SAND2010-1967P

Parallel Partitioning, Coloring, and Ordering for Scientific Computing

Erik G. Boman* Umit V. Catalyurek Cédric Chevalier
Karen D. Devine

March 29, 2010

Abstract

Three combinatorial problems that often arise in scientific computing, are partitioning, coloring, and ordering. In this chapter, we show examples of how these methods may be used in applications. We briefly describe the Zoltan software toolkit, which provides parallel algorithms and implementations for all these problems.

1 Introduction

Combinatorial problems often arise within the context of scientific computing. Only recently has combinatorial scientific computing (CSC) gained recognition as a fruitful research area. In this chapter, we will focus on three important CSC problems: Partitioning (load balancing), graph coloring, and sparse matrix ordering. We show by examples how these problems may occur in an application, and briefly describe how the *Zoltan toolkit* can be used to tackle these problems. Our target audience is computational scientists, who may have little prior knowledge of CSC.

The Zoltan toolkit [21, 22, 4] was originally developed to partition and distribute data among processors for parallel computing. Over the years, its scope has expanded to also include matrix ordering and graph coloring. Zoltan is a library written in C and has interfaces in C++ and Fortran. An important feature of Zoltan is that it is "data structure neutral", meaning that it interfaces with the application code primarily via callback function, not by passing explicit data structures. This makes it easier for Zoltan to work with a wide variety of applications. A short tutorial can be found in [23].

The rest of this chapter is organized as follows: First we discuss partitioning and load-balancing. Then we turn to coloring for a Jacobian matrix. Finally, we discuss ordering, with focus on fill-reducing ordering for sparse direct solvers.

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

2 Partitioning and Load Balancing

For efficient parallel computing, it is essential to balance the work (load) among processors (or processes). On many architectures, especially distributed-memory computers, it is also important to minimize the communication among processors. In this section we assume the "owner computes" model, where computations are performed by the processor (core) that owns the data. This gives rise to the *partitioning* problem: How to partition data among processors such that all processors have roughly equal load (work) and communication is minimized? Many algorithms have been proposed for this problem. Mainly, they fall into two categories: Geometric, or combinatorial (based on graph or hypergraph models). Zoltan implements several methods of each type.

From the application point of view, it is important to first identify the data objects to be distributed among processors. For mesh-based computations, this can be elements (regions) or nodes (vertices). For matrix-based computations, this is typically matrix rows or columns. For molecular dynamics, this is typically particles. Geometric methods are simple to use since they just require the coordinates of each data point and no connectivity. However, graph/hypergraph methods typically produce partitions with lower communication cost, precisely because they exploit such connectivity information. Next we examine how to partition meshes.

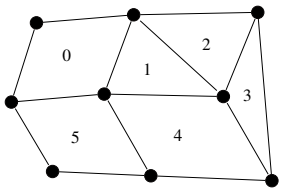
2.1 Partitioning for mesh computations.

A mesh is a discretization of the computational domain, mainly used to solve partial differential equations (PDEs). We focus on unstructured meshes commonly used in finite element and finite volume methods. The mesh is a set of entities called elements or regions which are geometrically limited by nodes and by the edges between nodes, as shown in Figure 1(a).

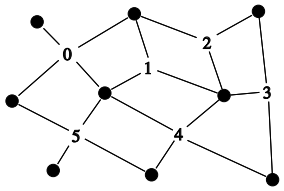
The goal of mesh partitioning is to divide the elements into p subsets (domains) such that the work per processor is approximately constant and that the communication cost is minimized. Typically, communication cost is approximated by the total volume, though other factors (e.g., number of messages) also play a role. Geometric methods are simple and fast. Popular geometric methods are recursive coordinate bisection (RCB) and space-filling curves (SFC). To explicitly limit communication, graph partitioning is often preferred. Note that graph partitioning is used even though it is known to have several deficiences [28]. We show that a hypergraph model is more accurate, and enables new partitioning approaches.

2.1.1 Mesh models

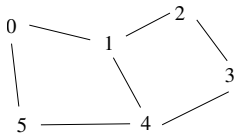
In the following models, we partition the elements. This is usually preferred because many computations are internal to the elements, so parallel codes require each element to be owned by a single processor. However, for a node-centric



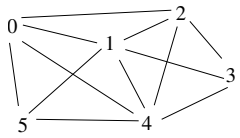
(a) A 2D mesh with 7 elements (3 quadrangles (0,5,4) and 3 triangles(1,2,3)) and 9 nodes.



(b) A hypergraph model corresponding to the previous mesh, where mesh elements are vertices in the graph. Note elements and nodes of the mesh are both present.



(c) Dual graph defined by the faces. Note that the nodes in the mesh are not explicitly represented.



(d) Dual graph defined by the nodes.

Figure 1: A mesh and its different associated models.

code, much of the discussion below still applies if you simply switch elements and nodes.

Dual graph models: In these models, the mesh is modeled as a graph in order to use a graph partitioning tool. These approaches consist in looking only at the elements of the mesh and their adjacencies. The more common relation of adjacency for two elements is to share a face, and in this case we can talk about face-based dual graph (Figure 1(c)). It is also possible to have a relation of adjacency between two elements based on the fact that they share a node, and in this case we speak about node-based dual graph (Figure 1(d)).

Performing graph partitioning on these dual graphs consists in dividing the elements in p disjunct subsets (parts, domains) of the same size, while minimizing the cut edges. In the case of a faced-based dual graph, the partitioning will minimize the number of faces shared between elements of different parts.

With a node-based dual graph, we tend to minimize the number of nodes shared between the parts. However, this model is not exact in terms of minimizing the number of nodes, due to the fact that each node correspond to a clique in the graph to partition and therefore the edge cut is not directly a measurement

of the number of nodes, even if the two are closely correlated.

Hypergraph Model: The number of nodes shared is particularly important for the widely used continuous Galerkin finite elements methods because the communications are on the nodes, not on the faces. As the graph-based approach does not model this accurately, we introduce an hypergraph based approach which can resolve this issue.

This model is perhaps easier to describe first with a bipartite graph, which is a graph with two kinds of vertices. One kind will be the elements of the mesh, the other kind being the nodes of the mesh. Edges are only between two vertices of different types and their meaning in this case is that the element needs this node.

With the hypergraph terminology, this model is defined by the set of vertices which correspond to the elements in the mesh, and a set of hyperedges which correspond to the nodes in the mesh. Hypergraph offers mainly two metrics that are relevant in our case: the number of cut hyperedges, which is the number of shared nodes in the mesh, and a connectivity metric that weights each cut hyperedge with the count the number of participating processors minus one (λ-1 cut metric) which exactly represents the communication volume for a node based mesh method.

2.1.2 Observations and Conclusions

There is, in general, no direct correlation between the minimization of the number of shared faces and the number of shared nodes implied by the mesh partitioning. Nevertheless, in our experience, the partitioning model does not impact the results much in most cases. Indeed, when all the faces of all the elements are the same type, as the separators are mostly contiguous, it is almost the same to compute a face separation or a node separation of the mesh.

However, for meshes with mixed types of elements, the hypergraph model will be more accurate to model the number of nodes shared by different parts. For example, for meshes with tetrahedrons, hexahedrons and pyramids, faces can be of different types, like triangle, quadrangle or any type of polygons, and this information does not appear with the dual graph models.

Moreover, the hypergraph model is intrinsically richer than the dual graph models as it keeps information for both elements and nodes. This will allow more complex objectives for partitioning, like minimizing the number of ghosts vertices or elements, or having a better load balancing of the computation and of the memory. Indeed, when a code uses ghosting of elements, the ghost elements are not modeled during the partitioning and thus are not balanced. This is an important phenomenon as the number of parts become larger but the size of locally owned elements stay constant, like on massive parallel computers with limited local memory (e.g., BlueGene).

Another point is that the structure of the hypergraph is exactly the same as the one of the mesh, thus it will be very easy to update the hypergraph in order to take into account mesh changes to do dynamic load balancing.

Hypergraphs thus offer a more accurate model for mesh partitioning and this

model can be more easily extended to handle the new challenges of petascale computing. Parallel software for hypergraph partitioning is currently available (in Zoltan).

2.2 Partitioning for sparse matrices

In scientific community, sparse matrix partitioning problem have been usually modeled using graphs and hypergraphs [9, 14, 28]. Graphs can be used to model two-way interactions, and hence for applications that cost metric is based on multi-way interactions, they can only provide an approximate solutions [10, 28]. Multi-way interactions, as well as non-symmetric dependencies, can be naturally modeled using hypergraphs. Therefore, during the last decade, hypergraph-based models and methods have gained wide acceptance in the literature for applications like parallelization of spase matrix-vector multiplications [14, 33, 32]. There are three basic hypergraph models, namely, the column-net model [10], the row-net model [10], and the column-row-net (fine-grain) model [11, 14]. The column-net and row-net models are respectively used for one-dimensional (1D) rowwise and 1D columnwise partitioning. The column-row-net model is used for two-dimensional(2D) nonzero-based fine-grain partitioning. In all these models, partitioning constraint of maintaining a balance on part weights corresponds to maintaining a computational load balance, and the partitioning objective of minimizing the cutsize exactly corresponds to minimizing the total communication volume.

There are other methods such as the orthogonal recursive bisection [33], the jagged-like [14], the checkerboard [12], and the hybrid [2] (see Chapter **??** for details) partitioning methods which utilize various combination of the basic hypergraph models to achieve different 2D partitionings.

2.3 Zoltan Capabilities

Zoltan supports partitioning through the `Zoltan_LB_Balance` function. Both graph and hypergraph partitioning are supported, as well as several geometric methods. The partitioning method is selected by the parameter `LB_METHOD`. Zoltan provides a common interface for a collection of partitioning algorithms, making it easy to try new methods and to compare.

Depending on which partitioning method one wishes to use, one must implement a few query (callback) functions, to describe the application data. There are separate query functions for geometry, graphs and hypergraphs.

A unique feature of Zoltan is its parallel hypergraph partitioner, PHG [22, 15]. This is a fully parallel multilevel partitioner that uses the hypergraph model. Zoltan also provides interfaces to two traditional graph partitioners: Scotch [17] and ParMetis [30]. In general, hypergraph partitioning produces better quality partitions but takes longer to compute.

Zoltan can also perform repartitioning for dynamic load balancing. This is useful to reduce the data migration cost. This feature has been integrated into the hypergraph partitioner [15].

Applications that use Zoltan for partitioning and load-balancing include the finite element simulation framework SIERRA, the circuit simulator Xyce, the Maxwell particle-in-cell code Pic3P, and the CFD code PHASTA. Zoltan has been used to partition meshes with over one billion elements.

3 Coloring

Typically, nonlinear problems are solved by a sequence of linear systems, e.g., with Newton's method. Examples are time-dependent PDEs, for example, weather and climate simulation. A significant amount of work (and memory) can be involved in computing the Jacobian matrix $J(x)$ for a smooth, nonlinear function $f(x)$. When f is highly complex, for example, may involve solving a PDE, exact derivatives is impossible. Two common options in this case is to estimate J by finite differences, or to compute it exactly using automatic differentiation (AD). The AD approach typically involves using AD software and is described in other chapters. Here, we outline how coloring is relevant to the finite difference approach.

3.1 Jacobians by finite differences

Suppose we need to explicitly form the Jacobian J. This is typically the case if one plans to solve the linear system $Jx = b$ using a direct solver. We note that using iterative methods like Jacobi-Free Newton-Krylov (JFNK) one only needs J implicitly, e.g., a black-box to compute matrix-vector products. By using finite differences, we can form the kth column of J by the formula $J_k(x) = J(x)e_k \approx (f(x + \epsilon e_k) - f(x))/\epsilon$, where e_k is the kth unit vector and ϵ is a small real number. Since $f(x)$ is typically already known, this costs one function evaluation. Thus, n function evalutions are needed to form the whole $n \times n$ Jacobian this way.

Fortunately, for sparse Jacobians there is a better way. The key idea is to form several columns of the Jacobian at once [19]. Columns that are *structurally orthogonal*, that is, their sparsity patterns do not overlap, can be computed in a single step. Let d be a binary vector with ones in a set of indices corresponding to a group of structurally orthogonal columns. Then we can estimate Jd using a single function evaluation along the direction d. The process to form all of J works as follows:

1. Derive the sparsity pattern of J.
2. Compute independent groups of columns of J (e.g., by coloring).
3. Compute the compressed Jacobian: $\hat{J} = JD$, where D is a binary matrix, corresponding to the grouping of the columns.
4. Uncompress $\hat{J}$ to recover J.

We wish to minimize the number of columns in the compressed Jacobian, which corresponds to the number of function evaluations. This combinatorial problem is a coloring problem, see [24]. One approach [18] is to form the column intersection graph, G_C, and color the vertices such that no two adjacent vertices have the same color (distance-1 coloring). A second approach, which uses less memory, is to compute a partial distance-2 coloring directly on the structure of J. See [24] for details. Both approaches can reduce the number of function evaluations by an order of magnitude (or more) compared to the naive method (one evaluation per column).

3.2 Preconditioning for iterative solvers.

In the previous section, we assumed we required an explicit Jacobian to be used in a direct solver. However, for large problems iterative solvers may be more efficient. In this case, J is only needed implicitly as an operator. Nevertheless, finite differencing and coloring can be very useful to form a preconditioner for J. One example is the parallel ocean code POP (part of the Community Climate System Model), which uses a JFNK method. Still, to form a preconditioner, an explicit Jacobian is needed as input. An interesting point is that an approximation to the Jacobian is often sufficient in this case, since the preconditioner only need solve the problem inexactly. Thus, we can form $\tilde{J} \approx J$, where $\tilde{J}$ contains only a subset of the entries in the true Jacobian J. Generally, $\tilde{J}$ requires less colors than J, and therefore fewer function evaluations.

3.3 Zoltan Capabilities.

Zoltan supports graph coloring by the `Zoltan_Color` function. The graph is given to Zoltan through callback functions. The output is a vector of integers, corresponding to different colors. Zoltan currently supports distance-1 [3, 7], distance-2 [8, 6], and partial distance-2 [6] coloring, specified by the parameter `COLOR_PROBLEM`. These are the most common coloring problems. Coloring problems arising from Hessian computations (star and acyclic) are planned for the future.

A unique feature of Zoltan is the ability to compute coloring in parallel on distributed memory platforms. The parallel coloring algorithms in Zoltan have been shown scalable up to thousands of processors [5].

4 Ordering

The *ordering* of data can affect performance in several ways. First, a reordering of data in an application can give better locality and thus improved performance due to better use of cache or fast memory. This has become increasingly important as memory hierarchies have grown deeper and become more complex. However, we will rather focus on a more specific problem: How to reorder a sparse matrix such as to minimize fill in a sparse direct solver?

4.1 Fill-reducing Ordering of Sparse Matrices

In many scientific computing applications, it is necessary to solve the system $Ax = b$. Either direct or iterative methods can be used. When A is ill-conditioned and no good preconditioner is known, direct solvers are preferred since iterative solvers will converge slowly or not at all. The work and memory required to solve $Ax = b$ depends on the fill in the factorization. A standard technique is therefore to permute (reorder) the matrix A such that one solves the equivalent system $(PAQ)(Q^T x) = Pb$, where P and Q are permutation matrices. This reordering is usually performed as a preprocessing step, and depends only on the nonzero structure not on the numerical values. In the symmetric positive definite case, a symmetric permutation is used to preserve symmetry and definiteness, i.e., $Q = P^T$. In the nonsymmetric case, numerical pivoting is required for stability during the factorization phase. Generally, only a column permutation is therefore performed in the setup phase to reduce fill.

There are two broad categories of fill-reducing ordering methods: *minimum degree* [31] and *nested dissection* [25] methods. The minimum-degree class takes a local perspective and the algorithms are greedy. The algorithms are inherently sequential but fast to compute. The nested dissection methods, on the other hand, take a global perspective. Typically, recursive bisection is used, which is suited for parallel processing. Nested dissection methods usually give less fill for large problems, and are better suited for parallel factorization. In practice, hybrid methods that combine nested dissection with a local ordering are most effective.

4.2 Symmetric Positive Definite Case

Most of the state-of-the-art symmetric ordering tools [17, 27, 29] are hybrid methods that combine multilevel graph partitioning for computing nested dissections with a variant of minimum degree for local ordering. The main idea of nested dissection is as follows. Consider a partitioning of vertices (V) into three sets: V_1, V_2 and V_S, such that the removal of V_S, called *separator*, decouples two parts V_1 and V_2. If we order the vertices of V_S after the vertices of V_1 and V_2, i.e., if the elimination process starts with vertices of either V_1 or V_2, no fill will occur between the vertices of V_1 and V_2. Furthermore, the elimination process in V_1 and V_2 are independent from each other and their elimination only incurs fill to themselves and V_S. In the nested dissection, the ordering of the vertices of V_1 and V_2 is simply computed by applying the same process recursively. In the hybrid methods, recursion is stopped when a part becomes smaller than predetermined size, and minimum degree-based local ordering is used to order the vertices in that part.

Catalyurek et al. [16, 13] proposed use of hypergraphs for ordering sparse matrices. Use of hypergraphs in this context have two advantages. First, although multilevel paradigm is a perfect fit in partitioning, in the ordering context it has a small flaw: a vertex separator found in the coarser graphs may not be minimal when separator projected to a finer graph in the multilevel hierarchy.

Use of hypergraphs eliviates this problem. Second, in many applications, like the solution of linear programming problems using an interior point method, symmetric matrices are in the form of $M = AA^T$ (or $M = AD^2A^T$) where A is a rectangular matrix. In such cases, one can achieve better and faster orderings for M by simply running nested dissection on A using hypergraph models [10, 1].

4.3 Unsymmetric Case

In unsymmetric direct solvers, partial pivoting is required for numerical stability and to avoid breakdown. Partial pivoting is usually performed by looking down a column for a large entry and then swapping rows. This defines a row permutation but it is determined during the numerical factorization. Therefore, fill-reducing ordering has focused on column ordering. Consider $\hat{A} = PAQ$. The column permutation Q can be computed in a preprocessing step based on the structure of A. Note that P is not known at this point, so Q should reduce fill for *any* P.

There are two popular approaches to column ordering. First, one can simply symmetrize A and compute a nested dissection ordering for either $A + A^T$ or A^TA. This is attractive because one can use the same algorithms and software as in the symmetric case. However, ordering based on $A + A^T$ does not necessarily give good quality ordering for A. Ordering based on A^TA generally works better and has a theoretical foundation since the fill for LU factors of A is contained within the Cholesky factor of A^TA. Unfortunately, A^TA is often much denser than A and takes too much memory to form explicitly. The second approach is therefore to order the columns of A to reduce fill in the Cholesky factor of A^TA but without forming A^TA. Several versions of minimum degree have been adapted in this way; the most popular is COLAMD [20]. A limitation of COLAMD is that it is inherently sequential and not suited for parallel computations.

A third approach, the HUND method, was recently proposed by Grigori et al. [26]. The idea is to use hypergraphs to perform a unsymmetric version of nested dissection, then switch to CCOLAMD on small subproblems. Hypergraph partitioning is used to obtain a singly bordered block diagonal (SBBD) form. It is shown in [26] that the fill in LU factorization is contained within the blocks in the SBBD form, even with partial pivoting. Experiments with serial LU factorization show that HUND compares favorably to other methods both in terms of fill and flops. Perhaps the greatest advantage of HUND, though, is that it can both be computed in parallel and produces orderings well suited for parallel factorization.

We remark that several parallel direct solvers (e.g., MUMPS) currently cannot perform column reordering but only apply symmetric permutations. In this case, HUND ordering can be applied symmetrically. The row reordering is not necessary but also does not destroy sparsity. In fact, with classical partial pivoting, the row permutation does not matter but with threshold pivoting the row permutation will make a difference (though we expect the effect to be small).

4.4 Zoltan Capabilities

Zoltan provides ordering algorithms both for locality and to reduce fill via the `Zoltan_Order` function. The locality-enhancing orderings are RCM and space-filling curves. These are intended to be used locally on each processor (core), and run in serial. The fill-reducing orderings are intended for solving large systems, and thus must be computed in parallel. The focus in Zoltan is on nested-dissection methods, since these generally work best on large problems. Classic nested dissection for symmetric problems is provided through interfaces to the third-party libraries Scotch [17] and ParMetis [30]. For nonsymmetric problems, Zoltan contains a native parallel implementation of HUND. This can of course also be applied to symmetric systems, but works best in the nonsymmetric case.

Zoltan does not provide a sparse direct solver, so should rather be used to compute a permutation vector before calling your direct solver of choice. Most direct solvers allow symmetric permutations, but only a few allow nonsymmetric permutations ($P \neq Q$). Zoltan ordering has been tested with several versions of SuperLU.

References

[1] C. Aykanat, A. Pınar, and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 26(6):1860–1879, 2004.

[2] Rob Bisseling, Tristan van Leeuwen, and Umit V. Catalyurek. Combinatorial problems in high-performance computing: Partitioning. In Uwe Naumann, Olaf Schenk, Horst D. Simon, and Sivan Toledo, editors, *Abstract, Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[3] E.G. Boman, D. Bozdag, U. Catalyurek, A. Gebremedhin, and F. Manne. A scalable parallel graph coloring algorithm for distributed memory computers. In *Euro-Par 2005*, volume 3648 of *LNCS*, pages 241–251. Springer, 2005.

[4] Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Lee Ann Riesen, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, William Mitchell, and James Teresco. *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4748W `http://www.cs.sandia.gov/Zoltan/ug_html/ug.html`.

[5] D. Bozdag, U. Catalyurek, F. Dobrian, A. Gebremedhin, M. Halappanavar, and A. Pothen. Poster abstract - combinatorial algorithms enabling scien-

tific computing: Petascale algorithms for graph coloring and matching. In *Proc. of 21st Supercomputing Conference*, 2009.

[6] D. Bozdag, U.V. Catalyurek, A.H. Gebremedhin, F. Manne, E.G. Boman, and F. Ozguner. Distributed-memory parallel algorithms for distance-2 coloring and their application to derivative computation. *SIAM Journal of Scientific Computing*, 2009. under review.

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

[8] Doruk Bozdağ, Umit Catalyurek, Assefaw H. Gebremedhin, Fredrik Manne, Erik G. Boman, and Füsun Özgüner. A parallel distance-2 graph coloring algorithm for distributed memory computers. In *The 2005 International Conference on High Performance Computing and Communications (HPCC-05)*, 2005.

[9] Ü. Çatalyürek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplications. *Lecture Notes in Computer Science*, 1117:75 – 86, 1996.

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

[11] Ü. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *Proc. IPDPS 8th Int'l Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001)*, April 2001.

[12] Ü. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *Proc. Supercomputing 2001*. ACM, 2001.

[13] Umit V. Catalyurek, Cevdet Aykanat, and Enver Kayaaslan. Hypergraph partitioning-based fill-reducing ordering. Technical Report OSUBMI-TR-2009-n02 and BU-CE-0904, The Ohio State University, Department of Biomedical Informatics and Bilkent University, Computer Engineering Department, 2009. submitted for publication.

[14] Umit V. Catalyurek, Cevdet Aykanat, and Bora Ucar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. Technical Report OSUBMI_TR_2008_n04, The Ohio State University, Department of Biomedical Informatics, 2008. To appear in SIAM J. Sci. Comput.

[15] U.V. Catalyurek, E.G. Boman, K.D. Devine, D. Bozdag, R.T. Heaphy, and L.A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Proc. of 21th International Parallel and Distributed Processing Symposium (IPDPS'07)*. IEEE, 2007.

[16] Ü. V. Çatalyürek. *Hypergraph Models for Sparse Matrix Partitioning and Reordering*. PhD thesis, Bilkent University, Computer Engineering and Information Science, Nov 1999.

[17] C. Chevalier and F. Pellegrini. PT-SCOTCH: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6–8):318–331, 2008.

[18] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring. *SIAM J. Numer. Anal.*, 20(1):187–209, 1983.

[19] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.

[20] T.A. Davis, J.R. Gilbert, S.I. Larimore, and E.G. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. on Mathematical Software*, 30(3):353–376, 2004.

[21] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[22] K.D. Devine, E.G. Boman, R.T. Heaphy, R.H. Bisseling, and U.V. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE, 2006.

[23] K.D. Devine, E.G. Boman, L.A. Riesen, U.V. Catalyurek, and C. Chevalier. Getting started with zoltan: A short tutorial. In Uwe Naumann, Olaf Schenk, Horst D. Simon, and Sivan Toledo, editors, *Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings, page 10 pages, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[24] Assefaw H Gebremedhin. What color is your jacobian? graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, 2005.

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

[26] L. Grigori, E.G. Boman, S. Donfack, and T.A. Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization. Technical Report 2008-1290J, Sandia National Laboratories, 2008. in review.

[27] B. Hendrickson and E. Rothberg. Effective sparse matrix ordering: just around the bend. In *Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing*, 1997.

[28] Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519 – 1534, 2000.

[29] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Dept. Computer Science, University of Minnesota, 1995. *http://www.cs.umn.edu/~karypis/metis*.

[30] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Sci. Comp.*, 20(1):359–392, 1998.

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

[32] Bora Ucar, Umit V. Catalyurek, and Cevdet Aykanat. A matrix partitioning interface to patoh in matlab. *Parallel Computing*, 2010. to appear.

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