HYPERGRAPH PARTITIONING FOR SPARSE LINEAR SYSTEMS: A CASE STUDY WITH A SIMPLE DISCONTINUOUS PDE

MASHA SOSONKINA * AND YOUSEF SAAD †

Abstract. A number of challenges need to be addressed when solving sparse linear systems in parallel. In addition to the usual difficulties in reducing communication and load imbalance, one must partition the problem carefully at the outset to achieve good convergence of the iterative process. Standard graph partitioners aim at balancing the number of unknowns and reducing communication volume, based on the nonzero pattern of the matrix. As is well-known this objective is not sufficient for realistic practical problems. This is the case for example, when solving discretized Partial Differential Equation (PDE) with discontinuous coefficients or when dealing with complex multi-physics phenomena. It is therefore desirable that the partitioner take into account information beyond the matrix adjacency graph. The present work shows how the flexibility of hypergraph partitioning can be exploited to develop heuristics for incorporating numerical information in partitioning tasks. We propose a modification of a standard hypergraph model and several weighting schemes to build partitionings which will more likely lead to good convergence of the process. In a set of numerical experiments we show comparisons with standard approaches, on simple two- and three-dimensional elliptic problems with discontinuous coefficients on rectangular meshes.

Keywords:. hypergraph ; partitioning ; iterative linear system solution ; preconditioning ; PDE ; discontinuity

1. Introduction. Although iterative linear system solution techniques require relatively little effort to parallelize, achieving good performance is not an easy task. A few of the performance issues are the way in which the matrix-vector product is performed and the quality of the preconditioner. The structure of the sparse matrix governs the performance of the former while the quality of the preconditioner is heavily based on the numerical properties of the matrix, which affect the convergence behavior of the iterative method. Both factors depend on how the sparse matrix is partitioned.

The first step in representing the pattern of a matrix with a nonsymmetric pattern is often to symmetrize it, so that to obtain an undirected graph as follows. Each equation and corresponding unknown is represented by a vertex and each nonzero entry is represented by an edge between vertices coupled by an equation. This "symmetric" graph representation is widely used by a variety of graph partitioners, such as Chaco [15] and MeTiS [16]. By attempting to minimize edge cuts, these algorithms may reduce the length of the boundary part but not necessarily the number of communications, as shown, e.g., in [13]. This could be detrimental to the parallel overhead since establishing an extra communication is typically more expensive than exchanging more data in the same data exchange.

Bipartite graph [13] and hypergraph [6] models have become common tools to partition general sparse matrices. These models can represent non-symmetric matrices and can produce non-symmetric partitions, e.g., a different partition for the rows (equations) and columns (unknowns). Traditionally, the corresponding partitioning techniques target minimizing communication overhead while maintaining load balance during parallel execution of communication-intensive tasks, such as matrix-vector multiplies, which has been studied in [6, 32]. The effect of the hypergraph partitioning

*masha@scl.ameslab.gov. Ames Laboratory/DOE, Iowa State University, Ames, IA 50011. This work was supported in part by Iowa State University under the contract DE-AC02-07CH11358 with the U.S. Department of Energy and by NERSC.

†saad@cs.umn.edu. Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455. Work supported by DOE under grant DOE – DE-FG-08ER25841 and by the Minnesota Supercomputer Institute.

on the convergence of iterative methods, however, is more difficult to understand since this convergence is determined by the numerical properties of the preconditioned matrix, which in turn depends in a very complex way on the partitioning. Even a simple parallel preconditioning, Additive Schwarz without overlap, which incurs no extra communications, may require well-conditioned local linear systems in order for the preconditioned matrix to be well-behaved.

Although communication costs and load balancing are still important when solving linear systems, the overall performance, as measured by the actual time spent to solve the system in parallel, may be more severely hampered by a poor quality preconditioner than by ineffective communication in the matrix-vector product or in the preconditioner. It is possible, for example, to lose convergence completely. Hence, it is desirable to partition a problem so as to obtain a good quality preconditioning This, however, is not an easy task in general.

To the best of our knowledge, only Duff *et al.* [11] use hypergraph partitioning for the purpose of building effective parallel preconditioners. They first reorder the matrix to increase the weight of the diagonal and scale the matrix. Then the following three steps are applied: (1) sparsify the input matrix by dropping nonzeros of magnitude smaller than a tolerance value; (2) apply hypergraph partitioning to the sparsified matrix; (3) construct a preconditioner corresponding to the diagonal blocks resulting from the hypergraph partitioning. These steps are repeated for a range of tolerance values, and at the end, the partition which maximizes the relative Frobenius norm of the preconditioner is used to partition the matrix and hence to build the parallel preconditioner. In a parallel computing environment, the total communication volume during the matrix-vector multiplications may be large, since the hypergraph partitioning does not have any control on the possible communication due to the dropped nonzeros.

Another way to target effective parallel preconditioning might be to add an additional constraint to a hypergraph partitioning algorithm. However, as Pinar and Hendrickson point out in [22], partitioning with the goal of optimizing some complex objective functions cannot, in general, be done by a single partitioning. Most of the time, the complex objective cannot be evaluated before partitioning is performed. They suggest partitioning first for a simple objective function and then trying to optimize the other(s) in a distinct phase. In the case of partitioning with the goal of optimizing the preconditioner quality, combining two constraints, structural and numerical, may render the partitions sub-optimal, such that neither constraint is properly satisfied.

Linear systems arising from many engineering and scientific applications are becoming ever harder to solve by iterative methods, and the difficulty is exacerbated in a parallel computing environment. This strongly suggests that one should attempt to exploit whatever information is available to aid in the constructing the preconditioner. So far, iterative methods have been classified into special purpose techniques (e.g., multigrid) and general-purpose techniques (e.g. preconditioned Krylov methods). Special purpose methods utilize information from the physical problem. For example, one can view multigrid as a method for solving a PDE defined on a mesh instead of a method for solving a sparse linear system. In contrast, general purpose methods rely only on the data consisting of the matrix and the right-hand side. In essence, these techniques attempt, without always succeeding, to mimic the black-box nature of direct solvers. As has been generally observed, general purpose solvers encounter more difficulties in a parallel environment than in a sequential environment.

In this paper, we take the viewpoint that *a possible remedy to overcome the difficulty is to exploit information on the problem in the process of partitioning.*

Note that partitionings which exploit information on the physical geometry or equations, have already been used. For example, the authors of [21], observe that a partitioning which exploits knowledge of the problem can be vastly superior to one obtained by the general MeTiS. While it is common to partition the domain "by-hand" or use some knowledge of the geometry, it is harder to exploit knowledge about the coefficients of the PDE for partitioning. In [26] one such technique was discussed.

One remaining major difficulty is that it is hard to analyze the effect a given partitioning strategy has on the number of iterations. It is for this reason that we consider a simple case based on a Poisson equation with discontinuities on a regular grid. A question arises what information might help obtain a more effective partitioning. In the case under consideration, we assume that the problem arises from the discretization of an elliptic PDE, where the coefficients of the PDE are discontinuous. Specifically, this means that the domain Ω in which the problem is set contains subregions where the coefficients are smooth – but the coefficients have jumps between these subregions. While it might be possible to automatically detect these boundaries (e.g., by monitoring changes in matrix entries) in simple cases, this is not always easy and the information obtained in this way may be unreliable. On the other hand this information is available at the time of the discretization and it is easy to provide it. For example, these interface points may be tagged at the time the geometry is set-up and the tags can be propagated down during the discretization so that, at the end, each coordinate which corresponds to an interface point is tagged.

2. Problems with discontinuities and solution methods. Elliptic equations with discontinuous coefficients have received a great deal of attention in the past. A prototypical equation of this type, formulated in 2-dimensional space, is of the form:

$$\begin{cases} -\nabla \cdot (\alpha(x,y)\nabla u) + \sigma(x,y)u = f(x,y) & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{2.1}$$

where Ω is a rectangle and the coefficient $\alpha(x,y)$ is discontinuous. This is just a diffusion equation which can, for example, model heat transfer in an inhomogeneous medium. The important case when the coefficients $\alpha(x,y)$ and $\sigma(x,y)$ are piecewise constant is known as the Neutron-diffusion equation, a more complex version of which is the well-known group diffusion equation [3]. One way to handle the group-diffusion equation, which is really an eigenvalue problem, is to solve a succession of equations like (2.1). For the ease of exposition, assume that $\alpha(x,y)$ is piecewise constant, i.e.,

$$\alpha(x,y) = \alpha_k \quad (x,y) \in \Omega_k, \tag{2.2}$$

where $\Omega_k \subset \Omega$, $\Omega_k \cap \Omega_\ell = \emptyset$, and $\bar{\Omega} = \cup_{k=1}^{N} \bar{\Omega}_k$. An example borrowed from the paper [33] is shown in Fig. 2.1, where the domain is $\Omega = (0,\ 1)^2$, $\sigma = 0$ and $\alpha(x,y)$ is a discontinuous function having the value 1,000 inside the star-shaped domain and 1 on the rest. Such problems have been studied extensively in the multigrid (MG) literature because they cause difficulties to MG and Algebraic MG (AMG) techniques and require a special treatment (see, for example [23, 33, 1]). An AMG coesening is typically derived based on the notion of strong/weak coupling, which essentially measures the relative size of the off-diagonal entries while, in [33], the location of the internal boundary is considered explicitly.

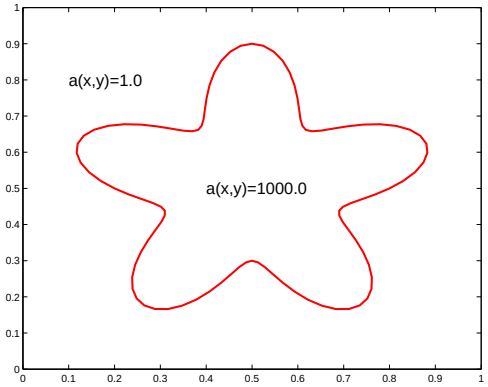


FIG. 2.1. *An elliptic problem with a star-shaped inner boundary (Example used in [33])*

Once discretized, the PDE (2.1) gives rise to a linear system of equations

$$Ax = b, \tag{2.3}$$

where A is a general sparse $n \times n$ matrix, b is the right-hand side vector, and x is the solution vector. One can solve the linear system by a preconditioned Krylov subspace method, see, e.g., [24] for details. The additive Schwarz method shown in Algorithm 2.1 provides a simple and attractive parallel preconditioningoption. As is well known, however, its convergence does not scale well, but the situation can be remedied to some extent by allowing overlap [28] whereby each subdomain k includes a layer of variables also owned by neighboring subdomains. The bigger the overlap, the better the convergence in general, though this leads to a more costly preconditioning operation. This preconditioner requires a solve for the local system which couples the local unknowns only. The Restrictive Additive Schwarz (RAS) [5] is a better approach in which, after the local solve, any overlapped variable is simply ignored, so the local subdomain keeps only its own part and discards the rest.

ALGORITHM 2.1. *Additive Schwarz in subdomain k*
Update local residual $r_k = (b - Ax)_k$
Solve $A_k \delta_k = r_k$
Update local solution $x_k = x_k + \delta_k$

More sophisticated preconditionings may be considered. In particular, distributed Schur Complement (dSC) techniques [25] often lead to superior convergence properties while retaining good parallelism. In brief, the dSC techniques are more effective than their standard Additive Schwarz counterparts because they deal with a smaller global system which only couples local and remote unknowns from neighboring subdomains. This solve is preceded by a communication phase, which is the same as the one used in the matrix-vector multiply to exchange values for the boundary unknowns. For details, see [25, 27].

3. Use of standard hypergraph partitioning model. The column-net (or row-net) hypergraph model proposed in [6] is often used to represent sparse matrices. In the column-net model of an $n \times n$ matrix A, the hypergraph $G = (V, H)$ consists of a vertex set $V = \{v_1, \ldots, v_n\}$ and the set of hyperedges, $H = \{h_1, \ldots, h_n\}$ in which each h_j is a subset of vertices. The vertex $v_i \in V$ corresponds to the ith row of the

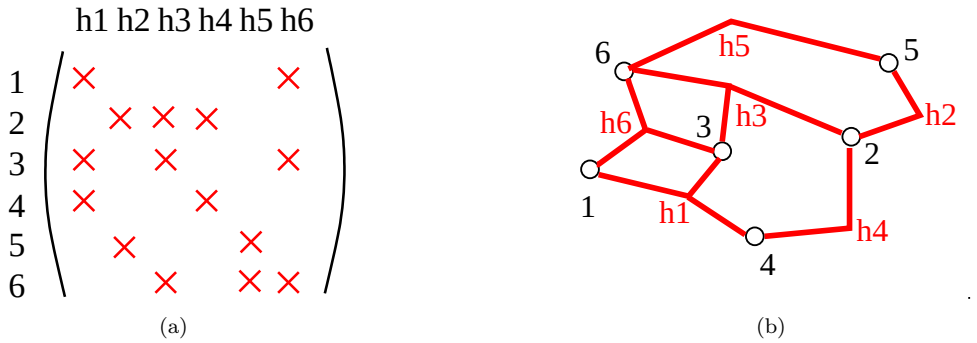


FIG. 3.1. *(a) Nonzero structure of a sample matrix, (b) Associated column-net hypergraph model*

matrix A. The hyperedge $h_j \in H$ corresponds to the jth column of A and it is equal to the set of rows i, such that a_{ij} is nonzero, i.e., $h_j = \{v_i \mid a_{ij} \neq 0\}$.

Fig. 3.1(a) shows the structure of a sample matrix, and Fig. 3.1(b) shows the associated column-net hypergraph model. In Fig. 3.1(b), the vertices are shown with circles, and the hyperedges are shown as lines connecting the vertices. In the column-net hypergraph model, weights can be associated with vertices and hyperedges. We use w_i and c_j to denote the weights of vertex v_i and hyperedge h_j, respectively.

3.1. Hypergraph partitioning. A P-way partition of a vertex set V is a set $\Pi = \{V_1, \ldots, V_P\}$ of subsets V_k of V, which are all nonempty, mutually disjoint, i.e., $V_k \cap V_\ell = \emptyset$, for $1 \leq k < \ell \leq P$; and collectively exhaustive, i.e., $V = \cup_{k=1}^{P} V_k$. Given this partitioning, the *connectivity set* Λ_j of a hyperedge h_j is the set of parts in which h_j has vertices. The *connectivity* λ_j of h_j is the cardinality of Λ_j, i.e., $\lambda_j = |\Lambda_j|$. A hyperedge h_j is said to be cut if $\lambda_j > 1$. For example, if in Fig. 3.1(b) $V_1 = \{1, 3, 4\}$ and $V_2 = \{2, 5, 6\}$, then the hyperedges h_3, h_4, and h_6 have connectivity two, and hence they are cut hyperedges, while the hyperedges h_1, h_2, and h_5 have connectivity one.

In the hypergraph partitioning problem, the objective is to minimize the *cutsize*, which is a measure of the hyperedge cuts. Two cutsize definitions are often used:

$$C^{\lambda}(\Pi) = \sum_{j \in H} c_j(\lambda_j - 1) \,, \tag{3.1}$$

$$C^{c}(\Pi) = \sum_{j \in H, \lambda_j > 1} c_j \,, \tag{3.2}$$

where c_j is the weight assigned to hyperedge h_j. These two objective functions are widely used in the VLSI community [18] and also in the scientific computing community (see, for example, [2, 4, 6, 29, 30, 31, 32]). The partitioning constraint is to satisfy a balancing requirement on the part weights, where the weight of a part is the sum of the weights of vertices in that part. A variant of this problem is the multi-constraint hypergraph partitioning [8, 17] in which each vertex has a vector of weights associated with it. The objective is the same as in (3.1), and the partitioning constraint is to satisfy a balancing constraint associated with each weight. It is well known that for the column-net hypergraph models of a matrix A, the partitioning objective function $C^{\lambda}(\Pi)$ corresponds exactly to the total communication volume (see [6, 12, 13, 14]) in the matrix-vector multiplies with A.

Section 4 presents a modification of the commonly-used hypergraph model for the linear systems arising from the discretizations of PDEs with discontinuities in specified domain regions. Hyperedge weight schemes, both for the standard and modified hypergraph models, are presented for treating boundary interfaces. The goal of these schemes is to favor certain types of hyperedges during partitioning. For the sake of exposition, the proposed techniques will be described using the column-net model only but row-net hypergraph models can be treated similarly.

3.2. Weight schemes for problems with discontinuities. Let us distinguish two types of columns: *Interior* (denoted as M^I) columns which couple only the unknowns corresponding to the mesh points *inside* a subdomain Ω_k (discontinuity region), and *Interface* or *Jump* columns (denoted as M^J) which couple the unknowns corresponding to the points lying across (or on) the interface boundary.

With this distinction of the columns, we may construct hyperedge sets H^I and H^J, which pertain to Interior and Jump columns, respectively. For example, in a standard column-net model, the columns sets M^I and M^J are essentially the same as the hyperedge sets H^I and H^J (cf. the definitions in the beginning of Section 3). However, we may use alternative hypergraph representations to define H^I and H^J differently. In particular, we consider in Section 4 a different definition of H^J.

Weights for H^I and H^J sets. To mark internal interfaces in a hypergraph representation, weighting schemes may be exploited which give different weights for hyperedges in H^I and H^J, such that hyperedges in H^J are assigned the smallest (say, a unit) weight to facilitate their splitting. The weights on $h_j \in H^I$ may be defined using the diagonal matrix values in the jth column.

Let γ_j be the scaled cardinality of h_j, which is the number of nodes in h_j divided by the average cardinality $\bar{h}$ over all hyperedges h_j:

$$\gamma_j = |h_j|/\bar{h} \ . \tag{3.3}$$

Then we may define the hyperedge weight c_j as

$$c_j = \begin{cases} 1 + \lceil |a_{jj}| \times \gamma_j \rceil & \text{if } h_j \in H^I, \\ 1 & \text{if } h_j \in H^J. \end{cases} \tag{3.4}$$

The inclusion of γ_j introduces a "size" factor into the weight equation (3.4). By putting more weight on hyperedges with more nodes, it reduces their likelihood of being split.

For a finite-difference discretization applied to (2.1), the coefficients a_{jj} on the main diagonal of matrix A in (2.3) are equal to the sum of the stencil coefficients (except for coordinates associated with domain boundary points). Consider a two-subdomain example ($\bar{\Omega}^+ \cup \Omega^- = \Omega$) and assume a positive jump in α from Ω^- to Ω^+. Then a_{jj} is greater than $a_{\ell\ell}$ for the unknowns j and ℓ corresponding to Ω^+ and Ω^-, respectively. Hence, the hyperedges associated with Ω^- will have smaller weights than hyperedges from Ω^+. (Note that only integer hyperedge weights are allowed in the current hypergraph partitioning software.)

Subdomain connectivity. In parallel with the definition of the connectivity set Λ_j (Section 3.1), let us introduce the *internal connectivity set* K_j consisting of those subdomains Ω_k that contain at least one vertex of the hyperedge h_j. Also, we denote by κ_j the cardinality of K_j. Then, to take into account the internal boundaries, the hypergraph partitioning objective function (3.1) may be rewritten as

$$C^{\lambda}(\Pi) = \sum_{j \in H} \frac{c_j}{\kappa_j}(\lambda_j - 1) \ . \tag{3.5}$$

Note that, for the hyperedges from H^I, κ_j equals one and equations (3.5) and (3.1) are the same. Furthermore, those hyperedges that have larger internal connectivity are more prone to being split, which supports our objective of cutting interface hyperedges. Thus, we may distinguish the interface hyperedges as those hyperedges with the internal connectivity κ greater than one. Let assume that the hyperedges that contain only vertices lying on the subdomain boundaries have $\kappa = 1$. An example in Fig. 4.1 shows two hyperedges, $h_i \in H^I$ and $h_j \in H^J$, with $\kappa_i = 1$ and $\kappa_j = 2$, respectively.

With the proposed weight schemes, which distinguish the "jump" hyperedges in their entirety, it may be difficult to control the precise location where a hyperedge from H^J is cut. For example, the jump hyperedge may actually be cut across two interface nodes. This appears detrimental to the iterative process and is contrary to our partitioning focus of keeping together strongly coupled nodes in an effort to obtain good convergence. Fortunately, hypergraphs provide a useful vertex aggregation mechanism since hyperedges, by definition, may contain an arbitrary number of vertices. Such a flexibility is quite helpful in this context and leads to a new hypergraph representation described next, firstly as an idea outline, secondly as a pseudo-code.

4. Hyperedge representation of internal boundary. To facilitate the precise splitting of interface hyperedges while preserving the internal-interface boundaries, we will re-construct the interface hyperedge set H^J, such that the boundary locations of a PDE with discontinuities are evidenced from the new hyperedge structure, which may be assigned distinct weights.

First, we order all the subdomains Ω_k according to decreasing absolute values of the coefficient α_k related to the domain, see equation (2.2). For simplicity we assume that ties are broken arbitrarily, although elaborate tie-breaking heuristics may help improve the balancing requirement of the resulting hypergraph representation. In addition, a hyperedge will be indexed by the same index j as one of its vertices which we will call the *principal vertex*. For example, in the column-net model for PDEs, h_j will represent column j and the principal vertex is v_j. Then, for each Ω_k in the sequence $(\Omega_1, \ldots, \Omega_N)$, we start from an interface hyperedge $h_j \in H^J$ to generate new hyperedges if the following conditions hold: h_j has not been considered already from a preceding subdomain; the connectivity set K_j contains Ω_k; and the principal vertex $v_j \notin \Omega_k$. Specifically, from such a hyperedge $h_j \in H^J$:

1. A new set of hyperedges is created, such that each new hyperedge contains only one (unique) pair of vertices (v_j, v_{j^-}), where $v_{j^-} \in h_j$ and $v_{j^-} \notin \Omega_k$. The smallest constant weight is assigned to each new hyperedge, which we will also call a *weak hyperedge*.
2. One hyperedge $\tilde{h}_j$ is created containing v_j and *all* those $v_{j^+} \in h_j$ that satisfy $v_{j^+} \in \Omega_k$. A large weight

$$\tilde{c}_j = 1 + \left\lceil |a_{jj}| \times \tilde{\gamma}_j \right\rceil$$

is assigned to $\tilde{h}_j$, such that the scaling factor $\tilde{\gamma}_j$ is defined for the hyperedge $\tilde{h}_j$ as in (3.3). We will call $\tilde{h}_j$ a *strong hyperedge*.

In the end, H^J is modified and augmented to mark possible splittings within certain columns of M^J. The H^J modification and weight assignment is implemented in the Split Interface Hyperedges (`SIH`ω) procedure shown in Algorithm 4.1, an explanation of which follows.

Let each $h_j \in H^J$ be attributed a sequence of tags marking the intersection of this hyperedge with each subdomain Ω_k. Specifically, for each $h_j \in H^J$, we introduce

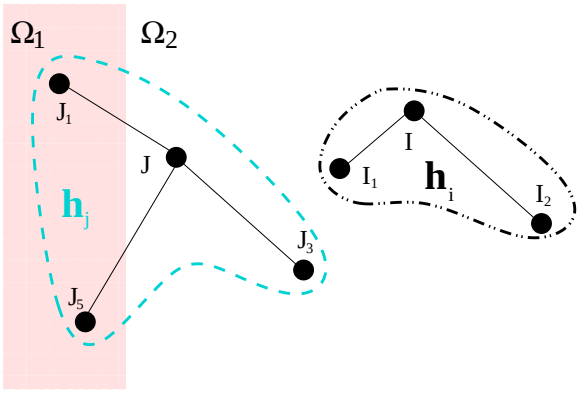


FIG. 4.1. *Examples of internal connectivity for hyperedges. (The vertices are denoted only by their indices omitting the symbol v. Interface and internal hyperedges depicted as dashed and dash-dotted closed splines, respectively.)*

a sequence t_j of N integers, called tags of the hyperedge h_j, such that, in accordance with the ordering of internal subdomains, $t_j = (\tau_j^1, \ldots, \tau_j^N)$, where

$$\tau_j^k = \begin{cases} 1 & \text{if } v_j \in \Omega_k \ ; \\ 0 & \text{if } \Omega_k \notin K_j \ ; \\ -1 & \text{if } v_j \notin \Omega_k \ . \end{cases} \tag{4.1}$$

Then, T^J is defined as a set of all such sequences. Note that the cardinality of T^J is equal to that of H^J.

SIHω considers the first nonzero tag $\tau_j^{k'}$ (line 03) from the current t_j and decides (in line 04) to modify h_j if $\tau_j^{k'}$ equals -1, i.e., if $v_j \notin \Omega_{k'}$. Then, for each vertex v_i in the hyperedge h_j, the tag sequence t_i of its corresponding hyperedge h_i is examined (line 07). A two-node (weak) hyperedge is created (line 08) only if v_i is *not* in $\Omega_{k'}$. For this to happen, three possibilities exist (1) h_i is not in H^J, i.e., $t_i \notin T^J$, (2) $\Omega_{k'}$ is not in K_i, i.e., ${\tau_i}^{k'} = 0$, and (3) v_i does not belong to $\Omega_{k'}$, i.e., ${\tau_i}^{k'} \neq 1$. These three choices are illustrated in Fig. 4.2(a). Otherwise, v_i is added to the strong hyperedge $\tilde{h}_j$ (line 10). A large weight is assigned (line 13) to the strong hyperedge (which may be a simple copy of h_j (line 12) if $\tau_j^{k'}$ equals 1). Note that the indices of the weak hyperedges start *after* the largest index of the of the strong hyperedges (line 01). Thus, the total number of the hyperedges in the modified set H^J is equal to the largest obtained index of the weak hyperedges (line 15).

ALGORITHM 4.1. *Split Interface Hyperedges with weights (*SIHω*)*
Input: *Interface hyperedges H^J and tags T^J; Matrix diagonal entries $\{a_{ii}\}_{i=1,\ldots,n}$.*
Output: *Interface hyperedges H^J (modified); Interface hyperedge weights C^J.*
01. $\ell = |H^J|$ `# Numbering of weak hyperedges.`
02. **Foreach** *hyperedge* $h_j \in H^J$ **Do:**
03. *Find in* t_j *the first* k' *such that* $\tau_j^{k'} \neq 0$
04. **If** $(\tau_j^{k'} == -1)$ **Then:**
05. $\tilde{h}_j = \{v_j\}$
06. **Foreach** $v_i \in h_j$ **Do:**
07. **If** $(t_i \notin T^J$ *or* ${\tau_i}^{k'} < 1)$ **Then:**

08. $\ell = \ell + 1; \quad \tilde{h}_\ell = \{v_j, v_i\}; \quad \tilde{c}_\ell = 1$
09. **Else:**
10. $\tilde{h}_j = \tilde{h}_j \cup \{v_i\}$
11. **Else:**
12. $\tilde{h}_j = h_j$
13. $\tilde{\gamma}_j = |\tilde{h}_j|/\bar{h}\ ; \quad \tilde{c}_j = 1 + \lceil |a_{jj}| \times \tilde{\gamma}_j \rceil$
14. **EndDo**
15. $H^J = \{\tilde{h}_1, \ldots, \tilde{h}_\ell\}\ ;\ C^J = \{\tilde{c}_1, \ldots, \tilde{c}_\ell\}$

Fig. 4.2(b) illustrates the SIHω algorithm: An interface hyperedge h_j has the internal connectivity set $K_j = \{\Omega_1, \Omega_2\}$ of cardinality $\kappa_j = 2$. If there are only two discontinuity regions in Ω, then the corresponding interface tag sequence t_j has two elements: $t_j = (\tau_j^1, \tau_j^2)$, where $\tau_j^1 = -1$ and $\tau_j^2 = 1$. Let us assume that the hyperedges with the principal vertices j_1 and j_5 are not in H^J, i.e., they have no tag sequences. Vertices j_2, j_3, and j_4 lie across the interface and the tag sequences of the respective hyperedges have $\tau_{j_2}^1 = \tau_{j_3}^1 = \tau_{j_4}^1 = 1$. Thus, for the original h_j, two weak hyperedges, $\tilde{h}_{\ell 1}$ and $\tilde{h}_{\ell 2}$, and the strong hyperedge $\tilde{h}_j$ are constructed, such that

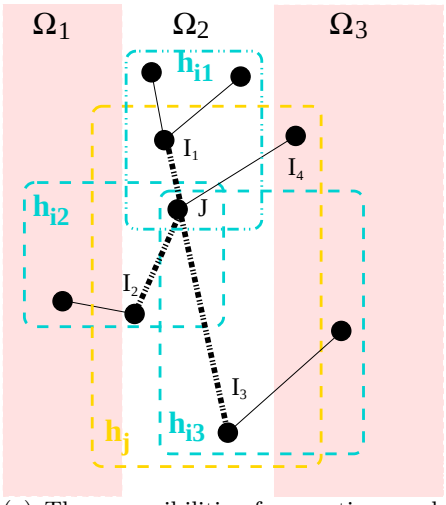
$$\begin{array}{lcllcl} \tilde{h}_{\ell 1} & = & \{j, j_1\}; & \tilde{h}_{\ell 2} & = & \{j, j_5\}; \\ \tilde{h}_j & = & \{j, j_2, j_3, j_4\}. & & & \end{array}$$

The weights are assigned as $\tilde{c}_{\ell 1} = \tilde{c}_{\ell 2} = 1$ on $\tilde{h}_{\ell 1}$ and $\tilde{h}_{\ell 2}$ and as $\tilde{c}_j$, calculated in line 13 of Algorithm 4.1, on $\tilde{h}_j$.

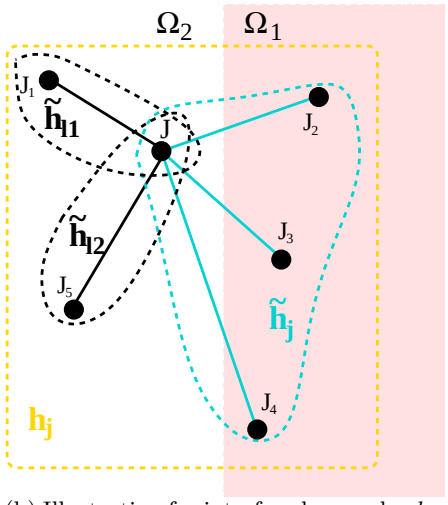
In summary, the objective of the new hypergraph representation is two-fold (1) to keep hyperedges in H^I together; (2) to avoid splitting interface hyperedges across the strongly connected nodes. We have modified the interface hyperedge set H^J, such that now, in a strong hyperedge $h_j \in H^J$, all the discretization neighbor vertices of the principal vertex v_j are from the *same* discontinuity subdomain Ω_k. This separability of the created hyperedges allows us to weigh them by assigning large weights to the hyperedges comprising only the nodes strongly connected in the discretization scheme and by scaling the weights based on the number of such connections. A two-node structure and small weight of the weak hyperedges facilitate their cutting by the partitioner. The vertex weights remain all ones to balance the parts in terms of the number of unknowns.

SIHω has no restrictions on the nature of internal interfaces and a type of discretized PDE, except the assumptions stated in equation (2.1). The algorithm starts with a standard hypergraph model for square matrices (arising in PDE applications) and a set of tags labeling the interface hyperedges. So it is applicable to a broad class of problems with internal boundaries. It is interesting to draw parallels between SIH and coarsening techniques in multigrid. The paper [33] discusses an *interface preserving* coarsening whose essence is to *select coarse grid points so that the interfaces are aligned with all the coarse grids* to paraphrase the author. In particular, first the points on the interfaces are assigned to be coarse grid points, then a standard coarsening for the remaining points is performed. This "selective interface" coarsening was found to be mandatory for MG to work.

Discontinuity information. It is typical to describe any existing discontinuities in a PDE by stating which terms of the equations are discontinuous and by specifying the location of the discontinuity region. This information may be evidenced in the resulting matrix after the problem is discretized. For example, we may detect the



(a) Three possibilities for creating weak hyperedges (dash-dotted straight lines) for Ω_3: (1) $\{v_j, v_{i_1}\}$, where the hyperedge $h_{i_1} \in H^I$, (2) $\{v_j, v_{i_2}\}$, where $\Omega_3 \notin K_{i_2}$, and (3) $\{v_j, v_{i_3}\}$, where $v_{i_3} \notin \Omega_3$.



(b) Illustration for interface hyperedge h_j, principal vertex v_j, and two subdomains Ω_1 and Ω_2, which have been ordered according to the coefficient relation: $\alpha_1 > \alpha_2$.

FIG. 4.2. *Split Interface Hyperedge (*SIHω*) procedure. The hyperedges are shown as closed-spline (dashed) lines.*

columns with the largest diagonal entries for the grid points in the discontinuity subregion or the columns having both large and small off-diagonal entries for the points lying across the discontinuity boundary. Some differencing methods modify the number of the column (row) matrix entries for the internal boundary points [19]. Thus, in the case of regular grids and the PDE type considered here, it is possible to derive the criterion for discriminating matrix columns from the problem statement and the differencing method. In general, however, it is difficult to state this criterion based solely on the matrix information, without any knowledge of the discontinuity attributes/tags which, however, may be readily provided by the user from the mesh information. For example, a set of tags for the interface nodes may be provided to convey the discontinuity information into a standard hypergraph representation and to label the interface hyperedges accordingly. The more precise this information is, the better the resulting partitioning is likely to perform.

5. Numerical experiments. The experiments were performed on two computing platforms at NERSC[1] using the PaToH [7] hypergraph partitioner, which implements partitioning sequentially and thus has some limitations on the matrix size. To perform truly large-scale tests, a new Zoltan [9] hypergraph partitioning is already available [10] and will be used in our future work. We have used the default choices for most settings in PaToH partitioning except that the partitioning of superior qual-

[1](1) The IBM SP RS/6000, named Seaborg (now retired), is a distributed memory computer with 380 compute nodes with 16 processors per node. Each processor has a peak performance of 1.5 GFlops. The nodes having 16 GBytes of shared memory were used. (2) The IBM p575 POWER5 system, named Bassi, is a distributed memory computer with 111 compute nodes with 8 processors per node. Each processor has a peak performance of 7.6 GFlops. The nodes are configured to use 20 GBytes of shared memory.

ity `QUALITY` was asked with the cost objective function $C^{\lambda}(\Pi)$. Note that in PaToH many parameter choices are based on randomized algorithms, e.g., the default for initial partitioning parameter is a greedy algorithm starting with a randomly selected vertex. By prescribing a particular seed for the randomized algorithms, we obtain consistent partitioning performance from one execution to another for the the sake of experiment comparisons.

5.1. Test problems. Consider a specific formulation of equations given in (2.1):

$$-\frac{\partial}{\partial x}(a(x,y)\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(b(x,y)\frac{\partial u}{\partial y}) = 0 \tag{5.1}$$

on a square $\bar{\Omega} = [0,1] \times [0,1]$ domain with Dirichlet boundary conditions on $\partial\Omega$ and the interface Γ separating the domain into exterior Ω^- and interior $\Omega^+ = (0.25, 0.75) \times (0.25, 0.75)$. The discretization is a five-point centered finite-difference scheme on a 20×20 grid, excluding boundary points. Consider two cases of the variability in the coefficients a and b: (P1) only $a(x,y)$ varies, such that $a(x,y) = 100$ and $b(x,y) = 1$ on Ω^+ and $a(x,y) = b(x,y) = 1$ elsewhere (Fig. 5.1(a)); (P2) both $a(x,y)$ and $b(x,y)$ vary, such that $a(x,y) = b(x,y) = 100$ in Ω^+ and $a(x,y) = b(x,y) = 1$ elsewhere (Fig. 5.1(b)). Note that, in P1, the one-directional nature of the variability essentially induces *multiple* interface regions within Ω^+, shown as dashed lines in Fig. 5.1(a).

As a more complex test, we consider a three-dimensional version of (5.1) on a cube $\Omega = [0,1] \times [0,1] \times [0,1]$ with the same boundary conditions. The interface Γ separates Ω^- from $\Omega^+ = (0.25, 0.75) \times (0.25, 0.75) \times (0.25, 0.75)$. Now we consider the following two cases of the variability in the coefficients: (P3) only a and b vary, such that $a = b = 20$ and $c = 1$ on Ω^+ and $a = b = c = 1$ elsewhere; and (P4) all the three coefficients a, b, and c have jumps, such that $a = b = c = 20$ in Ω^+ and $a = b = c = 1$ elsewhere. The discretization is done on the 7-point stencil on $100 \times 100 \times 100$ grid.

5.2. Algorithms tested. The testing has been performed using the pARMS [20] library of accelerators and preconditioners. In particular, we used the flexible variant of the restarted GMRES(20) accelerator [24] with the accuracy of 10^{-9} and Additive Schwarz (Algorithm 2.1) preconditioner with the ARMS procedure [24] to solve the local systems. Five inner iterations of the preconditioner were performed for this linear system. A number of methods to build the hypergraph are compared as to their effect on the iterative convergence. The following notation was used.

SIH denotes the *unweighted* hypergraph construction method: The interior (H^I) hyperedges correspond to the M^I matrix columns, the interface (H^J) hyperedges are created by Algorithm 4.1, and the weight set C is empty.

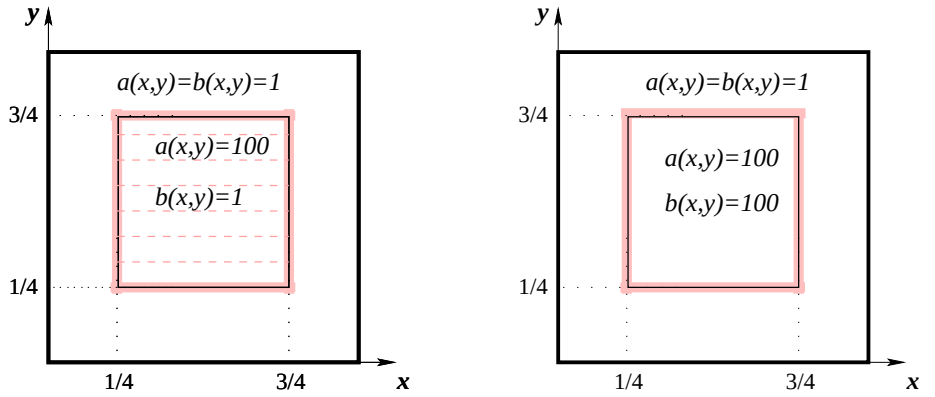
SIHω denotes the procedure in Algorithm 4.1: The interior hyperedges are assigned weights as in (3.4) and the interface ones have weights as shown in Algorithm 4.1.

St refers to the *unweighted* standard column-net model as described in Section 3.

Stω refers to `St` with weights assigned as in equation (3.4).

B1 refers to a partitioning performed by manually assigning vertices to the processors by exploiting the knowledge of internal boundary locations within Ω, such that interface hyperedges may be split only across the weak connections of points in the underlying discretization. In addition, in the subdomain Ω^+, we cut only the weak connections of descretization points as provided by coefficients b of problem P1.

B2 refers to `B1` but with an arbitrary hyperedge splitting inside Ω^+.



(a) Problem P1: jump in a coefficient only (b) Problem P2: jump in a and b coefficients

FIG. 5.1. *2-D test problem with discontinuity region Ω^+. The points corresponding to the matrix "jump" column set M^J are depicted as solid light-shaded thick lines, while the dashed lines in (a) go along the direction of the large coefficient.*

5.3. Results and discussion.

2-D Problem. Fig. 5.2–5.5 depict the outcomes of the hypergraph partitioning methods as denoted in Section 5.2 Tables 5.1(top) and 5.1(bottom) show the outer iteration numbers (column `It`) and solution time (column `T`) in seconds for 8- and 16-processor executions on Seaborg, respectively. Columns `rCut` present the number of hyperedge cuts relative to the total number of hyperedges for each construction method (Column `Alg`). It is remarkable that the partitioning of the hypergraph built by the `SIH`ω procedure leads to a convergence similar to the "ideal" one, i.e, corresponding to the manual partitioning `B1` shown in Fig. 5.5(a). Furthermore, if the discontinuity region is cut across (as in `B2`, Fig. 5.5(b)), the convergence deteriorates sharply for the problem P1. `SIH`ω has also the smallest `rCut` among weighted algorithms on P1.

3-D problem. This problem is more difficult to solve. Hence, to show the comparative performance within a reasonable computation time, we have decreased the values of the coefficient in Ω^+ from 100 to 20, increased the maximum number of iterations to 1000 while applying the preconditioner without inner iterations. Fig. 5.6–5.7 show (a) the number of outer iterations to convergence normalized by the number of the iterations when `SIH`ω is used, and (b) the total time spent to reach convergence for a series of processor numbers shown in the x-axes. For P3, the variant `SIH`ω still performs the best: All the bars are above 1 in Fig. 5.6(a). The gains of `SIH`ω on P3 are quite pronounced always leading to fast convergence. For P4, the weighted variant `St`ω appears competitive with `SIH`ω on some processor numbers. As in the case of the "toy" 2D problem, this is not unexpected since, to produce balanced partitions, all the weighted methods cut across discontinuities and result in partitions of similar quality. On the other hand, the absence of weights (as in `SIH` and `St`) often results in a poor convergence and solution time.

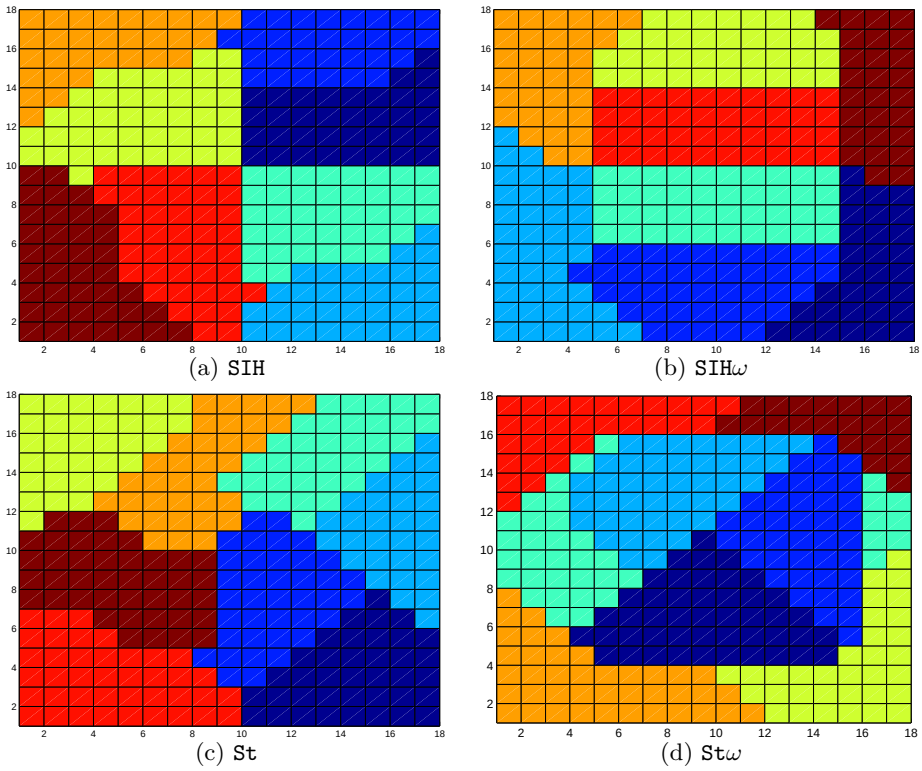
6. Conclusions. Hypergraphs were initially advocated mainly as a tool to better optimize communications and memory usage when solving non-symmetric linear systems on parallel platforms. We have shown that, thanks to their flexibility, hypergraphs provide a powerful environment to build partitionings of good intrinsic quality. We have proposed a heuristic to construct a hypergraph guided by the knowledge of

TABLE 5.1

Convergence and hyperedge cut information for 2-D problem (5.1): (top) 8-processor and (bottom) 16-processor execution.

	P1			P2		
Alg	It	T	rCut	It	T	rCut
SIH	200	.13	.26	98	.06	.36
SIHω	33	.03	.27	59	.04	.39
St	200	.12	.39	83	.06	.39
Stω	200	.13	.48	81	.06	.44
B1	29	.02	.39	36	.04	.39
B2	198	.12	.39	46	.03	.39

	P1			P2		
Alg	It	T	rCut	It	T	rCut
SIH	200	.13	.39	120	.08	.58
SIHω	39	.03	.46	86	.06	.60
St	200	.13	.61	106	.07	.61
Stω	200	.14	.71	116	.08	.70



(a) SIH (b) SIHω

(c) St (d) Stω

FIG. 5.2. *8-way partitioning for problem P1 using standard column-net model and its modification* SIH *as in Algorithm 4.1 with and without weights*

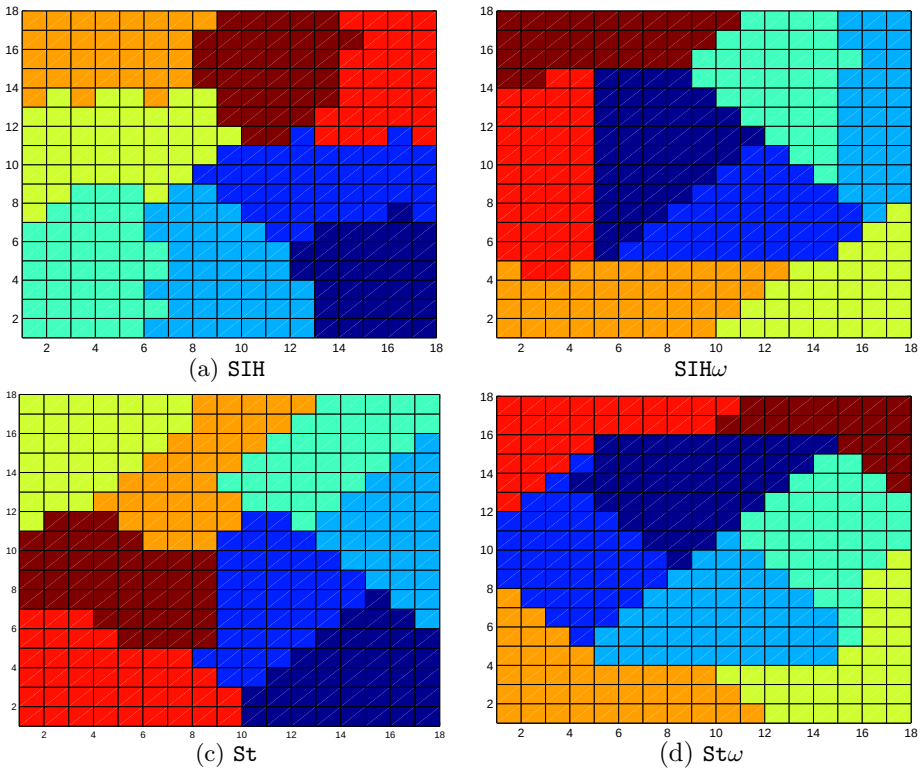


FIG. 5.3. *8-way partitioning for problem P2*

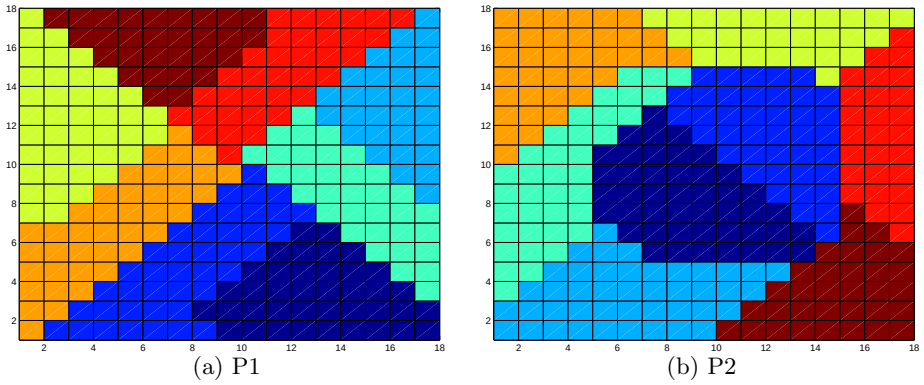
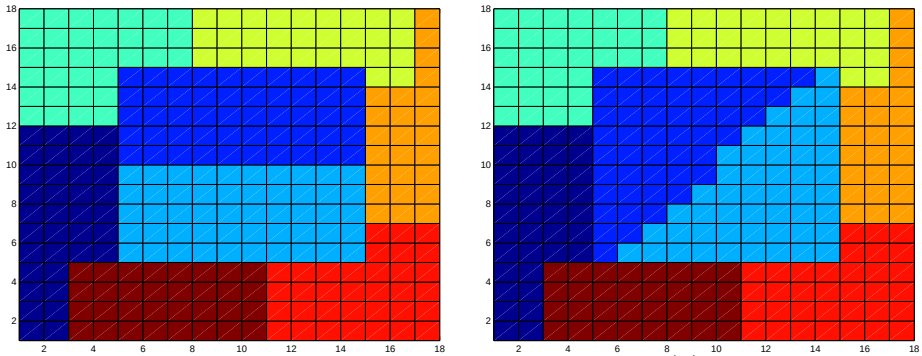


FIG. 5.4. *8-way partitioning standard column-net model weights assigned as in equations (3.4)*

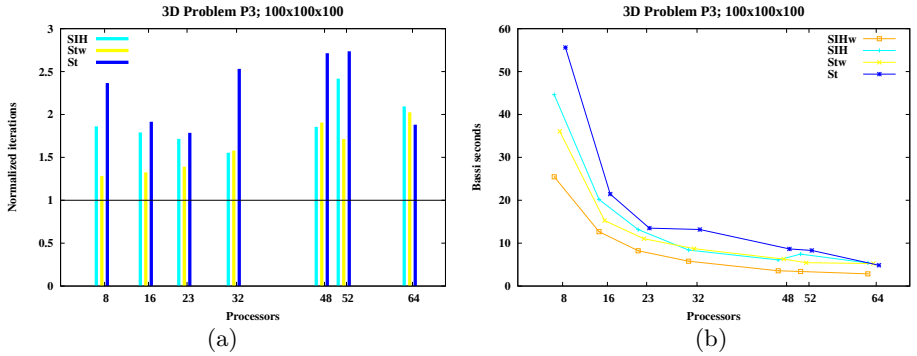
the underlying problem, such as locations of discontinuities in the physical domain and some geometric information about the discretization of the problem. Our experiments suggest that the strongly connected vertices, such as those within a discontinuity region with large function values, should be kept in the same part.

When it comes to partitioning a mesh, the experiments underscore the importance of aiming for a good quality partitioning, which should take into consideration the convergence behavior of the iterative solver perhaps more so than considerations related



(a) B1 (b) B2

FIG. 5.5. *8-way parts obtained with "By hand" partitioning*



(a) (b)

FIG. 5.6. *Performance for the 3D problem P3*

to load balancing and reduced communication cost. Indeed, straightforward applications of existing hypergraph models may lead to highly ineffective iterations. For hard problems, using matrix values for the hyperedge weight assignment in a combination with a special treatment of matrix columns (rows) that express the discontinuity may lead to good convergence whereas using unweighted hypergraph representations often leads to convergence failure.

Although we have considered only elliptic PDEs defined on regular grids, there are no restrictions on the nature of internal interfaces and on the type of PDE. The proposed methods for hypergraph construction and weight schemes are applicable to more complex problems with a broad class of internal boundaries. They require only some user-provided information about the locations of the internal interface boundaries.

REFERENCES

[1] R. E. ALCOUFFE, A. BRANDT, J. J. E. DENDY, AND J. W. PAINTER, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM Journal on Scientific and Statistical Computing, 2 (1981), pp. 430–454.

[2] C. AYKANAT, A. PINAR, AND Ü. V. ÇATALYÜREK, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM J. Sci. Comput., 25 (2004), pp. 1860–1879.

[3] G. BIRKHOFF AND R. E. LYNCH, *Numerical solution of elliptic problems*, SIAM, Philadelphia, PA, 1984.

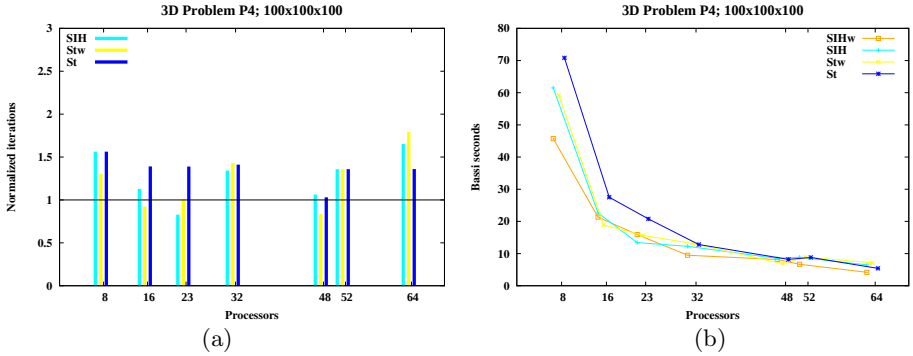


FIG. 5.7. *Performance for the 3D problem P4*

[4] R. H. BISSELING AND W. MEESEN, *Communication balancing in parallel sparse matrix-vector multiplication*, Electronic Transactions on Numerical Analysis, 21 (2005), pp. 47–65.

[5] X. C. CAI AND M. SARKIS, *A restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM Journal on Scientific Computing, 21 (1999), pp. 792–797.

[6] Ü. V. ÇATALYÜREK AND C. AYKANAT, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication.*, IEEE Trans. Parallel Distrib. Syst., 10 (1999), pp. 673–693.

[7] ———, *PaToH: A multilevel hypergraph partitioning tool, version 3.0*, tech. rep., Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey, 1999.

[8] Ü. V. ÇATALYÜREK AND C. AYKANAT, *A hypergraph-partitioning approach for coarse-grain decomposition*, in Proceedings of Scientific Computing 2001 (SC2001), Denver, Colorado, November 2001, pp. 10–16.

[9] K. DEVINE, E. BOMAN, R. HEAPBY, B. HENDRICKSON, AND C. VAUGHAN, *Zoltan data management service for parallel dynamic applications*, Computing in Science and Engg., 4 (2002), pp. 90–97.

[10] 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.

[11] I. S. DUFF, S. RIYAVONG, AND M. B. VAN GIJZEN, *Parallel preconditioners based on partitioning sparse matrices*, Tech. Rep. TR/PA/04/114, CERFACS, Toulouse, France, 2004.

[12] B. HENDRICKSON, *Graph partitioning and parallel solvers: has the emperor no clothes?*, Lect. Notes Comput. Sci., 1457 (1998), pp. 218–225.

[13] B. HENDRICKSON AND T. G. KOLDA, *Graph partitioning models for parallel computing.*, Parallel Computing, 26 (2000), pp. 1519–1534.

[14] B. HENDRICKSON AND T. G. KOLDA, *Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing*, SIAM J. Sci. Comput., 21 (2000), pp. 2048–2072.

[15] B. HENDRICKSON AND R. LELAND, *The Chaco user's guide — version*, 1994.

[16] G. KARYPIS AND V. KUMAR, *MeTiS, unstructured graph partitioning and sparse matrix ordering system. version 2.0*, tech. rep., University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, Aug. 1995.

[17] G. KARYPIS AND V. KUMAR, *Multilevel algorithms for multi-constraint hypergraph partitioning*, Tech. Rep. 99-034, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, November 1998.

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

[19] R. J. LEVEQUE AND Z. L. LI, *The immersed interface method for elliptic equations with discontinuous coefficients and singular sources*, SIAM Journal on Numerical Analysis, 31 (1994), pp. 1019–1044.

[20] Z. LI, Y. SAAD, AND M. SOSONKINA, *pARMS: A parallel version of the algebraic recursive multilevel solver*, Numerical Linear Algebra with Applications, 10 (2003), pp. 485–509.

[21] L. LITTLE, Z. LI, H. G. CHOI, AND Y. SAAD, *Particle partitioning strategies for the parallel computation of solid-liquid flows*, Computers in Math. with Applications, 43 (2002), pp. 1591–1616.

[22] A. PINAR AND B. HENDRICKSON, *Partitioning for complex objectives*, in Proceedings of the 15th

International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2001.

[23] A. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, S. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, 1987, ch. 4.

[24] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA, 2003.

[25] Y. Saad and M. Sosonkina, *Distributed Schur Complement techniques for general sparse linear systems*, SIAM J. Scientific Computing, 21 (1999), pp. 1337–1356.

[26] Y. Saad and M. Sosonkina, *Non-standard parallel solution strategies for distributed sparse linear systems*, in Parallel Computation: Proc. of ACPC'99, A. U. P. Zinterhof, M. Vajtersic, ed., Lecture Notes in Computer Science, Berlin, 1999, Springer-Verlag.

[27] Y. Saad, M. Sosonkina, and J. Zhang, *Domain decomposition and multi-level type techniques for general sparse linear systems*, in Domain Decomposition Methods 10, Providence, RI, 1998, American Mathematical Society.

[28] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, New York, 1996.

[29] B. Uçar and C. Aykanat, *Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies*, SIAM J. Sci. Comput., 25 (2004), pp. 1827–1859.

[30] ———, *Revisiting hypergraph models for sparse matrix partitioning*, SIAM Rev., (accepted for publication, 2006).

[31] ———, *Partitioning sparse matrices for parallel preconditioned iterative methods*, SIAM J. Sci. Comput., (submitted, 2004).

[32] 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.

[33] W. L. Wan, *Interface preserving coarsening multigrid for elliptic problems with highly discontinuous coefficients*, Numer. Linear Algebra Appl., 7 (2000), pp. 727–741.