On partitioning problems with complex objectives

Kamer Kaya
CERFACS
Toulouse, France
kamer@cerfacs.fr

François-Henry Rouet
Université de Toulouse
INPT (ENSEEIHT)-IRIT
Toulouse, France
frouet@enseeiht.fr

Bora Uçar
CNRS and ENS Lyon
Lyon, France
bora.ucar@ens-lyon.fr

February 2011

Abstract

Hypergraph and graph partitioning tools are used to partition work for efficient parallelization of many sparse matrix computations. Most of the time, the objective function that is reduced by these tools relates to reducing the communication requirements, and the balancing constraints satisfied by these tools relate to balancing the work or memory requirements. Sometimes, the objective sought for having balance is a complex function of the partition. We describe some important class of parallel sparse matrix computations that have such balance objectives. For these cases, the current state of the art partitioning tools fall short of being adequate. To the best of our knowledge, there is only a single algorithmic framework in the literature to address such balance objectives.

We propose another algorithmic framework to tackle complex objectives and experimentally investigate the proposed framework.

Keywords

Hypergraph partitioning, graph partitioning, sparse matrix partitioning, parallel sparse matrix computations, combinatorial scientific computing.

1 Introduction

Hypergraph and graph partitioning tools are used to partition work for efficient parallelization of many sparse matrix computations. Roughly speaking, the vertices represent the data and the computations, and the (hyper)edges represent dependencies of the computations into the data. For a parallel system of K processors, partitioning the vertices into K disjoint parts can be used to partition the data and the total work among the processors by associating each part with a unique processor. Therefore, a successful application of such partitioning tools should assign almost equal work/data to processors and should reduce the communication requirements. The first of these goals is attained by associating weights to vertices and then by guaranteeing that the K resulting parts have almost equal weights, defined as a function of individual vertex weights. The second goal is achieved by reducing a function related to the (hyper)edges that straddle two or more parts. There are a number of publicly available, state of the art tools. Among those tools Chaco [20], MeTiS [22], Mondriaan [35], PaToH [13], Scotch [27], and Zoltan [7] are widely used in many applications.

Sometimes the objective sought for having balance is simple. By this we mean that one can assign weights to the vertices before partitioning, and then measure the weight of a part by simply adding up the weights of vertices in that part. For example, if a vertex represents a row of a sparse matrix, then the number of nonzeros in that row can be used as the weight of the corresponding vertex. Then upon partitioning, the weight of a part corresponds to the total number of nonzeros assigned to that part, and hence balance can be obtained among processors easily by using the off-the-shelf partitioning tools listed above. This standard approach, however, is not sufficient for many important class of sparse matrix computations. Pınar and Hendrickson [28] discuss four different class of computations for which the standard approach falls short of achieving balance on the computational load of the processors. The

mentioned computations include FETI class of domain decomposition-based solvers, iterative methods with incomplete LU or Cholesky preconditioners, overlapped Schwarz solvers, and direct methods based on multifrontal solvers.

The main reason which renders the standard tools inadequate is aptly called a chicken-and-egg problem by Pınar and Hendrickson [28]. The objective sought for balancing is a complex function of the partition. By this, we mean that one cannot assess the balance by looking at a set of a priori given vertex weights; rather, the weight of a part is a complex function of a partition. The partitioner needs to compute the balance function to satisfy the objective but the function cannot be computed without obtaining the partition. We give a toy example for this phenomena here. Suppose that we want to partition a square matrix rowwise for efficient parallel computation of $y \leftarrow Ax$ where the input vector x and the output vector y are assumed to be assigned to the processors conformally with the rows of A, i.e., a processor holding row i of A holds the vector entries y_i and x_i as well. Suppose we want to obtain balance on the number of nonzero entries with which the scalar multiply-add operations with the x vector entries can be computed without communication. The objective is complex, because we cannot know which entries in a row will need an x-vector entry residing in another processor.

We discuss three special forms in sparse matrices in the next subsection. These forms embody most of the data partitioning approaches for efficient parallelization of sparse matrix computations that have complex balancing objectives. In Section 1.2, we review what is available in the literature for the same problem and highlight the main advantages of the proposed framework. Section 2 includes most of the background material. In Section 3, we propose a general framework for the problem of partitioning for complex objectives. This section is divided into three subsections, each being devoted to one of the special forms along with experiments on a set of matrices. We present our conclusions and outline some future work in Section 4.

1.1 Problem definition

Consider the following three forms of an $m \times n$ sparse matrix A for a given integer $K > 1$:

$$A_{SB} = \begin{bmatrix} A_{11} & & & A_{1S} \\ & A_{22} & & A_{2S} \\ & & \ddots & \vdots \\ & & A_{KK} & A_{KS} \end{bmatrix} \quad (1) \qquad A_{BL} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{bmatrix} \quad (2)$$

$$A_{DB} = \begin{bmatrix} A_{11} & & & A_{1S} \\ & A_{22} & & A_{2S} \\ & & \ddots & \vdots \\ & & A_{KK} & A_{KS} \\ A_{S_1} & A_{S2} & \cdots & A_{SK} \quad A_{SS} \end{bmatrix} . \quad (3)$$

The first form A_{SB} of equation (1) is called the singly bordered block-diagonal form (although this does not specify whether the border is formed by rows or columns, we assume a columnwise border throughout the paper). The second one A_{BL} of equation (2) is called the block form, for lack of a better name. The third one A_{DB} (3) is called the doubly bordered block diagonal form. All these three forms have different uses in parallel sparse matrix computations. We assume that these forms are going to be used to partition the matrices among K processors. The following cases are common (see for example [4, 19, 33]). In A_{SB} and A_{BL} forms each processor holds a row stripe, i.e., processor k holds, respectively, $[A_{kk} \;\; A_{kS}]$ and $[A_{k1} \cdots A_{kk} \cdots A_{kK}]$. In A_{DB}, a processor holds the arrow-head formed by the blocks A_{kk}, A_{kS} and A_{Sk}, and perhaps parts of or all of A_{SS}.

A typical application of the singly bordered form A_{SB} arises in the context of parallel QR factorization computations (this is described in sufficient details in [4]). In this parallel factorization, processors factorize their diagonal submatrices and update the submatrices in their border in a first step simultaneously. Then, after a global synchronization, the border block is factored in a second step. Although the computational requirement for sparse QR is not a linear function of nonzeros in a matrix (see [16, Chapter 5]), having balance on the number of nonzeros on the diagonal blocks will usually help (see [4]). Therefore, in an efficient parallelization, the size of the border should be kept small, and processors should have balanced load on the diagonal blocks. Again, having balance among the computational

costs associated with the diagonal blocks is a complex function which cannot be evaluated without a given partition. Furthermore, one would want to have balance in terms of total number of nonzeros per row stripe as well with a hope to have better balance during local factorization step.

Consider a well-known iterative method (such as CG, BiCG, and GMRES) for solving a linear system $Ax = y$ with a block diagonal preconditioner, where no-fill LU or incomplete Cholesky factorizations are used for application of the preconditioner (see [29] for such preconditioners and their use in iterative methods). In these iterative methods, at each iteration, sparse matrix vector multiplies (SpMxv) with A and/or A^T are performed and the preconditioner is applied. For an efficient parallelization of an iteration, the matrix A can be permuted into A_{BL}. Within this setting, processors should have balanced number of nonzeros for load balanced SpMxv operations with A, and they should also have almost equal number of nonzeros in the associated diagonal blocks (i.e., A_{kk} for processor k) for load balanced preconditioner application. Furthermore, communication cost should also be reduced. A common metric for the communication cost is the total volume of communication which is simply equal to the number of nonzero off diagonal column segments in the A_{BL} (see [19, 32]). As the pattern of the preconditioner is not available beforehand, methods in [33, 34] are not applicable. Furthermore, balancing the number of nonzeros in the diagonal blocks is a complex function that cannot be evaluated without a given partition. Notice that the requirements of the partitioning problem here correspond to those of the toy example mentioned earlier.

The applications of A_{DB} include FETI class of domain decomposition-based solvers [17, 26, 28], preconditioned iterative methods [5], and hybrid solvers [25, 36]. In these applications the matrix A is square and usually symmetric in pattern. In these approaches, a direct or an iterative method is applied to the diagonal blocks, a function of these diagonal blocks are applied to the border blocks to compute contributions to the solves with the A_{SS} block, and another solve or factorization is performed on the possibly updated A_{SS} (see also discussions in [26, 28]). For efficient parallelization, diagonal blocks A_{kk} should incur closer cost during factorization or during solves with them; the borders should have balanced number of nonzeros; and the size of the border should be small. Again, balance on the loads regarding the diagonal blocks and the off diagonal blocks are complex functions that cannot be evaluated without a given partition.

1.2 Related work and contributions

The problem of partitioning for complex objectives was studied before for specific problems [6, 26, 32] and in a general setting [28] with some specific applications. Although these studies address different objectives, their framework is very similar. Moulitsas and Karypis [26] call the framework as predictor-corrector. In this predictor-corrector framework, a partitioning is obtained using standard tools and with standard (simple) objectives in the predictor step. Then, the partition is evaluated for the complex objectives and refinements to the current partition are performed in the corrector step. Certain methods in [6, 32] do not go back and forth between different objectives, rather they fix one of them and try to improve the others by exploiting flexibilities. On the other hand, the specific approach proposed in [26] and the general framework formalized in [28] apply iterative improvement based heuristics to improve the partition for all objectives.

There are a few difficulties and challenges arising in the corrector step. First of all, in order to efficiently compute and evaluate the functions, most of the time, large two-dimensional data structures are required where one of the dimensions is K (also true for the simple objectives). This is a common issue arising in K-way, move based refinement algorithms [3, 30]. Secondly, efficient mechanisms that avoid cycles in move based approaches are hard to design. Furthermore, ties among the gains of moves arise almost always, and effective and efficient tie-breaking mechanisms are hard to design and implement for K-way refinement scheme (some efficient mechanisms exist for recursive bisection based approaches, see [1]). Therefore, vertices are usually visited in a random order and best moves are performed (as in [3, 26]). This heuristic, although can be helpful, can also be very shortsighted.

The direct K-way partitioning method can handle complex partitioning objectives. This can simply be accomplished by plugging in such refinement heuristics into the partitioning routine (instead of the standard refinement heuristics). A direct K-way partitioning method thusly modified will resemble to the predictor-corrector approach outline above. The same difficulties will therefore arise. Furthermore, trying to refine for all objectives will likely make the partitioner stuck to a very small neighborhood in the search space. Although for simple objective functions a K-way direct refinement approach is shown

to be better than a recursive bisection based approach [3] in a similar setting, there is no evidence that this will be the case for complex objective functions.

We propose another approach for partitioning for complex objectives. The main idea is to use the recursive bisection based partitioning scheme and to evaluate the complex functions with respect to the existing coarser partition obtained as a result of the previous bisection steps. Once the functions are evaluated, some weights are assigned to the vertices, as the complex functions with respect to the coarser partition are now simple. For the main bisection method, we therefore use available tools as a black box. The advantages of this framework is that one does not need to write a refinement routine, and the framework is easily applicable to graph and hypergraph models with differing objective functions. We will apply the framework with the standard hypergraph partitioning tools to address the three partitioning problems (A_{SB}, A_{BL}, and A_{DB} with the complex objectives) described above.

2 Background

We collect some background material on graph and hypergraph partitioning problems and recursive bisection based hypergraph partitioning methods for completeness.

2.1 Graph partitioning

Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{N})$, the problem of K-way graph partitioning by vertex separator (GPVS) asks for a set of vertices $\mathcal{V}_S$ of minimum size whose removal decomposes $\mathcal{G}$ into K subgraphs with balanced sizes. The problem is NP-hard [8]. Formally, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ is a K-way vertex partition by vertex separator $\mathcal{V}_S$ if the following conditions hold: $\mathcal{V}_k \subset \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$; $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for $1 \leq k < \ell \leq K$ and $\mathcal{V}_k \cap \mathcal{V}_S = \emptyset$ for $1 \leq k \leq K$; $\bigcup_k \mathcal{V}_k \cup \mathcal{V}_S = \mathcal{V}$; there is no edge between vertices lying in two different parts $\mathcal{V}_k$ and $\mathcal{V}_\ell$ for $1 \leq k < \ell \leq K$; parts have balanced sizes, i.e., $W_{max}/W_{avg} \leq (1 + \varepsilon)$, where W_{max} is the maximum part size, W_{avg} is the average part size (defined as $(|\mathcal{V}| - |\mathcal{V}_S|)/K$), and ε is a given maximum allowable imbalance ratio. See the studies [4, 9, 21] for applications of and the heuristics for the GPVS problem. Although a weighted formulation of GPVS exists, it is usually posed with unit weights for the matrix partitioning purposes.

Let A be a pattern-symmetric matrix, and G_A be its standard graph model. The GPVS of G_A can be used to permute the matrix into double-bordered form. The vertices in the separator define the border, and the vertices in each part define the diagonal blocks. Achieving balance on part sizes will imply balance on the number of rows in each diagonal block. There is, however, no guarantee that diagonal blocks will have equal number of nonzeros.

Given an edge-weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{N})$, the problem of K-way graph partitioning by edge separator (GPES) asks for a set of edges with the minimum total weight whose removal decomposes the graph into K subgraphs with balanced sizes. The part size definition and the balance criteria are as before. The problem is NP-complete (see the minimum K-cut problem at `http://www.nada.kth.se/~viggo/problemlist/`). In GPES, all vertices therefore are partitioned among K parts with the minimum total weight of the edges straddling different parts.

Let A be a pattern-symmetric matrix, and G_A be its standard graph model. The GPES of G_A can be used to permute the matrix into the block form A_{BL}. If unit weights are used for vertices, then a solution to GPES will result in balance in the number of rows/columns per diagonal block. If the number of nonzeros in a row is used as the weight of the corresponding vertex, then a balance will be obtained in terms of the number of nonzeros per row stripe. There is, however, no guarantee that the diagonal blocks will have equal number of nonzeros. Furthermore, the edge cut is not an exact measure of communication [12, 18].

2.2 Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{E}$. Every net $n_j \in \mathcal{E}$ is a subset of vertices, i.e., $n_j \subseteq \mathcal{V}$. Weights can be associated with the vertices. We use $w(v_i)$ to denote the weight of the vertex v_i.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part is nonempty, i.e., $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$; parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for

$1 \leq k < \ell \leq K$; and the union of parts gives $\mathcal{V}$, i.e., $\bigcup_k \mathcal{V}_k = \mathcal{V}$. A K-way vertex partition of $\mathcal{H}$ is said to be balanced if

$$\frac{W_{max}}{W_{avg}} \leq (1+\varepsilon), \tag{4}$$

where $W_{max} = \max_k\{W(\mathcal{V}_k)\}$, $W(\mathcal{V}_k)$ is the weight of a part $\mathcal{V}_k$ defined as the sum of the weights of the vertices in that part, i.e., $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$, W_{avg} is the average part weight, i.e., $W_{avg} = \sum_{v_i \in \mathcal{V}} w(v_i)/K$, and ε represents the allowed imbalance ratio.

In a partition Π of $\mathcal{H}$, a net that has at least one vertex in a part is said to *connect* that part. *Connectivity set* Λ_j of a net n_j is defined as the set of parts connected by n_j. *Connectivity* $\lambda_j = |\Lambda_j|$ of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* (*external*) if it connects more than one part (i.e., $\lambda_j > 1$), and *uncut* (*internal*) otherwise (i.e., $\lambda_j = 1$). The set of external nets of a partition Π is denoted as $\mathcal{E}_E$. The partitioning objective is to minimize the cutsize defined over the cut nets. There are various cutsize definitions. Two relevant definitions are:

$$cutsize(\Pi) = \sum_{n_j \in \mathcal{E}_E} 1 \,, \tag{5}$$

$$cutsize(\Pi) = \sum_{n_j \in \mathcal{E}_E} (\lambda_j - 1) \,. \tag{6}$$

In (5), each cut net contributes one to the cutsize. In (6), each cut net n_j contributes $\lambda_j - 1$ to the cutsize. Sometimes costs are associated with the nets, in which case those costs enter as a factor into the equations (5) and (6). For our purposes in this paper, we do not associate costs with nets and use the above cutsize definitions. The hypergraph partitioning problem can be defined as the task of dividing the vertices of a hypergraph into K parts such that the cutsize is minimized, while a given balance criterion (4) is met. The hypergraph partitioning problem is known to be NP-hard [24].

A recent variant of the above problem is the multi-constraint hypergraph partitioning [3, 11, 14, 15, 23, 31]. In this variant, a set of T weights are associated with each vertex v, i.e., $w(v,1), w(v,2), \ldots, w(v,T)$. Let

$$W(\mathcal{V}_k, t) = \sum_{v \in \mathcal{V}_k} w(v,t)$$

denote the weight of part $\mathcal{V}_k$ for constraint t. Then a partition Π is said to be balanced if $\forall t \in \{1, 2, \ldots, T\}$

$$\frac{W_{max}(t)}{W_{avg}(t)} \leq (1+\varepsilon(t)), \tag{7}$$

where $W_{max}(t) = \max_k\{W(\mathcal{V}_k, t)\}$, $W_{avg}(t) = \sum_{v_i \in \mathcal{V}} w(v_i, t)/K$, and $\varepsilon(t)$ is a predetermined imbalance ratio for the constraint t. We note that PaToH [13], a commonly used hypergraph partitioning tool, uses the same load imbalance parameter for all constraints.

Different interpretations and applications of hypergraph partitioning can be used to permute a matrix into the A_{BL}, A_{SB} and A_{DB} forms. We will present them in Section 3.

2.3 Recursive bisection based hypergraph partitioning

We remind the reader some important concepts in the recursive bisection based K-way hypergraph partitioning methods. The number of parts K is assumed to be a power of 2 for the ease of presentation, otherwise this is not a requirement (see for example PaToH [13] which uses recursive bisection scheme and can partition into arbitrary number of parts). In this partitioning scheme, the vertices of a given hypergraph are partitioned into two sets recursively. The recursive calls forms a tree with K leaves, and $K-1$ internal nodes each with two children. This tree is referred to as the bisection tree. The first bisection, or the root of the bisection tree, corresponds to partitioning the vertices of the original hypergraph into two. The leaf nodes correspond to the parts, and two parts having the same parent are said to be of the same bisection.

While optimizing the cutsize metric (5), after a bisection step, one discards the cut nets and forms the two hypergraphs with two parts of vertices and the internal nets. This is referred to as discarding cut nets during bisections. On the other hand while optimizing for the other cutsize metric (5), one

splits the cut nets. Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be the two vertex partitions obtained at a bisection. Then for any net $n_j \cap \mathcal{V}_1 \neq \emptyset$, one puts a net containing the vertices $n_j \cap \mathcal{V}_1$ in the hypergraph containing vertices $\mathcal{V}_1$, and for any net $n_j \cap \mathcal{V}_2 \neq \emptyset$, one puts a net containing the vertices $n_j \cap \mathcal{V}_2$ in the hypergraph containing vertices $\mathcal{V}_2$. These are well explained in [13].

3 A framework for complex objectives and its evaluation

We propose a framework within which standard tools of the hypergraph partitioning problem can be used effectively to address complex partitioning objectives. In this framework, we follow the recursive bisection paradigm. The first bisection is performed as it would be done for the single constraint case. Then the subsequent recursive bisection steps use the partial (or coarse) partition information to set secondary constraints and use multi-constraint bisection routines. This way, at each bisection step, the two parts will satisfy a balance constraint approximatively, as the real balance can only be determined after the bisection. The abstract framework is given below, where concrete instantiation for the three complex partitioning problems mentioned in Section 1.1 are given in the following subsections. The initial call has the arguments $R = [1, \ldots, m]$, $C = [1, \ldots, n]$, $K = 2^\ell$ for some ℓ, $low = 1$, and $up = K$ for an $m \times n$ matrix A.

Algorithm 1 $RB(A, R, C, K, low, up)$

Input: A: a sparse matrix
Input: R: row indices
Input: C: column indices
Input: K: number of parts
Input: low, up: id of the lowest and highest numbered parts
Output: *partition*: partition information for the rows
1: form the model of the matrix $A(R, C)$
2: **if** this is not the first bisection step **then**
3: use previous bisection information to set up the second constraints
4: partition into two $\langle R_1, R_2 \rangle \leftarrow$BisectRows$(A(R, C))$ ▶ with standard tools
5: set $partition(R_1) \leftarrow low$ and set $partition(R_2) \leftarrow up$
6: create the two column sets, either use net splitting or net discarding, giving C_1 and C_2
7: $RB(R_1, C_1, K/2, low, (low + up - 1)/2)$ ▶ recursive bisection
8: $RB(R_2, C_2, K/2, (low + up - 1)/2 + 1, up)$ ▶ recursive bisection

The advantage of this approach over the predictor-corrector approach is that the proposed approach enables multi-level refinement (by harnessing such heuristics available in the standard tools) whereas, predictor-corrector approach does not. Writing down a multi-level refinement heuristic for the predictor-corrector approach will indeed troublesome for the reasons outlined in 1.2. On the other hand, when the secondary constraints are not as important as the first one or when different and very loose imbalance ratios are used, predictions might as well turn out to be acceptable, and one would not need to sacrifice the cutsize for addressing different constraints.

For the experiments, we use the multilevel graph and hypergraph partitioning tools MeTiS [22] and PaToH [13]. We have selected a set rectangular, square and pattern unsymmetric, and pattern symmetric matrices from University of Florida sparse matrix collection and partitioned them into $K = \{32, 64, 128\}$ parts. As MeTiS and PaToH include randomized algorithms, we run each experiment 10 times and report the average result.

We now describe the data set. Let m and n denote, respectively, the number of rows and columns of a matrix and Z denote the number of nonzeros. Then, the rectangular matrices satisfy the following properties (from a matrix family in the collection, we have chosen two matrices having the two largest number of nonzero): $n < m < 3n$, $m > 70000$, $n < 1000000$, $2.7183 \times m \leq Z \leq 7500000$. The square pattern unsymmetric matrices have a symmetry score less than 0.90, satisfy the following properties, and they have the largest number of nonzeros in their family in the collection: $70000 < n < 500000$, $2.7183 \times n \leq Z < 7500000$. The pattern symmetric matrices satisfy the following properties and have the largest number of nonzeros in their family: they have a zero-free diagonal, $74000 < n < 400000$, $5 \times n \leq Z < 7500000$. Among all those matrices we discarded ones having more than $3 \times \sqrt{m}$ nonzeros

in a column or more than $3 \times \sqrt{n}$ nonzeros in a row (it is not advisable to partition matrices having large number of nonzeros in a row or column along the rows or columns [15, p.672]). The details of these matrices are given in Table 1.

Table 1: Properties of the matrices used for the experiments. Here, m and n denote, respectively, the number of rows and columns of a matrix, and Z denote the number of nonzeros.

Matrix	m	n	Z
Rectangular matrices			
ch7-9-b5	423360	317520	2540160
ch8-8-b5	564480	376320	3386880
kneser_10_4_1	349651	330751	992252
Square matrices with an unsymmetric pattern			
ch7-8-b5	141120	141120	846720
crashbasis	160000	160000	1750416
epb3	84617	84617	463625
lhr71c	70304	70304	1528092
mac_econ_fwd500	206500	206500	1273389
RFdevice	74104	74104	365580
shyy161	76480	76480	329762
stomach	213360	213360	3021648
twotone	120750	120750	1206265
Matrices with a symmetric pattern			
ASIC_100ks	99190	99190	578890
bmw7st_1	141347	141347	7318399
boneS01	127224	127224	5516602
cage12	130228	130228	2032536
cfd2	123440	123440	3085406
consph	83334	83334	6010480
denormal	89400	89400	1156224
Dubcova3	146689	146689	3636643
engine	143571	143571	4706073
FEM_3D_thermal2	147900	147900	3489300
fem_filter	74062	74062	1731206
finan512	74752	74752	596992
Ga10As10H30	113081	113081	6115633
helm2d03	392257	392257	2741935
Lin	256000	256000	1766400
mono_500Hz	169410	169410	5033796
offshore	259789	259789	4242673
pkustk13	94893	94893	6616827
poisson3Db	85623	85623	2374949
shipsec5	179860	179860	4598604
thermal1	82654	82654	574458
vfem	93476	93476	1434636
xenon2	157464	157464	3866688

3.1 The singly bordered form

The off the shelf method to permute a matrix into a singly bordered form as shown in (1) is to use the column-net hypergraph model. In this model, an $m \times n$ matrix A is represented with a hypergraph $\mathcal{H} = (\mathcal{R}, \mathcal{C})$, where for each row i of A there is a vertex v_i in $\mathcal{R}$, for each column j of A there is a net n_j in $\mathcal{C}$, and $v_i \in n_j$ iff $a_{ij} \neq 0$. Each vertex v_i is assigned a weight of $|\{j : a_{ij} \neq 0\}|$, the number of nonzeros in the corresponding row. Then, partitioning this hypergraph into K parts under the objective function (5) can be used to permute the matrix A into the singly bordered form [4, Section 5]. The rows

Table 2: Results for the A_{SB} experiments. The minimum, the mean, the maximum, the average, and the geometric mean of the ratio of the results obtained by the framework to the results obtained by the standard method. "Cutsize" refers to the number of coupling columns and "Imbal(A_{kk})" refers to the imbalance on the number of nonzeros on the diagonal blocks.

	min	mean	max	avg	geomean
Cutsize	0.78	1.08	1.74	1.11	1.10
Imbal(A_{kk})	0.36	0.64	1.57	0.70	0.67

corresponding to the vertices in part k are permuted before the rows corresponding to the vertices in part ℓ for $1 \leq k < \ell \leq K$. This defines a partial row permutation where the permutation of the rows in a common block is arbitrary. The column permutation is found as follows. The columns corresponding to the nets that are internal to the part k are permuted before those corresponding to the nets internal to the part ℓ for $1 \leq k < \ell \leq K$. Then the coupling columns are permuted to the end. With this approach one thus reduces the number of coupling columns and obtains balance on the number of nonzeros in the row stripes $[A_{kk} \;\; A_{kS}]$, for $k = 1, \ldots, K$. This does not however imply balance on the diagonal blocks A_{kk}. In [4, Section 5], unit weighted vertices in an unconventional partitioning formulation in which one enforces balance on internal nets are used (a related formulation and the associated tool are used also in [10] but the tool is not publicly available). This result in a singly bordered form where the diagonal blocks have balanced number of rows as well as balanced number of columns (but balance on the diagonal blocks is not addressed explicitly).

Our alternative is to use the outlined recursive bisection based framework to minimize the number of coupling columns while trying to obtain balance on the number of nonzeros in the diagonal blocks as well as in the row stripes. For this purpose, for each row vertex v_i, we associate two weights (after the first bisection) in the third line of the framework:

$$\begin{aligned} w(v_i, 1) &= |\{j : a_{ij} \neq 0\}|, \\ w(v_i, 2) &= |\{j : a_{ij} \neq 0 \text{ and column } j \text{ is not cut yet}\}| \; . \end{aligned}$$

Here $w(v_i, 1)$ is the number of nonzeros in row i and kept the same throughout the bisections to have balance in the row stripes $[A_{kk} \;\; A_{kS}]$. On the other hand $w(v_i, 2)$ relates to the diagonal block weight, and by changing $w(\cdot, 2)$ at every bisection we make these weights become closer to the exact weight that will be seen at the end. Notice that each bisection results in two parts with balanced $W(\cdot, 2)$, however, two parts from two different bisections will only be related indirectly, if they lie in the same bisection subtree. As we are interested in the cut-net metric (5), the coupling columns are discarded at the sixth line of the framework.

We have conducted experiments with all the matrices given in Table 1 to compare the framework with the standard method (SM) of partitioning the column net hypergraph model. The partitioner in the bisection step of the framework and in the standard method is PaToH. In all of these experiments, both of the approaches obtained balance on the number of nonzeros per row stripes quite satisfactorily (both are less than 0.04 on average). In 60 instances, both of the methods obtained balance on the number of nonzeros in the diagonal blocks within 10% of the perfect balance. Skipping those results, we normalized the cutsize and the imbalance obtained by the framework to those obtained by SM. Some statistical indicators (the minimum, the mean, the maximum, the average and the geometric mean) about these results are given in Table 2; the results which correspond to the instances that gave the minimum, the maximum and the mean are given in Table 3.

As seen in Table 2 the framework obtains 30% better balance on the number of nonzeros in the diagonal blocks, on average. This comes with an increase about 11% on the number of coupling columns. This degradation in the cutsize is expected as the standard method has only one constraint. Previously, the average increase in the cutsize with the metric (5) is reported to be around 34% (compare tables 2 and 5 in [3]) in the two constraint case. We think therefore that the increase of 11% in the cutsize is well spent to reduce the imbalance of the diagonal blocks by 30%. Surprisingly, the cutsize is even reduced in some cases; this was very rare and most were for the matrix `finan512`.

Table 3: Results for the A_{SB} experiments. The instances give the min, mean, and max values in the previous table.

		Standard method		Framework	
matrix	K	Cutsize	Imbal(A_{kk})	Cutsize	Imbal(A_{kk})
finan512	32	12163	0.11	9494 (min)	0.04
boneS01	64	41590	0.26	44749 (mean)	0.16
twotone	128	10886	0.38	18914 (max)	0.34
RFdevice	128	25547	0.21	26344	0.08 (min)
poisson3Db	32	31414	0.22	34305	0.14 (mean)
shyy161	64	5720	0.07	6084	0.11 (max)

3.2 The block form

For a square matrix, the off the shelf method to permute a matrix into the block form shown in (2) is to use an algorithm for the graph partitioning by edge separators (GPES) problem (for example Chaco [20], MeTiS [22], or Scotch [27]). However, as stated in Section 2.1, GPES does not correspond to our requirements, even with more involved variants (see [19]). We therefore not perform tests with it.

Another off the shelf method is to use the column-net hypergraph model (described in the previous subsection) with the objective function (6). The row permutation is done as in the previous section. The column permutation is determined in a post-process either in a relatively straightforward manner (refereed to as the naive method [32]) or in a way to optimize other communication metrics [6]. A straightforward post-process would be to first permute the internal columns as is done in the previous section and then to permute a coupling column j to the block which has the minimum number of diagonal nonzeros (so far) among those blocks that the coupling column j touches.

The proposed recursive bisection based framework can be used here as follows. As the objective function is (6), we will use the net splitting methodology. While doing so, we designate one of copies as the main copy (uncut nets are already main copies) with an intend to assign the associated column to one of the parts that will be resulting from the recursive calls on the part of the main copy. That is, the net splitting operation marks either the copy in C_1 or the copy in C_2 as the main one. Without loss of generality, consider a split net n_j whose main copy is in C_1. Then in the following bisection $RB(A, R_1, C_1, \ldots)$, the weight of vertex $i \in R_1$ for which $a_{ij} \neq 0$ will bear a weight of one for the split net n_j (that nonzero entry is in the diagonal block), whereas no vertex in R_2 will bear a weight for the same net. In other words, our alternative using the proposed recursive bisection based approach assigns the following weights to the vertices:

$$\begin{aligned} w(v_i, 1) &= |\{j : a_{ij} \neq 0\}|, \\ w(v_i, 2) &= |\{j : a_{ij} \neq 0 \text{ and column } j \text{ is a main copy}\}| \ . \end{aligned}$$

As before, keeping $w(\cdot, 1)$ always equal to the number of nonzeros in the corresponding row results in balance in the row stripes $[A_{k1}, \ldots, A_{kK}]$, whereas $w(\cdot, 2)$ will approximate the number of nonzeros in diagonal blocks. The last level bisections therefore will be almost accurate modulo the bisection itself. Again the weights of two distant parts will be loosely related.

A possible alternative to the above weighting scheme is to use the number of off-diagonal entries as the second weight. The reasoning is that if the parts have balanced weights for the two constraints, then the diagonal blocks will most likely be balanced. We have tried this scheme and observed that this reasoning holds. Observe however that most of the weights would be zero for the second constraint. In such cases the partitioner is forced to put two vertices in the same cut net (whose main copy is not in the associated bisection part) to be in different sides of the bisection to obtain balance. This scheme therefore results in a large increase in the cutsize.

We have conducted experiments with all the square matrices given in Table 1 to compare the framework with the standard method (SM) of partitioning the column net hypergraph model with the objective of minimizing the cutsize given in (6). The partitioner in the bisection step of the framework and in the standard method is PaToH. In all of these experiments, both of the approaches obtained balance on the number of nonzeros per row stripes quite satisfactorily (both are less than 0.06 on average). In 85 instances, both of the methods obtained balance on the number of nonzeros in the diagonal blocks within

Table 4: Results for the A_{BL} experiments. The minimum, the mean, the maximum, the average, and the geometric mean of the ratio of the results obtained by the framework to the results obtained by the standard method. "Cutsize" refers to the number of nonzeros column segments (the total volume of communication) and "Imbal(A_{kk})" refers to the imbalance on the number of nonzeros on the diagonal blocks.

	min	mean	max	avg	geomean
Cutsize	1.00	1.10	1.53	1.15	1.16
Imbal(A_{kk})	0.35	0.78	2.00	0.84	0.75

Table 5: Results for the A_{BL} experiments. The instances in the upper part give the min, mean, and max values in the previous table. All instances in which the standard method obtained better balance than the framework are given in the lower part.

		Standard method		Framework	
matrix	K	Cutsize	Imbal(A_{kk})	Cutsize	Imbal(A_{kk})
consph	64	69009	0.13	75835 (mean)	0.09
fem_filter	128	65924	0.67	100846 (max)	0.68
twotone	64	17210	0.31	25395	0.11 (min)
cfd2	128	63103	0.12	72406	0.10 (mean)
ch7-8-b5	32	140226	0.10	140450 (min)	0.20 (max)
fem_filter	32	27450	0.47	37864	0.51
fem_filter	64	42320	0.59	56541	0.60
fem_filter	128	same as above		same as above	
engine	64	32959	0.08	37847	0.10
engine	128	50358	0.11	55905	0.12
ch7-8-b5	32	140226	0.10	140450	0.20
ch7-8-b5	64	159997	0.19	168090	0.24

10% of the perfect balance. Skipping those results, we normalized the cutsize and the imbalance obtained by the framework to those obtained by SM. Some statistical indicators (the minimum, the mean, the maximum, the average and the geometric mean) about these results are given in Table 4; the results which correspond to the instances that gave the minimum, the maximum, and the mean are given in the upper part of Table 5.

As in the singly bordered case, we were expecting an increase in the cutsize. Compared to again previously reported results, 15% increase in the cutsize is acceptable. This however resulted in only 16% improvement in the balance on the number of nonzeros on the diagonal blocks. The geometric mean is small, though, indicating a few outliers of a large value. We therefore present in the lower part of the Table 5 all results where the framework obtained worse balance than the standard method. Neither of the methods obtain good balance for the `fem_filter` matrix; the difference of 0.02 on 64-way partitioning of `engine` is not significant but implies 25% better result. Upon discarding those, the framework results in about 20% improvement. Although a significant improvement is obtained in terms of balance with acceptable increase in terms of the cutsize, both methods need further effort.

3.3 The doubly bordered form

The off the shelf method to permute a square symmetric matrix into the doubly bordered block diagonal form shown in (3) is to use an algorithm for graph partitioning by vertex separators (GPVS) problem (again such as Chaco [20], MeTiS [22], or Scotch [27]). This approach however does not obtain balance in terms of number of nonzeros in the row stripes, or the number of nonzeros in the diagonal blocks, or the number of nonzeros in the border. A recent alternative [10] is to decompose a given matrix A as $A = M^T M$ (patternwise) and then to permute the matrix M into the singly bordered form (1). Again the same unconventional partitioning formulation mentioned in Section 3.1 is used to obtain balance on the number of rows per diagonal block (the balance on the number of nonzeros per diagonal block is not addressed explicitly) while minimizing the number of coupling columns.

The proposed framework uses the same decomposition $A = M^T M$ as in [10] (we have used their C4

Table 6: Results for the A_{DB} experiments. The minimum, the maximum, and the average imbalance ratios of the three methods for all pattern symmetric matrices.

	SMg		SMh		Framework	
	A_{kk}	A_{kS}	A_{kk}	A_{kS}	A_{kk}	A_{kS}
min	0.01	0.01	0.01	0.26	0.02	0.23
max	9.18	32.56	2.71	2.77	1.83	1.37
average	0.71	1.58	0.34	0.62	0.26	0.48

routine) and exploits the approach described in Section 3.1 with only different weights. Consider a row i of M not touching any coupling columns. If after the bisection it does not touch any newly formed coupling columns, then the row will correspond to $|\{j : m_{ij} \neq 0\}|^2$ many nonzeros in the corresponding block of A (as the outer product of the column i of M^T and the row i will contain that many nonzeros). If, on the other hand, a row i touches α coupling columns, then the associated outer product will cover $(|\{j : m_{ij} \neq 0\}| - \alpha)^2$ nonzeros in the corresponding diagonal block of A. Therefore we assign the following weights:

$$\begin{aligned} w(v_i, 1) &= |\{j : m_{ij} \neq 0\}|^2, \\ w(v_i, 2) &= |\{j : m_{ij} \neq 0 \text{ and column } j \text{ is not cut}\}|^2 \quad . \end{aligned}$$

Although these weights are accurate when one looks at a single vertex at a time, they will not be correct when put together. We expect that the errors made in the weights will cancel the negative effects to some extend, as the same error is made for each vertex.

We have conducted experiments with all the square matrices with a symmetric pattern given in Table 1 to compare the framework with the standard method based on the GPVS using MeTiS and with a variant of the method proposed in [10]. The variant that we use makes use of the standard hypergraph partitioning tool, rather than the customized one. In this setting, we partition the matrix M (satisfying $A = M^T M$ patternwise) rowwise as described as the standard method in Section 3.1. We use SMg and SMh to refer, respectively, to the standard methods based on graph and hypergraph models. We again compare the three methods by giving statistical indicators and detailed results.

Table 6 display the minimum, the maximum, and the average imbalance ratios obtained by the three methods on the pattern symmetric data set. As seen in this table, quite disastrous imbalance ratios for the diagonal blocks and the border block can be obtained by the SMg method. The maximum imbalance ratio obtained by the other two methods are much better but still are off the acceptable limits (especially the ratios regarding the diagonal block). Nevertheless, the average imbalance values in the framework are clearly preferable to those of SMg, and enjoys about 23% improvement over those of SMh.

To put the improvements reported in Table 6 in perspective, we provide further statistics on the pattern symmetric matrices for which at least one of the methods obtained an imbalance ratio of more than 10% on the number of nonzeros in the diagonal blocks. This time we also report the size of the border obtained by SMh and the framework by normalizing them to those of SMg. The results are shown in the first half of Table 7. Under the light of these plain results, the framework is clearly preferable to both of the standard methods if about 35% increase in the border size is acceptable. The method SMh does not look promising if one looks at the average numbers. However, the geometric mean is small, indicating the existence of a few large outliers. Upon having a closer look at the data, we had that there is a single instance (the matrix `finan512` with $K = 128$ shown in Table 8) in which SMh and the framework obtained, respectively, 14% and 8% imbalance, whereas SMg obtained 2% imbalance on the number of nonzeros on the diagonal blocks. Ignoring this instance we obtain the results shown in the lower half of Table 7. This time SMh is about 10% better than SMg in terms of the imbalance; the framework obtains reductions around 28% and 18%, respectively, for the diagonal blocks and the border block.

Some instances giving the statistical indicators are shown in Table 8. All three methods have problems with $K = 128$ partitioning of `fem_filter` and `Ga10As10H30`, where the imbalance on the number of nonzeros in the diagonal blocks are off the limits. As is seen, the size of the borders obtained by SMh and the framework are smaller than that obtained by SMg, in a single case. In an additional three instances (32, 64 and 128-way partitioning of `fem_filter`) SMh obtains better border size than SMg. If

Table 7: Results for the A_{DB} experiments. The minimum, the mean, the maximum, the average and the geometric mean of the ratio of the results obtained by the framework and SMh to the results obtained by the standard method SMg.

	SHh vs SMg			Framework vs SMg		
	$\|S\|$	A_{kk}	A_{kS}	$\|S\|$	A_{kk}	A_{kS}
min	0.82	0.10	0.07	0.88	0.10	0.03
mean	1.20	0.79	0.91	1.33	0.67	0.80
max	1.85	7.00	3.50	2.10	4.00	3.20
average	1.21	1.01	0.96	1.35	0.78	0.85
geomean	1.19	0.77	0.82	1.33	0.64	0.69
	Ignoring one outlier					
min	0.82	0.10	0.07	0.88	0.10	0.03
mean	1.19	0.78	0.91	1.32	0.67	0.79
max	1.85	2.57	3.07	2.10	1.61	3.20
average	1.20	0.90	0.91	1.35	0.72	0.82
geomean	1.19	0.74	0.80	1.33	0.62	0.67

Table 8: Results for the A_{DB} experiments. The outlier for the previous table (`finan512` with $K = 128$) and some other results that gave the statistical indicators are presented. Some matrix names are shorted: `ASIC_100ks` to `ASICks`, `Dubcova3` to `Dubc3`, `FEM_3D_thermal2` to `FEM3D`, `fem_filter` to `fem_fil`, `Ga10As10H30` to `Ga10As`, `helm2d03` to `helm03`, `mono_500Hz` to `mono5`.

		SMg			SMh			Framework		
			Imbal			Imbal			Imbal	
matrix	K	$\|S\|$	A_{kk}	A_{kS}	$\|S\|$	A_{kk}	A_{kS}	$\|S\|$	A_{kk}	A_{kS}
ASICks	128	3491	0.19	0.95	3506	0.15	0.86	3917	0.12	0.67
consph	32	16826	0.33	0.48	31081	0.48	0.46	35258	0.25	0.42
consph	128	29797	0.54	0.52	47444	1.39	0.76	55328	0.87	0.60
Dubc3	32	4010	0.11	0.32	4770	0.04	0.37	5781	0.07	0.37
engine	32	7479	0.11	0.68	9790	0.11	0.81	11363	0.12	0.54
FEM3D	128	18898	0.14	0.15	24264	0.09	0.46	25732	0.13	0.48
fem_fil	128	15119	9.18	32.56	13545	0.88	2.34	17424	0.95	0.95
finan512	64	3279	0.02	0.10	4449	0.14	0.35	4508	0.08	0.23
finan512	128	6794	0.13	0.45	5567	0.22	0.34	5962	0.10	0.30
Ga10As	128	69497	6.34	5.30	71187	2.71	1.85	81428	1.83	1.37
helm03	128	12934	0.12	0.30	13690	0.03	0.38	13946	0.08	0.52
Lin	128	35787	0.17	0.37	42585	0.18	0.35	41177	0.09	0.30
mono5	32	18113	0.30	0.42	22202	0.12	0.34	23985	0.10	0.35

the increase in the border size is within the acceptable limits, the framework obtains better partitions than the other two in terms of balance on the diagonal and border blocks.

4 Conclusion and future work

We have described a set of sparse matrix computations whose efficient parallelization need some complex partitioning objectives to be attained. We have documented three sparse matrix forms which exemplified those objectives. Those three forms are the block form, the singly bordered block diagonal form, and the doubly bordered block diagonal form. The complex partitioning objectives in these forms were to achieve balance on the number nonzeros on certain parts of the matrices, rather than row stripes. We have presented a framework to address such kind of complex partitioning objectives, and evaluated the framework within hypergraph partitioning for obtaining the three forms.

For the block form and the singly bordered block diagonal form, we have obtained satisfactory results. The proposed framework obtains better balance with an acceptable increase in the associated cost function (total volume of communication in the block form and the size of the border in the singly

bordered form). For these two case, we expect improvement with better and more adaptive weighting schemes. For the doubly bordered form, we have seen improvements over the existing graph and hypergraph partitioning based methods in terms of balance. The standard GPVS tools (that are available in MeTiS [22]) are demonstrated to be susceptible to drastic imbalances. However, the results of the framework are still not competitive with the standard GPVS tools in terms of the border size. We are therefore considering the use of the proposed framework in the recursive bisection based GPVS algorithms. A significant obstacle here is that there is no GPVS tool that can handle multi-constraints. One can try to use the multi-constraint GPES routines and use the well-known wide-to-narrow separator refinement heuristics [2]. However, here one will need a heuristic for bipartite vertex cover with multiple objectives/constraints. We plan to investigate the latter problem closely.

Acknowledgments

We are thankful to X. S. Li and I. Yamazaki of Lawrence Berkeley National Laboratory for bringing the partitioning problem for their hybrid solver to our attention and remarks and comments on the problem.

References

[1] C. J. Alpert and A. B. Kahng, *Recent directions in netlist partitioning: A survey*, Integration, the VLSI Journal, 19 (1995), pp. 1–81.

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

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

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

[5] M. Benzi and B. Uçar, *Block triangular preconditioners for M-matrices and Markov chains*, Electronic Transactions on Numerical Analysis, 26 (2007), pp. 209–227.

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

[7] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, U. Catalyurek, D. Bozdag, W. Mitchell, and J. Teresco, *Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User's Guide*, Sandia National Laboratories, Albuquerque, NM, 2007.

[8] T. N. Bui and C. Jones, *Finding good approximate vertex and edge partitions is NP-hard*, Information Processing Letters, 42 (1992), pp. 153–159.

[9] ——, *A heuristic for reducing fill-in in sparse matrix factorization*, in 6th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, USA, 1993, pp. 445–452.

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

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

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

[13] ——, *PaToH: A multilevel hypergraph partitioning tool, version 3.0*, Tech. Rep. BU-CE-9915, Computer Engineering Department, Bilkent University, 1999.

[14] Ü. V. Çatalyürek and C. Aykanat, *A hypergraph-partitioning approach for coarse-grain decomposition*, in ACM/IEEE SC2001, Denver, CO, November 2001.

[15] U. V. Çatalyürek, C. Aykanat, and B. Uçar, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, SIAM Journal on Scientific Computing, 32 (2010), pp. 656–683.

[16] 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.

[17] C. Farhat and F.-X. Roux, *An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 379–396.

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

[19] B. Hendrickson and T. G. Kolda, *Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing*, SIAM Journal on Scientific Computing, 21 (2000), pp. 2048–2072.

[20] B. Hendrickson and R. Leland, *The Chaco user's guide, version 2.0*, Sandia National Laboratories, Alburquerque, NM, 87185, 1995.

[21] B. Hendrickson and E. Rothberg, *Improving the run time and quality of nested dissection ordering*, SIAM Journal on Scientific Computing, 20 (1998), pp. 468–489.

[22] 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*, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[23] ——, *Multilevel algorithms for multi-constraint graph partitioning*, Tech. Rep. 98-019, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, May 1998.

[24] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley–Teubner, Chichester, U.K., 1990.

[25] X. S. Li, M. Shao, I. Yamazaki, and E. G. Ng, *Factorization-based sparse solvers and preconditioners*, in Journal of Physics: Conference Series, vol. 180, IOP Publishing, 2009, p. 012015.

[26] I. Moulitsas and G. Karypis, *Partitioning algorithms for simultaneously balancing iterative and direct methods*, Tech. Rep. 04-014, Department of Computer Science and Engineering, University of Minnesota, 2004.

[27] F. Pellegrini, *SCOTCH 5.1 User's Guide*, Laboratoire Bordelais de Recherche en Informatique (LaBRI), 2008.

[28] A. Pinar and B. Hendrickson, *Partitioning for complex objectives*, in Proceedings of the 15th International Parallel & Distributed Processing Symposium, IPDPS '01, Washington, DC, USA, 2001, IEEE Computer Society, p. 121 (CDROM).

[29] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, 2nd ed., 2003.

[30] L. A. Sanchis, *Multiple-way network partitioning with different cost functions*, IEEE Transactions on Computers, 42 (1993), pp. 1500–1504.

[31] K. Schloegel, G. Karypis, and V. Kumar, *Parallel multilevel algorithms for multi-constraint graph partitioning*, in Euro-Par, 2000, pp. 296–310.

[32] B. Uçar and C. Aykanat, *Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1827–1859.

[33] B. Uçar and C. Aykanat, *Partitioning sparse matrices for parallel preconditioned iterative methods*, SIAM Journal on Scientific Computing, 29 (2007), pp. 1683–1709.

[34] ——, *Revisiting hypergraph models for sparse matrix partitioning*, SIAM Review, 49 (2007), pp. 595–603.

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

[36] I. Yamazaki, X. S. Li, and E. G. Ng, *Partitioning, load balancing, and matrix ordering in a parallel hybrid solver*. Presentation at SIAM Conference on Parallel Processing for Scientific Computing (PP10), 2010.