph coarsening graph matching, and automatic differentiation.

In addition to the four post-doctoral scientists mentioned above, the following graduate and undergraduate students were advised by CSCAPES personnel.

- The following three PhD students graduated during the period of the CSCAPES Institute. Michael Wolf (Illinois), postdoc at Sandia; Mahantesh Halappanavar (Old Dominion) Research Staff member at Center for Adaptive Supercomputing at PNNL; Doruk Bozdag (Ohio State), currently at Google.
  The following PhD students are pursuing their PhD degrees. At Purdue: Ariful Azad, Duc Nguyen (Ross Fellow), Arif Khan. Ahmet Sariyuce is at Ohio State, and Andrew Lyons (Argonne) at Vanderbilt.
- CSCAPES researchers also worked with the following summer students: Aydin Buluc (Sandia), currently the Luis Alvarez fellow at LBL. Nana Arizumi (Argonne); Heather Cole-Mullen (Argonne); Michael Wolf (Sandia); and Arif Khan (Sandia).
- Visiting international PhD students: Johannes Langguth, Mohammed Mostafa Ali Patwary (both from Bergen, Norway, visited Purdue). Patwary is now a postdoctoral scholar at Northwestern University.

In addition, Sandia sponsored a Harvey Mudd College Computer Science Clinic team. Clinic teams consist of four college seniors who work together for an entire school year on a research problem designated by the sponsoring institution. Sandia's team (see Figure 3) investigated the feasibility of two-dimensional matrix partitioning for a wide class of problems, implementing two-dimensional Cartesian and Recursive Bisection methods in the Trilinos library and evaluating the methods on parallel computers at NERSC. This clinic project provided the students' with their first exposure to parallel computing and large-scale software projects.

Figure 3. Sandia's Harvey Mudd College Computer Science clinic team (Michael Leece, Joe DeBlasio, Audrey Lawrence, and Katie Ewing) studied the feasibility of two-dimensional matrix partitioning; they are shown here with their poster on project presentation day – a college-wide event for seniors and sponsors – in May 2011.

4.2. Outreach: Conferences

The CSCAPES Institute participated in a number of outreach events from 2007- 2010. All of them are listed on the cscapes web page, so we will mention a few highlights here. We organized seminars among our institutions (roughly once a month for the initial three years and a few times a semester after that) using the Access Grid and other remote conferencing tools. The premier conference in the area is the SIAM Workshop on CSC, and these workshops were held in 2007, 2009 and 2011. CSCAPES researchers serve on the steering committee of the CSC community that organizes these workshops, and also served as Chairs of the organizing committee. CSCAPES funding was used to enable graduate students and post-doctoral scholars to attend these workshops. A CSCAPES software tools workshop was held in Santa Fe, NM with tutorials on the use of Zoltan, ADIC, OpenAD, Colpack, and talks from the developers of the tools and the user community. A Dagstuhl workhop on CSC was held in February 2009 in Dagstuhl, Germany, where CSCAPES researchers gave a number of plenary talks and again offered tutorials on the software tools developed by us. A book on CSC was published based largely on papers presented at this workshop, and CSCAPES researchers contributed to x chapters in this book. We also participated in the AD2008 conference, and also contributed x papers to the Proceedings of the conference. Finally, we organized four minisymposia with 16 talks on CSC at the International Conference on Industrial and Applied Mathematics (ICIAM) 2011 in Vancouver.

Acknowledgments

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U. S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. The work at Argonne was supported by the Mathematical,

Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy under Contract W-31-109-Eng-38. The CSCAPES Institute is supported by the U.S. Department of Energy's Office of Science through grant DE-FC-0206-ER-25774, as part of its SciDAC program.

References

[1] Laboratory for Web Algorithmics. http://law.dsi.unimi.it/.

[2] 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.

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

[4] P. Boldi, M. Santini, and S. Vigna. Permuting web graphs. In K. Avrachenkov, D. Donato, and N. Litvak, editors, *WAW*, volume 5427 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2009.

[5] 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.

[6] E. G. Boman and S. Rajamanickam. A study of combinatorial issues in a sparse hybrid solver. In *Proc. of SciDAC 2011*, 2011.

[7] D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Ozgunner. Distributed-memory parallel algorithms for distance-2 coloring and related problems in derivative computation. *SIAM J. Sci.Comput.*, 32(4):2418–2446, 2010.

[8] D. Bozdağ, A. H. Gebremedhin, F. Manne, E. G. Boman, and U. V. Catalyurek. A framework for scalable greedy coloring on distributed-memory parallel computers. *Journal of Parallel and Distributed Computing*, 68(4):515–535, 2008.

[9] U. Catalyurek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, and L. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Proc. of 21th International Parallel and Distributed Processing Symposium (IPDPS'07)*. IEEE, 2007. Best Algorithms Paper Award.

[10] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. A. Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distrib. Comput.*, 69:711–724, August 2009.

[11] U. V. Catalyurek, F. Dobrian, A. Gebremedhin, M. Halappanavar, and A. Pothen. Distributed-memory parallel algorithms for matching and coloring. In *2011 International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), Workshop on Parallel Computing and Optimization (PCO'11)*, pages 1966–1975, 2011.

[12] U. V. Catalyurek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen. Multithreaded graph coloring algorithms. Submitted to Parallel Computing, 2011.

[13] J. Chen and I. Safro. Algebraic distance on graphs. (under revision in SIAM Journal of Scientific Computing), 2010.

[14] C. Chevalier and E. G. Boman. An accurate hypergraph model for mesh partitioning. In *Proc. of SIAM Workshop on Combinatorial Scientific Computing (CSC09)*, Seaside, CA, Oct 2009.

[15] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228, New York, NY, USA, 2009. ACM.

[16] Y. Choi and W. Szpankowski. Compression of graphical structures. In *ISIT'09: Proceedings of the 2009 IEEE Symposium on Information Theory*, pages 364–368, Piscataway, NJ, USA, 2009. IEEE Press.

[17] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE, 2006.

[18] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[19] F. Dobrian, M. Halappanavar, and A. Pothen. Exact and approximation algorithms for vertex-weighted matching. Submitted for publication, 2011.

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

[21] A. Gebremedhin, A. Pothen, A. Tarafdar, and A. Walther. Efficient computation of sparse Hessians using coloring and automatic differentiation. *INFORMS Journal on Computing*, 21(2):209–223, 2009.

[22] A. Gebremedhin, A. Pothen, and A. Walther. Exploiting sparsity in Jacobian computation via coloring and automatic differentiation: A case study in a Simulated Moving Bed process. In C. Bischof et al, editor, *Proceedings of AD2008*, volume 64 of *Lecture Notes in Computational Science and Engineering*, pages 339–349. Springer, 2008.

[23] A. Gebremedhin, A. Tarafdar, F. Manne, and A. Pothen. New acyclic and star coloring algorithms with applications to Hessian computation. *SIAM J. Sc. Com.*, 29:1042–1072, 2007.

[24] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen. ColPack: Graph Coloring Software for Sparse Derivative Matrix Computation and Beyond. Submitted to ACM Transactions on Mathematical Software, 2010.

[25] A. Griewank, D. Juedes, and J. Utke. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Softw.*, 22:131–167, 1996.

[26] L. Grigori, E. Boman, S. Donfack, and T. Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization. *SIAM J. Sci. Comp.*, 32(6):3426–3446, 2010.

[27] M. Halappanavar. *Algorithms for vertex-weighted matching in graphs.* PhD thesis, Old Dominion University, 2009.

[28] M. Halappanavar, J. Feo, O. Villa, A.Tumeo, F. Dobrian, and A. Pothen. Approximate weighted matching on emerging manycore and multithreaded architectures. Submitted for publication, 2011.

[29] B. Hendrickson and R. Leland. The chaco user's guide, version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, 1993.

[30] K. E. Jansen. A stabilized finite element method for computing turbulence. *Computer Methods in Applied Mechanics and Engineering*, 174(3-4):299 – 317, 1999.

[31] G. Karypis and V. Kumar. ParMETIS 1.0: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Dept. Computer Science, University of Minnesota, 1997. *http://www.cs.umn.edu/˜metis.*

[32] Y. Kawajiri and L.Biegler. Large scale nonlinear optimization for asymmetric operation and design of Simulated Moving Beds. *J. Chrom. A*, 1133:226–240, 2006.

[33] A. Lyons. Acyclic and star colorings of joins of graphs and an algorithm for cographs. In *Proceedings of the Eighth Cologne Twente Workshop on Graphs and Combinatorial Optimization(CTW09)*, pages 199–204. 2009.

[34] S. H. K. Narayanan, B. Norris, P. Hovland, D. C. Nguyen, and A. H. Gebremedhin. Sparse Jacobian computation using ADIC2 and ColPack. Preprint ANL/MCS-P1841-0211, Mathematics and Computer Science Division, Argonne National Laboratory, 2011. To appear in ICCS 2011.

[35] S. H. K. Narayanan, B. Norris, and B. Winnicka. ADIC2: Development of a component source transformation system for differentiating C and C++. In *Proceedings of International Conference on Computational Science (ICCS 2010)*, pages 1845–1853. Elsevier, 2010.

[36] M. A. Patwary, A. H. Gebremedhin, and A. Pothen. New Multithreaded Ordering and Coloring Algorithms for Multicore Architectures. In *Euro-Par 2011*, 2011. to appear.

[37] F. Pelligrini. SCOTCH 3.4 user's guide. Research Rep. RR-1264-01, LaBRI, Nov. 2001.

[38] D. Ron, I. Safro, and A. Brandt. A fast multigrid algorithm for energy minimization under planar density constraints. *Multiscale Modeling & Simulation*, 8(5):1599–1620, 2010.

[39] D. Ron, I. Safro, and A. Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

[40] I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. *Journal of Experimental Algorithmics*, 13:1.4–1.20, 2008.

[41] I. Safro and B. Temkin. Multiscale approach for the network compression-friendly ordering. *J. of Discrete Algorithms*, 9:190–202, June 2011.

[42] H. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2-3):135 – 148, 1991. Parallel Methods on Large-scale Structural Analysis and Physics Applications.

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