Available online at www.sciencedirect.com

ScienceDirect

J. Parallel Distrib. Comput. 68 (2008) 609–625

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices[☆]

Cevdet Aykanat[a,*], B. Barla Cambazoglu[b], Bora Uçar[c,1]

[a]*Computer Engineering Department, Bilkent University, 06800 Bilkent, Ankara, Turkey*
[b]*Department of Biomedical Informatics, Ohio State University, Columbus, OH 43210, USA*
[c]*CERFACS, 42. Av. G. Coriolis, 31057 Toulouse, France*

Received 27 July 2006; received in revised form 20 September 2007; accepted 25 September 2007
Available online 7 October 2007

Abstract

K-way hypergraph partitioning has an ever-growing use in parallelization of scientific computing applications. We claim that hypergraph partitioning with multiple constraints and fixed vertices should be implemented using direct K-way refinement, instead of the widely adopted recursive bisection paradigm. Our arguments are based on the fact that recursive-bisection-based partitioning algorithms perform considerably worse when used in the multiple constraint and fixed vertex formulations. We discuss possible reasons for this performance degradation. We describe a careful implementation of a multi-level direct K-way hypergraph partitioning algorithm, which performs better than a well-known recursive-bisection-based partitioning algorithm in hypergraph partitioning with multiple constraints and fixed vertices. We also experimentally show that the proposed algorithm is effective in standard hypergraph partitioning.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Hypergraph partitioning; Multi-level paradigm; Recursive bisection; Direct K-way refinement; Multi-constraint; Fixed vertices

1. Introduction

1.1. Motivation

In the literature, combinatorial models based on hypergraph partitioning are proposed for various complex and irregular problems arising in parallel scientific computing [4,10,17,26,50,53], VLSI design [2,42], software engineering [6], and database design [22,23,41,43,46]. These models formulate an original problem as a hypergraph partitioning problem, trying to optimize a certain objective function (e.g., minimizing the total volume of communication in parallel volume rendering, optimizing the placement of circuitry on a dice area, minimizing the access to disk pages in processing GIS queries) while maintaining a constraint (e.g., balancing the computational load in a parallel system, using disk page capacities as an upper bound in data allocation) imposed by the problem. In general, the solution quality of the hypergraph partitioning problem directly relates to the formulated problem. Hence, efficient and effective hypergraph partitioning algorithms are important for many applications.

1.2. Definitions

A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$ [5]. Each net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$. The set of vertices connected by a net n_j are called its pins and denoted as $Pins(n_j)$. The size of a net n_j is equal to the number of its pins, that is, $s(n_j)=|Pins(n_j)|$. A cost $c(n_j)$ is associated with each net n_j. The nets connecting a vertex v_i are called its nets and denoted as $Nets(v_i)$. The degree of a vertex v_i is equal to the number of its nets, that is, $d(v_i)=|Nets(v_i)|$. A weight $w(v_i)$ is associated with each

[☆] This work is partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under project EEEAG-106E069.
[*] Corresponding author. Fax: +90 312 2664047.
E-mail addresses: aykanat@cs.bilkent.edu.tr (C. Aykanat), barla@bmi.osu.edu (B.B. Cambazoglu), ubora@cerfacs.fr (B. Uçar).
[1] The work of this author was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under the program 2219, by the University Research Committee of Emory University, and by Agence Nationale de la Recherche through project ANR-06-CIS6-010.

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.09.006

vertex v_i. In case of multi-constraint partitioning, multiple weights $w^1(v_i), w^2(v_i), \ldots, w^T(v_i)$ may be associated with a vertex v_i, where T is the number of constraints.

$\Pi=\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition if each part $\mathcal{V}_k$ is non-empty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. In Π, a net is said to connect a part if it has at least one pin in that part. The connectivity set Λ_j of a net n_j is the set of parts connected by n_j. The connectivity $\lambda_j=|\Lambda_j|$ of a net n_j is equal to the number of parts connected by n_j. If $\lambda_j=1$, then n_j is an internal net. If $\lambda_j>1$, then n_j is an external net and is said to be cut.

The K-way hypergraph partitioning problem (e.g., see [2]) is defined as finding a vertex partition $\Pi=\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ for a given hypergraph $\mathcal{H}=(\mathcal{V}, \mathcal{N})$ such that a partitioning objective defined over the nets is optimized while a partitioning constraint is maintained.

In general, the partitioning objective is to minimize a cost function defined over the cut nets. Frequently used cost functions [42] include the cut-net metric

$$cutsize(\Pi)=\sum_{n_j\in\mathcal{N}_{\text{cut}}} c(n_j), \tag{1}$$

where each cut net incurs its cost to $cutsize(\Pi)$, and the connectivity-1 metric

$$cutsize(\Pi)=\sum_{n_j\in\mathcal{N}_{\text{cut}}} c(n_j)(\lambda_j-1), \tag{2}$$

where each cut net n_j incurs a cost of $c(n_j)(\lambda_j-1)$ to $cutsize(\Pi)$. In Eqs. (1) and (2), $\mathcal{N}_{\text{cut}}$ denotes the set of cut nets. In this work, we use the connectivity-1 metric.

Typically, the partitioning constraint is to maintain one or more balance constraints on the part weights. A partition Π is said to be balanced if each part $\mathcal{V}_k$ satisfies the balance criteria

$$W^t(\mathcal{V}_k)\leqslant(1+\varepsilon^t)W^t_{\text{avg}} \quad \text{for } k=1,2,\ldots,K \quad \text{and } t=1,2,\ldots,T. \tag{3}$$

In Eq. (3), for the tth constraint, each weight $W^t(\mathcal{V}_k)$ of a part $\mathcal{V}_k$ is defined as the sum of the weights $w^t(v_i)$ of the vertices in that part, W^t_{avg} is the weight that each part must have in the case of perfect balance, and ε^t is the maximum imbalance ratio allowed. In case of hypergraph partitioning with fixed vertices [1], there is an additional constraint on part assignment of some vertices, i.e., a number of vertices are assigned to parts prior to partitioning with the condition that, at the end of the partitioning, those vertices will remain in the part that they are assigned to.

1.3. Issues in hypergraph partitioning

The hypergraph partitioning problem is known to be NP-hard [42], and the algorithms used in partitioning a hypergraph are heuristics. Consequently, the partitioning algorithms must be carefully designed and implemented for increasing the quality of the optimization. At the same time, the computational overhead due to the partitioning process should be minimized in case this overhead is a part of the entire cost to be minimized (e.g., the duration of preprocessing within the total run-time of a parallel application).

The very first works (mostly in the VLSI domain) on hypergraph partitioning used the recursive bisection (RB) paradigm. In the RB paradigm, a hypergraph is recursively bisected (i.e., two-way partitioned) until the desired number of parts is obtained. At each bisection step, cut-net removal and cut-net splitting techniques [16] are adopted to optimize the cut-net and connectivity-1 metrics, respectively. Iterative improvement heuristics based on vertex moves or swaps between the parts are used to refine bisections to decrease the cutsize. The performance of iterative improvement heuristics deteriorate in partitioning hypergraphs with large net sizes [36] and small vertex degrees [29]. Moreover, those improvement heuristics do not have a global view of the problem, and hence solutions are usually far from being optimal.

The multi-level hypergraph partitioning approach emerged as a remedy to these problems [8]. In multi-level bisection, the original hypergraph is coarsened into a smaller hypergraph after a series of coarsening levels, in which highly coherent vertices are grouped into supervertices, thus decreasing the sizes of the nets. After the bisection of the coarsest hypergraph, the generated coarse hypergraphs are uncoarsened back to the original, flat hypergraph. At each uncoarsening level, a refinement heuristic (e.g., FM [28] or KL [39]) is applied to minimize the cutsize while maintaining the partitioning constraint. The multi-level partitioning approach has proven to be very successful [16,30,33,34,36] in optimizing various objective functions.

With the widespread use of hypergraph partitioning in modeling computational problems outside the VLSI domain, the RB scheme adopting the FM-based local improvement heuristics turned out to be inadequate due to the following reasons. First, in partitioning hypergraphs with large net sizes, if the partitioning objective depends on the connectivity of the nets (e.g., the connectivity-1 metric), good partitions cannot always be obtained. The possibility of finding vertex moves that will reduce the cutsize is limited, especially at the initial bisection steps, where net sizes are still large, as nets with large sizes are likely to have large numbers of pins on both parts of the bisection [36]. Second, in partitioning hypergraphs with large variation in vertex weights, targeted balance values may not always be achieved since the imbalance ratio needs to be adaptively adjusted at each bisection step. Third, the RB scheme's nature of partitioning hypergraphs into two equally weighted parts restricts the solution space. In general, imbalanced bisections have the potential to lead to better cutsizes [47]. Finally, several formulations that are variations of the standard hypergraph partitioning problem (e.g., multiple balance constraints, multi-objective functions, fixed vertices), which have recently started to find application in the literature, are not appropriate for the RB paradigm.

As stated above, the RB scheme performs rather poorly in problems where a hypergraph representing the computational structure of a problem is augmented by imposing more than one constraints on vertex weights or introducing a set of fixed vertices into the hypergraph. In the multi-constraint partitioning case, the solution space is usually restricted since multiple constraints may further restrict the movement of vertices between the parts. Equally weighted bisections have a tendency to minimize the maximum of the imbalance ratios (according to multiple weights) to enable the feasibility of the following bisections. This additional restriction has manifested itself as 20–30% degradation in cutsizes with respect to the single-constraint formulation in some recent applications [35,51,54].

In partitioning with fixed vertices, the RB-based approaches use fixed vertices to guide the partitioning at each bisection step. A set of vertices fixed to a certain subset of parts (a practical option is to select the vertices fixed to the first $K/2$ parts) are placed in the same part in the first bisection step. As there is no other evident placement information, the same kind of action is taken in the following bisection steps as well. Note that this is a restriction since any bisection that keeps the vertices fixed to half of the parts in the same side of the partition is feasible. That is, parts can be relabeled after the partitioning took place according to the fixed vertex information. In other words, there are combinatorially many part labelings that are consistent with the given fixed vertex information, and the RB-based approaches do not explore these labelings during partitioning. Combined with the aforementioned shortcomings of the RB scheme, this can have a dramatic impact on the solution quality. In Section 5.4, we report cutsize improvements up to 33.30% by a carefully chosen part labeling combined with K-way refinement.

1.4. Contributions

In this work, we propose a new multi-level hypergraph partitioning algorithm with direct K-way refinement. Based on this algorithm, we develop a hypergraph partitioning tool capable of partitioning hypergraphs with multiple constraints. Moreover, we extend the proposed algorithm and the tool in order to partition the hypergraphs with fixed vertices. The extension is to temporarily remove the fixed vertices, partition the remaining vertices, and then optimally assign the fixed vertices to the obtained parts prior to direct K-way refinement. The fixed-vertex-to-part assignment problem is formulated as an instance of the maximum-weighted bipartite graph matching problem.

We conduct experiments on a wide range of hypergraphs with different properties (i.e., number of vertices, average net sizes). The experimental results indicate that, in terms of both execution time and solution quality, the proposed algorithm performs better than the state-of-the-art RB-based algorithms provided in PaToH [16]. In the case of multiple constraints and fixed vertices, the obtained results are even superior.

The rest of the paper is organized as follows. In Section 2, we give an overview of the previously developed hypergraph partitioning tools and a number of problems that are modeled as a hypergraph partitioning problem in the literature. The proposed hypergraph partitioning algorithm is presented in Section 3. In Section 4, we present an extension to this algorithm in order to encapsulate hypergraph partitioning with fixed vertices. In Section 5, we verify the validity of the proposed work by experimenting on well-known benchmark data sets. The paper is concluded in Section 6.

2. Previous work on hypergraph partitioning

2.1. Hypergraph partitioning tools

Although hypergraph partitioning is widely used in both academia and industry, the number of publicly available tools is limited. Other than the Mondriaan partitioning tool [53], which is specialized on sparse matrix partitioning, there are five general-purpose hypergraph partitioning tools that we are aware of: hMETIS [33], PaToH [16], MLPart [9], Parkway [48], and Zoltan [24], listed in chronological order.

hMETIS [33] is the earliest hypergraph partitioning tool, published in 1998 by Karypis and Kumar. It contains algorithms for both RB-based and direct K-way partitioning. The objective functions that can be optimized using this tool are the cut-net metric and the sum of external degrees metric, which simply sums the connectivities of the cut nets. The tool has support for partitioning hypergraphs with fixed vertices.

PaToH [16] was published in 1999 by Çatalyürek and Aykanat. It is a multi-level, RB-based partitioning tool with support for multiple constraints and fixed vertices. Built-in objective functions are the cut-net and connectivity-1 cost metrics. A high number of heuristics for coarsening, initial partitioning, and refinement phases are readily available in the tool.

MLPart [9] was published in 2000 by Caldwell et al. This is an open source hypergraph partitioning tool specifically designed for circuit hypergraphs and partitioning-based placement in VLSI layout design. It has support for partitioning with fixed vertices.

Parkway [48] is the first parallel hypergraph partitioning tool, published in 2004 by Trifunovic and Knottenbelt. It is suitable for partitioning large hypergraphs in multi-processor systems. The tool supports both the cut-net and connectivity-1 cost metrics.

Also, Sandia National Labs' Zoltan toolkit [25] contains a recently developed parallel hypergraph partitioner [24]. This partitioner is based on the multi-level RB paradigm and currently supports the connectivity-1 cost metric.

2.2. Applications of hypergraph partitioning

Hypergraph partitioning has been used in VLSI design [2,20,32,42,45] since 1970s. The application of hypergraph partitioning in parallel computing starts by the work of Çatalyürek and Aykanat [15,17]. This work addresses 1D (rowwise or columnwise) partitioning of sparse matrices for efficient parallelization of matrix–vector multiplies. Later, Çatalyürek and Aykanat [18,19] and Vastenhouw and Bisseling [53] proposed hypergraph partitioning models for 2D (non-zero-based) partitioning of sparse matrices. In these models, the partitioning objective is to minimize the total volume of communication while maintaining the computational load balance. These matrix partitioning models are used in different applications that involve repeated matrix–vector multiplies, such as computation of response time densities in large Markov models [26], restoration of blurred images [52], and integer factorization in the number field sieve algorithm in cryptology [7].

In parallel computing, there are also hypergraph partitioning models that address objectives other than minimizing the total volume of communication. For example, Aykanat et al. [4] consider minimizing the border size in permuting sparse rectangular matrices into singly bordered block diagonal form for efficient parallelization of linear programming solvers, LU and QR factorizations. Another example is the communication hypergraph model proposed by Uçar and Aykanat [49] for considering message latency overhead in parallel sparse matrix–vector multiples based on 1D matrix partitioning.

Besides matrix partitioning, hypergraph models are also proposed for other parallel and distributed computing applications. These include workload partitioning in data aggregation [11], image-space-parallel direct volume rendering [10], data declustering for multi-disk databases [41,43], and scheduling file-sharing tasks in heterogeneous master–slave computing environments [37,38,40].

Formulations that extend the standard hypergraph partitioning problem (e.g., multiple vertex weights and fixed vertices) also find application. For instance, multi-constraint hypergraph partitioning is used for 2D checkerboard partitioning of sparse matrices [19] and parallelizing preconditioned iterative methods [51]. Hypergraph partitioning with fixed vertices is used in formulating the remapping problem encountered in image-space-parallel volume rendering [10].

Finally, we note that hypergraph partitioning also finds application in problems outside the parallel computing domain such as road network clustering for efficient query processing [22,23], pattern-based data clustering [44], reducing software development and maintenance costs [6], topic identification in text databases [13], and processing spatial join operations [46].

3. K-way hypergraph partitioning algorithm

The proposed algorithm follows the traditional multi-level partitioning paradigm. It includes three consecutive phases: multi-level coarsening, initial partitioning, and multi-level K-way refinement. Fig. 1 illustrates the algorithm.

3.1. Multi-level coarsening

In the coarsening phase, a given flat hypergraph $\mathcal{H}^0$ is converted into a sufficiently small hypergraph $\mathcal{H}^m$, which has vertices with high degrees and nets with small sizes, after m successive coarsening levels. At each level ℓ, an intermediate coarse hypergraph $\mathcal{H}^{\ell+1}=(\mathcal{V}^{\ell+1}, \mathcal{N}^{\ell+1})$ is generated by coarsening the finer parent hypergraph $\mathcal{H}^{\ell}=(\mathcal{V}^{\ell}, \mathcal{N}^{\ell})$. The coarsening phase results in a sequence $\mathcal{H}^1, \mathcal{H}^2, \dots, \mathcal{H}^m$ of m coarse hypergraphs.

The coarsening at each level ℓ is performed by coalescing vertices of $\mathcal{H}^\ell$ into supervertices in $\mathcal{H}^{\ell+1}$. For vertex grouping, agglomerative or matching-based heuristics may be used. In our case, we use the randomized heavy-connectivity matching heuristic [16,17]. In this heuristic, vertices in vertex set $\mathcal{V}^\ell$ are visited in a random order. In the case of unit-cost nets, every visited, unmatched vertex $v_i \in \mathcal{V}^\ell$ is matched with a currently unmatched vertex $v_j \in \mathcal{V}^\ell$ that shares the maximum number of

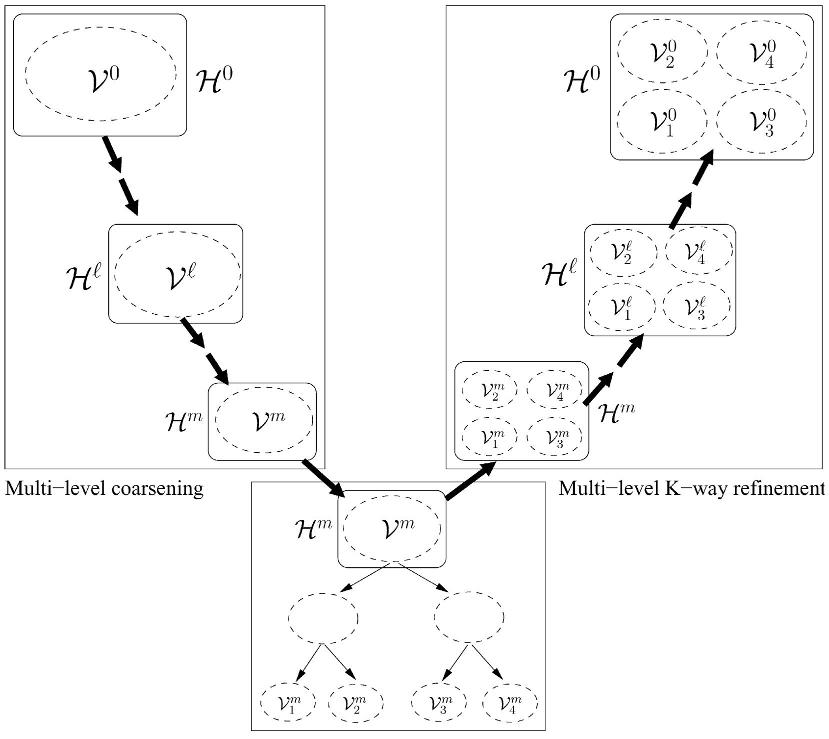
$\mathcal{H}^0$
$\mathcal{V}^0$
$\mathcal{H}^\ell$
$\mathcal{V}^\ell$
$\mathcal{H}^m$
$\mathcal{V}^m$
Multi−level coarsening
$\mathcal{V}_1^m$ $\mathcal{V}_2^m$ $\mathcal{V}_3^m$ $\mathcal{V}_4^m$
Recursive−bisection−based initial partitioning
$\mathcal{V}_1^\ell$ $\mathcal{V}_2^\ell$ $\mathcal{V}_3^\ell$ $\mathcal{V}_4^\ell$
$\mathcal{V}_1^0$ $\mathcal{V}_2^0$ $\mathcal{V}_3^0$ $\mathcal{V}_4^0$
Multi−level K−way refinement

Fig. 1. The proposed multi-level K-way hypergraph partitioning algorithm.

nets with v_i. In the case of nets with variable costs, v_i is matched with a vertex v_j such that $\sum_{n_h \in \mathcal{C}_{ij}} c(n_h)$ is the maximum over all unmatched vertices that share at least one net with v_i, where $\mathcal{C}_{ij}=\{n_h : n_h \in Nets(v_i) \wedge n_h \in Nets(v_j)\}$, i.e., $\mathcal{C}_{ij}$ denotes the set of nets that connect both v_i and v_j. Each matched vertex pair ($v_i \in \mathcal{V}^{\ell}$, $v_j \in \mathcal{V}^{\ell}$) forms a single supervertex in $\mathcal{V}^{\ell+1}$.

As the coarsening progresses, nets with identical pin sets may emerge. Such nets become redundant for the subsequent coarser hypergraphs and hence can be removed. In this work, we use an efficient algorithm for identical net detection and elimination. This algorithm is similar to the algorithm in [3], which is later used in [31] for supervariable identification in nested-dissection-based matrix reordering.

3.2. RB-based initial partitioning

The objective of the initial partitioning phase is to obtain a K-way initial partition $\Pi^m=\{\mathcal{V}_1^m, \mathcal{V}_2^m, \ldots, \mathcal{V}_K^m\}$ of the coarsest hypergraph $\mathcal{H}^m$ before direct K-way refinement. For this purpose, we use the multi-level RB scheme of PaToH to partition $\mathcal{H}^m$ into K parts. We have observed that it is better to avoid further coarsening within PaToH since $\mathcal{H}^m$ is already coarse enough. At each bisection step of PaToH, we use the greedy hypergraph growing heuristic to bisect the intermediate hypergraphs and the tight boundary FM heuristic [16,17] for refinement. At the end of the initial partitioning phase, if the current imbalance is over the allowed imbalance ratio ε, a balancer, which performs vertex moves (starting with the moves having highest FM gains, i.e., the highest reduction in the cutsize) among the K parts at the expense of an increase in the cutsize, is executed to drop the imbalance below ε.

Although possibilities other than RB exist for generating the initial set of vertex parts, RB emerges as a viable and practical method. A partition of the coarsest hypergraph $\mathcal{H}^m$ generated by RB is very amenable to FM-based refinement since $\mathcal{H}^m$ contains nets of small sizes and vertices of high degrees.

3.3. Multi-level uncoarsening with direct K-way refinement

Every uncoarsening level ℓ includes a refinement step, followed by a projection step. In the refinement step, which involves a number of passes, partition Π^{ℓ} is refined by moving vertices among the K vertex parts, trying to minimize the cutsize while maintaining the balance constraint. In the projection step, the current coarse hypergraph $\mathcal{H}^{\ell}$ and partition Π^{ℓ} are projected back to $\mathcal{H}^{\ell-1}$ and $\Pi^{\ell-1}$. The refinement and projection steps are iteratively repeated until the top-level, flat hypergraph $\mathcal{H}^0$ with a partition Π^0 is obtained. This algorithm is similar to the one described in [36].

At the very beginning of the uncoarsening phase, a connectivity data structure Λ and a lookup data structure δ are created. These structures keep the connectivity of the cut nets to the vertex parts. Λ is a 2D ragged array, where each 1D array keeps the connectivity set of a cut net. That is, $\Lambda(n_i)$ returns the connectivity set Λ_i of a cut net n_i. No information is stored in Λ for internal nets. δ is an $|\mathcal{N}_{\text{cut}}|$ by K, 2D array to lookup the connectivity of a cut net to a part in constant time. That is, $\delta(n_i, \mathcal{V}_k)$ returns the number of the pins that cut net n_i has in part $\mathcal{V}_k$, i.e., $\delta(n_i, \mathcal{V}_k)=|Pins(n_i) \cap \mathcal{V}_k|$.

Both Λ and δ structures are allocated once at the beginning of the uncoarsening phase and maintained during the projection steps. For this purpose, after each coarsening level, a mapping between the nets of the fine and coarse hypergraphs is computed so that Λ and δ arrays are modified appropriately in the corresponding projection steps. Since the pin counts of the nets on the parts may change due to the uncoarsening, the pin counts in the δ array are updated by iterating over the cut nets in the coarse hypergraph and using the net map created during the corresponding coarsening phase. Similar to the net map, vertex maps are computed in the coarsening steps to be able to determine the part assignments of the vertices in the fine hypergraphs during the projection. Part assignments of vertices are kept in a 1D *Part* array, where $Part(v_i)$ shows the current part of vertex v_i.

During the refinement passes, only boundary vertices are considered for movement. For this purpose, a FIFO queue B of boundary vertices is maintained. A vertex v_i is boundary if it is among the pins of at least one cut net n_j. B is updated after each vertex move if the move causes some non-boundary vertices to become boundary or some boundary vertices become internal to a part. Each vertex v_i has a lock count b_i, indicating the number of times v_i is inserted into B. The lock counts are initially set to 0 at the beginning of each refinement pass. Every time a vertex enters B, its lock count is incremented by 1. No vertex v_i with a b_i value greater than a prespecified threshold is allowed to re-enter B. This way, we avoid moving the same vertices repeatedly. The boundary vertex queue B is randomly shuffled at the beginning of each refinement pass.

For vertex movement, each boundary vertex $v_i \in B$ is considered in turn. The move of vertex v_i is considered only to those parts that are in the union of the connectivity sets of the nets connecting v_i, excluding the part containing v_i, if the move satisfies the balance criterion. Note that once the imbalance on part weights is below ε, it is never allowed to rise above this ratio during the direct K-way refinement. After gains are computed, the vertex is moved to the part with the highest positive FM gain. Moves with negative FM gains as well as moves with non-positive leave gains are not performed. A refinement pass terminates when queue B becomes empty. No more refinement passes are made if a predetermined pass count is reached or improvement in the cutsize drops below a prespecified threshold.

For FM-based move gain computation for a vertex v_i, we use the highly efficient algorithm given in Fig. 2. This algorithm first iterates over all nets connecting vertex v_i and computes the leave gain for v_i. If the leave gain is not positive, no further positive move gain is possible and hence the algorithm simply returns. Otherwise, the maximum arrival loss is computed by iterating over all nets connecting v_i as well as a separate move gain for all parts (excluding v_i's current part) that are connected by at least one cut net of v_i. Finally, a total move gain is computed for each part by adding the move gain for the part to the leave gain and subtracting the maximum arrival loss. The

```
COMPUTE-FM-GAINS($v_i$, $Part$, $\Lambda$, $\delta$)
    ▷ compute the leave gain by iterating over all nets connecting $v_i$
    $leaveGain \leftarrow 0$
    for each net $n_j \in Nets(v_i)$ do
        if $\delta(n_j, Part(v_i)) = 1$ then
            $leaveGain \leftarrow gain + c(n_j)$
    ▷ if there is no positive leave gain, simply return
    if $leaveGain = 0$ then
        return NO-PART-TO-MOVE
    ▷ compute the maximum arrival loss by iterating over all nets connecting $v_i$
    ▷ and the move gains for all parts (excluding $Part(v_i)$)
    ▷ connected by at least one of the cut nets of $v_i$
    $targetParts \leftarrow \emptyset$
    $maxArrivalLoss \leftarrow 0$
    for each net $n_j \in Nets(v_i)$ do
        $maxArrivalLoss \leftarrow maxArrivalLoss + c(n_j)$
        if $\lambda_j > 1$ then
            ▷ find the parts $v_i$ can move and compute the respective move gains
            for each part $\mathcal{V}_k \in \Lambda(n_j) - Part(v_i)$ do
                if $\mathcal{V}_k \notin targetParts$ then
                    ▷ insert part $\mathcal{V}_k$ in the set of parts that $v_i$ can move to
                    $targetParts \leftarrow targetParts \cup \{\mathcal{V}_k\}$
                    $MoveGain(\mathcal{V}_k) \leftarrow 0$
                ▷ update the move gain to part $\mathcal{V}_k$
                $MoveGain(\mathcal{V}_k) \leftarrow MoveGain(\mathcal{V}_k) + c(n_j)$
    ▷ compute the maximum move gain and the part $v_i$ will be moved to
    $maxMoveGain \leftarrow -\infty$
    for each part $\mathcal{V}_k \in targetParts$ do
        ▷ check if the move satisfies the balance criterion
        if $W(\mathcal{V}_k) + w(v_i) < (1+\epsilon)W_{\text{avg}}$ then
            $MoveGain(\mathcal{V}_k) \leftarrow MoveGain(\mathcal{V}_k) + (leaveGain - maxArrivalLoss)$
            if $MoveGain(\mathcal{V}_k) > maxMoveGain$ then
                $maxMoveGain \leftarrow MoveGain(\mathcal{V}_k)$
                $maxPart \leftarrow \mathcal{V}_k$
    return $\langle maxPart, maxMoveGain \rangle$
```

Fig. 2. The algorithm for computing the K-way FM gains of a vertex v_i.

maximum move gain is determined by taking the maximum of the total move gains.

3.4. Extension to multiple constraints

Extension to multi-constraint formulation requires verifying the balance constraint for each weight component. As before, zero-gain moves are not performed. During the coarsening phase, the maximum allowed vertex weight is set according to the constraint which has the maximum total vertex weight over all vertices. In the initial partitioning phase, the multi-constraint RB-based partitioning feature of PaToH is used with default parameters to obtain an initial K-way partition.

4. Extensions to hypergraphs with fixed vertices

Our extension to partitioning hypergraphs with fixed vertices follows the multi-level paradigm, which is, in our case, composed of three phases: coarsening with modified heavy-connectivity matching, initial partitioning with maximum-weighted bipartite graph matching, and K-way direct refinement with locked fixed vertices. Throughout the presentation, we assume that, at each coarsening/uncoarsening level ℓ, f_i^ℓ is a fixed vertex in the set $\mathcal{F}^\ell$ of fixed vertices, and o_j^ℓ is an ordinary vertex in the set $\mathcal{O}^\ell$ of ordinary vertices, where $\mathcal{O}^\ell = \mathcal{V}^\ell - \mathcal{F}^\ell$. For each part $\mathcal{V}_k^0$, there is a set $\mathcal{F}_k^0$ of fixed vertices that must end up in $\mathcal{V}_k^0$ at the end of the partitioning such that $\mathcal{F}^0 = \mathcal{F}_1^0 \cup \mathcal{F}_2^0 \cdots \cup \mathcal{F}_K^0$. We also assume that the weights of the fixed vertex sets are fairly balanced.

For the coarsening phase of our algorithm, we modify the heavy-connectivity matching heuristic such that no two fixed vertices $f_i^\ell \in \mathcal{F}^\ell$ and $f_j^\ell \in \mathcal{F}^\ell$ are matched at any coarsening level ℓ. However, any fixed vertex f_i^ℓ in a fixed vertex set $\mathcal{F}_k^\ell$ can be matched with an ordinary vertex $o_j^\ell \in \mathcal{O}^\ell$, forming a fixed supervertex $f_i^{\ell+1}$ in $\mathcal{F}_k^{\ell+1}$. Ordinary vertices are matched as before. Consequently, fixed vertices are propagated throughout the coarsening such that $|\mathcal{F}_k^{\ell+1}| = |\mathcal{F}_k^\ell|$, for $k = 1, 2, \ldots, K$ and $\ell = 0, 1, \ldots, m-1$. Hence, in the coarsest hypergraph $\mathcal{H}^m$, there are $|\mathcal{F}^m| = |\mathcal{F}^0|$ fixed supervertices.

In the initial partitioning phase, a hypergraph $\tilde{\mathcal{H}}^m = (\mathcal{O}^m, \tilde{\mathcal{N}}^m)$ that is free from fixed vertices is formed by temporarily removing fixed supervertices from $\mathcal{H}^m$. In $\tilde{\mathcal{H}}^m$, $\tilde{\mathcal{N}}^m$ is a subset of nets in $\mathcal{N}^m$ whose pins contain at least two ordinary vertices, i.e., $\tilde{\mathcal{N}}^m = \{n_i^m : n_i^m \in \mathcal{N}^m \wedge |\mathcal{O}^m \cap Pins(n_i^m)| > 1\}$. Note that the nets that connect only one ordinary vertex are not retained in $\tilde{\mathcal{H}}^m$ since single-pin nets do not contribute to the cutsize at all. After $\tilde{\mathcal{H}}^m$ is formed, it is partitioned to obtain a K-way vertex partition $\tilde{\Pi}^m = \{\mathcal{O}_1^m, \mathcal{O}_2^m, \ldots, \mathcal{O}_K^m\}$ over the set $\mathcal{O}^m$ of ordinary vertices. Partition $\tilde{\Pi}^m$ induces an initial part assignment for each ordinary vertex in $\mathcal{V}^m$, i.e., $o_i^m \in \mathcal{O}_k^m \Rightarrow Part(v_i^m) = \mathcal{V}_k^m$. However, this initial assignment induced by $\tilde{\Pi}^m$ may not be appropriate in terms of the cutsize since fixed vertices are not considered at all in computation of the cutsize. Note that $cutsize(\tilde{\Pi}^m)$ is a lower bound on $cutsize(\Pi^m)$. A net n_j has the potential to increase the cutsize by its cost times the number of fixed vertex parts that it connects, i.e., by $c(n_j^m)\bar{\lambda}_j^m$, where $\bar{\lambda}_j^m = |\bar{\Lambda}_j^m| = |\{\mathcal{F}_k^m : Pins(n_j^m) \cap \mathcal{F}_k^m \neq \emptyset\}|$. Therefore, $U = cutsize(\tilde{\Pi}^m) + \sum_{n_j^m} c(n_j^m)\bar{\lambda}_j^m$ is an upper bound on $cutsize(\Pi^m)$. At this point, a relabeling of ordinary vertex parts must be found so that the cutsize is tried to be minimized as the fixed vertices are assigned to appropriate parts. We formulate this relabeling problem as a maximum-weighted bipartite graph matching problem [12]. This formulation is valid for any number of fixed vertices.

In the proposed formulation, the sets of fixed supervertices and the ordinary vertex parts form the two node sets of a bipartite graph $\mathcal{B} = (\mathcal{X}, \mathcal{Y})$. That is, in $\mathcal{B}$, for each fixed vertex set $\mathcal{F}_k^m$, there exists a node $x_k \in \mathcal{X}$, and for each ordinary vertex part $\mathcal{O}_\ell^m$ of $\tilde{\Pi}^m$, there exists a node $y_\ell \in \mathcal{Y}$. The bipartite graph contains all possible (x_k, y_ℓ) edges, initially with zero weights. The weight of an edge (x_k, y_ℓ) is increased by the cost of every net that connects at least one vertex in both $\mathcal{F}_k^m$ and $\mathcal{O}_\ell^m$. That is, a net n_j^m increases the weight of edge (x_k, y_ℓ) by its cost $c(n_j^m)$ if and only if $Pins(n_j^m) \cap \mathcal{F}_k^m \neq \emptyset$ and $Pins(n_j^m) \cap \mathcal{O}_\ell^m \neq \emptyset$. This weight on edge (x_k, y_ℓ) corresponds to a saving of $c(n_j^m)$ from upper bound U if $\mathcal{F}_k^m$ is matched with $\mathcal{O}_\ell^m$.

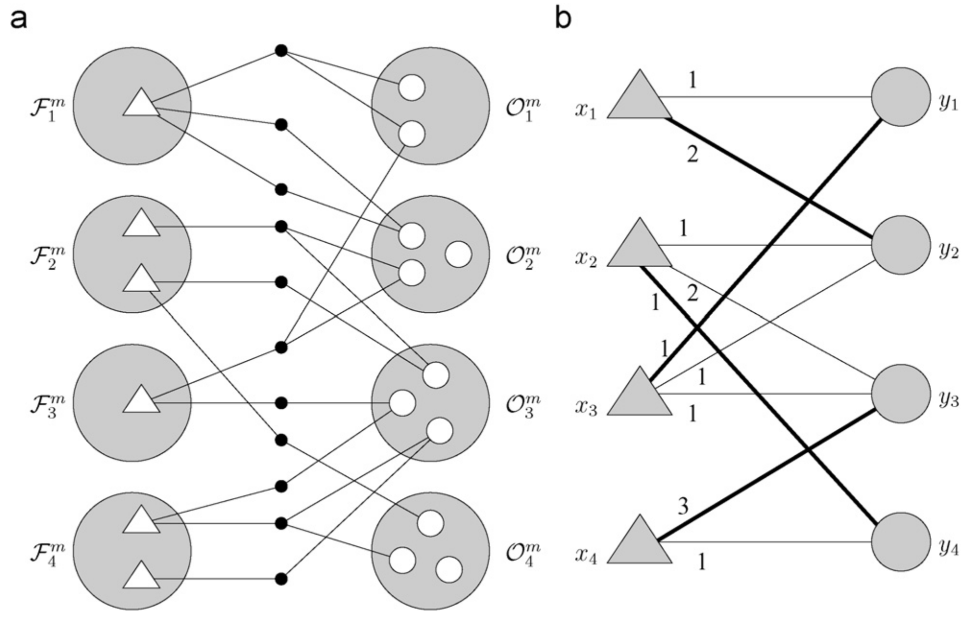

Fig. 3. (a) A sample coarse hypergraph. (b) Bipartite graph representing the hypergraph in Fig. 3(a) and assignment of parts to fixed vertex sets via maximum-weighted matching shown by bold edges.

In this setting, finding the maximum-weighted matching in bipartite graph $\mathcal{B}$ corresponds to finding a matching between fixed vertex sets and ordinary vertex parts which has the minimum increase in the cutsize when fixed vertices are re-introduced into $\tilde{\mathcal{H}}^m$. Each edge (x_k, y_ℓ) in the resulting maximum-weighted matching $\mathcal{M}$ matches a fixed vertex set to an ordinary vertex part. Using $\mathcal{M}$, ordinary vertex parts are relabeled. Vertices in $\mathcal{O}_\ell^m$ are reassigned to part $\mathcal{V}_k^m$ if and only if edge (x_k, y_ℓ) is in $\mathcal{M}$. Note that the fixed vertices in $\mathcal{F}_k^m$ are in part $\mathcal{V}_k^m$ and hence the partition conforms to the given partition on fixed vertices. This relabeling induces an initial partition Π^m. Here, $cutsize(\Pi^m)=U - weight(\mathcal{M})$, where $weight(\mathcal{M})$ is the weight of matching $\mathcal{M}$ and is equal to the saving on the cutsize. Since $\mathcal{M}$ is the maximum-weighted matching, this defines an optimum solution for the relabeling problem.

Fig. 3(a) shows a sample, coarse hypergraph $\mathcal{H}^m$, where fixed and ordinary vertices are, respectively, represented as triangles and circles. For ease of presentation, unit net costs are assumed and only the nets connecting the fixed vertices and ordinary vertices are displayed since all cost contribution on the edges of the constructed bipartite graph are due to such nets. Note that, in this example, the upper bound on $cutsize(\Pi^m)$ is $U=3+11=14$. The linear assignment of fixed vertex sets to ordinary vertex parts (i.e., $\mathcal{F}_k^m$ matched with $\mathcal{O}_k^m$, for $k=1, 2, \ldots, K$) has a cost saving of $weight(\mathcal{M})=1+1+1+1=4$. Hence, $cutsize(\Pi^m)=U - weight(\mathcal{M})=14-4=10$.

Fig. 3(b) displays the bipartite graph constructed for the sample hypergraph in Fig. 3(a), without displaying the zero-weight edges for clarity. In this figure, triangles and circles denote the sets of fixed vertices and ordinary vertex parts, respectively. As seen in Fig. 3(b), there exists an edge (x_2, y_3) with a weight of 2. This is because two unit-cost nets connect $\mathcal{F}_2^m$ and $\mathcal{O}_3^m$. In the figure, the set of bold edges shows the maximum-weighted matching $\mathcal{M}=\{(x_1, y_2), (x_2, y_4), (x_3, y_1), (x_4, y_3)\}$, which assigns the vertices in $\mathcal{F}_1^m$, $\mathcal{F}_2^m$, $\mathcal{F}_3^m$, and $\mathcal{F}_4^m$ to $\mathcal{O}_2^m$, $\mathcal{O}_4^m$, $\mathcal{O}_1^m$, and $\mathcal{O}_3^m$, respectively. As seen in the figure, matching $\mathcal{M}$ obtains the highest possible cost saving of $weight(\mathcal{M})=2+1+1+3=7$. Hence, $cutsize(\Pi^m)=U - weight(\mathcal{M})=14-7=7$. This cutsize is $10-7=3$ less than the cutsize achieved by linear assignment.

During the K-way refinement phase, Π^m is refined using a modified version of the algorithm described in Section 3.3. Throughout the uncoarsening, the fixed vertices are locked to their parts and are not allowed to move between the parts. Hence, each fixed vertex f_i^0 whose corresponding supervertex in the mth level is f_i^m ends up in part $\mathcal{V}_k^0$ if and only if $f_i^m \in \mathcal{F}_k$.

5. Experiments

5.1. Experimental platform

In the experiments, a Pentium IV 3.00 GHz PC with 1 GB of main memory, 512 KB of L2 cache, and 8 KB of L1 cache is used. All algorithms are implemented in C and are compiled in gcc with -O3 optimization option. Due to the randomized nature of some of the heuristics, the results are reported by averaging the values obtained in 20 different runs, each randomly seeded.

The hypergraphs used in the experiments are the row-net hypergraphs [17] of some widely used square sparse matrices obtained from the University of Florida Sparse Matrix Collection [21]. The properties of the hypergraphs are given in Table 1, where the hypergraphs are sorted in increasing order of the number of pins. In all hypergraphs, the number of nets is equal to the number of vertices, and the average vertex degree is equal to the average net size since all matrices are square matrices. Since the internal data structures maintained during

Table 1
Properties of the hypergraphs used in the experiments

Data set	# of vertices	# of pins	Avg. net size
dawson5	51, 537	1, 010, 777	19.61
language	399, 130	1, 216, 334	3.05
Lin	256, 000	1, 766, 400	6.90
poisson3Db	85, 623	2, 374, 949	27.74
helm2d03	392, 257	2, 741, 935	6.99
stomach	213, 360	3, 021, 648	14.16
barrier2-1	113, 076	3, 805, 068	33.65
Hamrle3	1, 447, 360	5, 514, 242	3.81
pre2	659, 033	5, 959, 282	9.04
cage13	445, 135	7, 479, 343	16.80
hood	220, 542	10, 768, 436	48.83
bmw3_2	227, 362	11, 288, 630	49.65

the partitioning do not fit into the memory for the Hamrle3, cage13, and pre2 data sets, they are partitioned on a PC with 2 GB main memory, all other parameters remaining the same. We compare the proposed algorithm, referred to here as kPaToH, with PaToH [16] for two reasons. First, the implementation of kPaToH is based on PaToH. Second, in previous experiments (e.g., see [14,27]), PaToH was found to be performing better than the other hypergraph partitioners.

In the tables, the minimum cutsizes (Min cutsize) and average cutsizes (Avg cutsize) achieved by both partitioners are reported over all data sets together with their average partitioning times (Avg time), for varying number K of parts, where $K \in \{32, 64, 128, 256, 512\}$. The rightmost two columns in Tables 2, 5, 6, and 8 show the percent average cutsize improvement (%Cutsize) and the speedup (Spdup) of kPaToH over PaToH. The averages over all data sets are displayed as a separate entry at the bottom of these tables. Unless otherwise stated, the maximum number of K-way refinement passes in kPaToH is set to 3. Since identical net elimination brings around 5% improvement on the execution time of kPaToH but no speedup on PaToH, we run both PaToH and kPaToH without identical net elimination for a fair comparison. In single-constraint partitioning, weight $w^1(v_i)$ of a vertex v_i is set equal to its vertex degree $d(v_i)$, i.e., $w^1(v_i)=d(v_i)$. The allowed imbalance threshold is set to 10% and is met in all experiments. PaToH is executed with default options.

5.2. Experiments on standard hypergraph partitioning

Table 2 displays the performance comparison of PaToH and kPaToH for standard hypergraph partitioning. According to the averages over all data sets, as K increases, kPaToH begins to perform better in reducing the cutsize compared to PaToH. The average cutsize improvement of 4.82% at $K=32$ rises to 6.81% at $K=256$. A similar behavior is observed in the improvement of kPaToH over PaToH in the minimum cutsizes achieved. In the speedups kPaToH obtains over PaToH, a slight decrease is observed as K increases. However, even at $K=256$, kPaToH runs 1.62 times faster than PaToH and is 1.78 times faster on the overall average.

According to Table 2, except for a single case (the language data set with $K=32$), kPaToH achieves lower cutsizes than PaToH for all data sets and K values. In general, kPaToH performs relatively better in reducing the cutsize on hypergraphs having average net sizes between 6 and 20. This is expected since PaToH is already very effective in partitioning hypergraphs with low net sizes (e.g., language and Hamrle3). On the other hand, in partitioning hypergraphs with very large net sizes (e.g., barrier2-1 and bmw3_2), the performance gap between the partitioners begins to decrease. This is mainly due to performing only the moves with positive gains. Such moves are rare when the nets are highly connected to the parts.

Tables 3 and 4 show the overall percent execution time dissection of the PaToH and kPaToH algorithms, respectively. The tables further display the percent execution time dissection of coarsening and uncoarsening phases for both algorithms. These experiments are conducted on three data sets (language, pre2, and hood) each with different average net size characteristics (3.05, 9.04, and 48.83, respectively), for $K=32$ and 256. The dissections of both PaToH and kPaToH algorithms are given according to the traditional multi-level partitioning paradigm, which involves an initialization phase followed by the coarsening, initial partitioning, and uncoarsening phases. In case of PaToH, these phases are repeated for each hypergraph produced via bisection, and hence the cost of splitting the hypergraph into two after bisections is also considered.

According to Tables 3 and 4, the main overhead of PaToH is at the coarsening step, whereas the percent overhead of uncoarsening is relatively more dominant in case of kPaToH. In general, as K increases from 32 to 256, percent uncoarsening and splitting overheads of PaToH slightly increase. In case of kPaToH, the K-way initial partitioning phase is what most suffers from large K values. In kPaToH, the behavior of the uncoarsening phase is data set dependent. As the average net size increases, the percent overhead of uncoarsening begins to increase with increasing K. This is because the refinement step, which takes the most of the uncoarsening time, is not affected by changing K if the average net size is low as in the case of the language data set. In the hood data set, the increase in the percent overhead of the uncoarsening phase from 27.1% to 57.7% as K goes from 32 to 256 is due to the increase in the solution space, which prevents quick termination of the refinement phase.

5.3. Experiments on multi-constraint partitioning

Tables 5 and 6 show the performance of PaToH and kPaToH in multi-constraint partitioning (2 and 4 constraints, respectively). In the 2-constraint case, a unit weight is used as the second vertex weight for all vertices, i.e., $w^2(v_i)=1$, in addition to the first vertex weight $w^1(v_i)=d(v_i)$. In the 4-constraint case, random integer weights $w^3(v_i)=\alpha_i$, where $1 \leqslant \alpha_i \leqslant w^1(v_i)-1$, and $w^4(v_i)=w^1(v_i)-\alpha_i$ are, respectively, used as the third and fourth vertex weights.

As seen from Tables 5 and 6, kPaToH performs much better than PaToH. A comparison of Tables 2, 5, and 6 shows

Table 2
Performance of PaToH and kPaToH in partitioning hypergraphs with a single partitioning constraint and no fixed vertices

Data set	K	Min. cutsize		Avg. cutsize		Avg. time		Improvement	
		PaToH	kPaToH	PaToH	kPaToH	PaToH	kPaToH	%Cutsize	Spdup
dawson5	32	6, 959	6,286	7, 468	6,907	1.524	0.715	7.51	2.13
	64	11, 293	10,136	11, 907	10,643	1.809	0.934	10.62	1.94
	128	19, 058	17,140	19, 393	17,767	2.099	1.291	8.39	1.63
	256	29, 655	28,035	30, 351	28,396	2.380	1.762	6.44	1.35
language	32	94, 210	94,178	95, 399	95,956	12.266	9.721	−0.58	1.26
	64	107, 299	106,728	108, 432	107,758	13.064	9.830	0.62	1.33
	128	119, 636	117,781	120, 234	119,184	13.835	9.992	0.87	1.38
	256	131, 251	130,679	131, 690	131,526	14.489	10.303	0.12	1.41
Lin	32	49, 458	43,926	50, 800	44,733	5.763	4.751	11.94	1.21
	64	68, 994	60,107	70, 645	60,832	6.632	5.505	13.89	1.20
	128	91, 701	79,910	93, 622	80,878	7.471	6.510	13.61	1.15
	256	119, 529	105,567	121, 346	105,916	8.327	7.942	12.72	1.05
poisson3Db	32	40, 599	38,212	41, 759	39,314	9.358	7.867	5.85	1.19
	64	59, 198	56,075	60, 013	57,371	10.407	9.072	4.40	1.15
	128	84, 630	81,849	86, 118	82,896	11.366	10.416	3.74	1.09
	256	121, 733	114,384	123, 051	116,147	12.240	11.738	5.61	1.04
helm2d03	32	13, 016	12,487	13, 591	12,965	7.689	2.845	4.61	2.70
	64	19, 677	18,841	20, 251	19,236	8.757	3.228	5.01	2.71
	128	29, 169	27,660	29, 696	28,096	9.801	3.790	5.38	2.59
	256	42, 763	40,517	43, 079	40,950	10.850	4.717	4.94	2.30
stomach	32	26, 231	25,757	27, 054	26,184	6.635	3.327	3.22	1.99
	64	37, 885	36,732	38, 918	37,113	7.795	4.097	4.64	1.90
	128	54, 651	52,150	55, 370	52,817	8.968	5.175	4.61	1.73
	256	78, 289	74,863	79, 143	75,572	10.156	6.774	4.51	1.50
barrier2-1	32	52, 877	51,472	53, 560	52,623	9.797	7.292	1.75	1.34
	64	73, 864	71,879	75, 037	73,149	11.135	8.609	2.52	1.29
	128	102, 750	99,629	104, 035	100,679	12.406	9.895	3.23	1.25
	256	142, 833	135,074	143, 995	136,757	13.526	11.372	5.03	1.19
Hamrle3	32	35, 728	35,419	36, 814	36,747	21.190	8.798	0.18	2.41
	64	52, 475	51,813	53, 770	52,885	24.201	9.772	1.65	2.48
	128	75, 818	73,923	76, 851	75,194	26.802	11.418	2.16	2.35
	256	106, 555	105,704	107, 983	106,384	29.187	13.687	1.48	2.13
pre2	32	82, 591	75,860	85, 456	80,238	24.406	15.070	6.11	1.62
	64	108, 714	99,609	112, 486	105,476	28.484	16.929	6.23	1.68
	128	139, 605	120,469	143, 879	122,822	32.250	18.071	14.64	1.78
	256	177, 310	137,899	183, 037	141,091	35.702	19.743	22.92	1.81
cage13	32	369, 330	339,563	373, 617	345,740	45.887	45.590	7.46	1.01
	64	490, 789	448,407	497, 744	455,056	51.035	49.528	8.58	1.03
	128	643, 278	584,178	647, 609	589,316	55.754	52.972	9.00	1.05
	256	824, 294	749,315	829, 962	752,394	59.928	56.450	9.35	1.06
hood	32	22, 799	22,204	24, 392	23,041	15.693	5.386	5.54	2.91
	64	37, 877	37,058	39, 855	38,239	18.383	6.607	4.05	2.78
	128	60, 039	56,903	61, 087	58,198	20.983	8.073	4.73	2.60
	256	91, 007	86,009	92, 367	87,284	23.515	10.303	5.50	2.28
bmw3_2	32	29, 861	28,298	31, 129	29,792	15.383	5.545	4.30	2.77
	64	44, 208	42,465	45, 376	43,820	18.150	6.682	3.43	2.72
	128	65, 752	63,652	67, 551	64,956	20.853	8.065	3.84	2.59
	256	100, 504	97,714	102, 548	99,341	23.454	10.196	3.13	2.30
AVERAGE	32	1.000	0.950	1.000	0.952	1.000	0.606	4.82	1.88
	64	1.000	0.947	1.000	0.945	1.000	0.611	5.47	1.85
	128	1.000	0.938	1.000	0.938	1.000	0.633	6.18	1.77
	256	1.000	0.934	1.000	0.932	1.000	0.679	6.81	1.62

Table 3
Percent dissection of PaToH execution time

Execution phase	language		pre2		hood	
	$K=32$	$K=256$	$K=32$	$K=256$	$K=32$	$K=256$
Initialization	1.0	0.9	1.1	0.8	2.1	1.4
Coarsening	73.5	70.9	84.5	82.8	86.3	84.6
Visit order computation	0.8	1.2	0.5	0.6	0.2	0.3
Vertex matching	72.3	70.0	73.6	74.2	80.7	81.4
Vertex mapping	2.2	2.7	1.1	1.1	0.5	0.5
Hypergraph construction	24.7	26.1	24.8	24.1	18.6	17.8
Initial partitioning	1.9	1.9	1.0	1.3	0.4	0.9
Uncoarsening	19.1	20.6	9.5	10.6	4.6	5.8
Initial gain computations	27.0	27.0	32.8	30.7	19.9	20.1
Refinement	44.2	43.4	17.4	24.1	18.5	29.4
Projection	28.8	29.6	49.8	45.2	61.6	50.5
Splitting	4.5	5.7	3.9	4.5	6.6	7.3

Table 4
Percent dissection of kPaToH execution time

Execution phase	language		pre2		hood	
	$K=32$	$K=256$	$K=32$	$K=256$	$K=32$	$K=256$
Initialization	2.9	2.8	3.7	2.8	12.0	6.4
Coarsening	45.2	38.5	32.1	22.6	56.8	28.8
Visit order computation	0.4	0.4	0.6	0.6	0.2	0.2
Vertex matching	81.0	84.5	70.5	73.5	76.7	77.9
Vertex mapping	1.3	1.3	1.2	1.2	0.5	0.6
Hypergraph construction	17.3	13.8	27.7	24.7	22.6	21.3
Initial partitioning	4.5	11.3	4.4	11.8	4.1	7.1
Uncoarsening	47.4	47.4	59.8	62.8	27.1	57.7
Initialization	1.5	6.2	1.3	5.4	1.5	4.3
Refinement	88.3	82.7	91.6	84.8	87.9	85.6
Projection	10.2	11.1	7.1	9.8	10.6	10.1

that increasing number of partitioning constraints favors kPaToH. On average, the cutsize improvement of kPaToH over PaToH, which is 5.82% in the single-constraint case, increases to 20.98% in the 2-constraint case and further increases to 40.02% in the 4-constraint case. In other words, the degradation in the solution qualities of kPaToH is considerably less than that of PaToH as the number of constraints increases.

The speedups, although being slightly lower, are close to the speedups in the single-constraint case. This slight decrease stems from the fact that the overhead that multi-constraint partitioning introduces to the FM-based refinement algorithm is higher in kPaToH compared to PaToH.

5.4. Experiments on partitioning with fixed vertices

In experiments on partitioning hypergraphs with fixed vertices, we use hypergraphs emerging in a real-life problem [10]. The properties of the hypergraphs are given in Table 7. In naming the data sets, the numbers after the dash indicate the number F of fixed vertices in the hypergraph, e.g., there are $F=32$ fixed vertices in the `BF-32` data set. In the experiments, each part is assigned an equal number of fixed vertices. In `CC` data sets, the net size variation is low, whereas, in `BF` and `OP` data sets, the net sizes show high variation.

Table 8 illustrates the performance results obtained in partitioning hypergraphs with fixed vertices. These results are impressive as kPaToH outperforms PaToH by up to 31.28% in reducing the cutsize. On the overall average, kPaToH incurs a lower cutsize (between 17.09% and 21.07%) than PaToH. At the same time, kPaToH is around 2.3 times faster on the average.

In general, kPaToH shows better cutsize performance than PaToH as K decreases and F increases. This is due to the fact that the disability of PaToH to recursively bisect fixed vertices between two parts in a way that will lead to better cutsizes in the following bisections becomes more pronounced if the number of fixed vertices per part is high. In general, compared to PaToH, the relative performance of kPaToH in minimizing the cutsize is better in `BF` and `OP` data sets, which have high variation in net sizes.

Table 5
Performance of PaToH and kPaToH in partitioning hypergraphs with two partitioning constraints

Data set	K	Min. cutsize		Avg. cutsize		Avg. time		Improvement	
		PaToH	kPaToH	PaToH	kPaToH	PaToH	kPaToH	%Cutsize	Spdup
dawson5	32	11, 294	7,721	12, 598	8,209	1.418	0.749	34.84	1.89
	64	18, 342	12,206	19, 446	13,240	1.673	1.003	31.91	1.67
	128	28, 382	21,065	30, 553	22,383	1.919	1.380	26.74	1.39
	256	45, 929	34,469	48, 331	35,921	2.142	1.839	25.68	1.16
language	32	110, 620	100,817	114, 748	102,026	10.342	9.547	11.09	1.08
	64	124, 426	113,295	127, 849	114,223	11.024	9.662	10.66	1.14
	128	135, 843	125,074	140, 173	126,498	11.742	9.489	9.76	1.24
	256	149, 615	138,889	154, 821	139,876	12.159	10.024	9.65	1.21
Lin	32	60, 912	44,377	62, 727	46,004	5.060	4.661	26.66	1.09
	64	84, 861	61,455	86, 483	62,713	5.805	5.386	27.49	1.08
	128	114, 890	81,905	117, 727	83,223	6.509	6.385	29.31	1.02
	256	151, 652	108,679	153, 346	109,597	7.177	7.702	28.53	0.93
poisson3Db	32	47, 813	38,750	50, 122	41,726	8.138	7.858	16.75	1.04
	64	71, 849	58,218	74, 269	59,538	9.099	8.939	19.83	1.02
	128	104, 590	84,667	108, 143	85,879	9.964	10.213	20.59	0.98
	256	152, 908	119,271	154, 651	120,802	10.703	11.515	21.89	0.93
helm2d03	32	21, 292	13,322	22, 531	14,056	6.491	2.875	37.62	2.26
	64	30, 305	20,056	32, 557	20,693	7.384	3.244	36.44	2.28
	128	44, 819	29,254	46, 078	30,100	8.240	3.788	34.68	2.18
	256	62, 859	42,427	64, 195	43,225	9.046	4.694	32.67	1.93
stomach	32	34, 168	26,555	35, 787	27,906	6.051	3.365	22.02	1.80
	64	48, 082	38,487	49, 632	39,917	7.088	4.154	19.57	1.71
	128	66, 512	55,439	68, 199	56,549	8.115	5.211	17.08	1.56
	256	92, 662	77,772	95, 056	79,367	9.128	6.693	16.51	1.36
barrier2-1	32	63, 376	55,270	65, 498	57,687	8.711	6.958	11.92	1.25
	64	89, 650	78,398	92, 626	80,831	9.876	8.354	12.73	1.18
	128	125, 234	110,596	127, 423	113,046	10.949	9.881	11.28	1.11
	256	171, 482	151,321	177, 107	155,823	11.922	11.449	12.02	1.04
Hamrle3	32	49, 678	37,991	54, 846	39,470	19.498	9.015	28.04	2.16
	64	66, 303	54,298	74, 097	56,011	22.257	10.013	24.41	2.22
	128	94, 701	77,382	99, 669	79,023	24.763	11.544	20.71	2.14
	256	132, 449	110,070	135, 964	110,872	27.072	13.729	18.46	1.97
pre2	32	106, 199	80,682	114, 920	88,445	22.688	15.680	23.04	1.45
	64	139, 973	112,521	155, 620	121,046	26.474	17.567	22.22	1.51
	128	200, 692	168,097	207, 614	173,300	29.936	19.665	16.53	1.52
	256	270, 510	216,747	280, 857	224,725	33.100	22.134	19.99	1.50
cage13	32	432, 428	368,686	443, 298	381,828	37.214	45.613	13.87	0.82
	64	568, 292	485,297	582, 279	499,532	41.490	49.207	14.21	0.84
	128	736, 109	635,187	746, 979	649,153	45.307	52.391	13.10	0.86
	256	942, 314	824,163	957, 385	832,816	48.754	56.081	13.01	0.87
hood	32	30, 184	23,793	32, 279	26,388	14.767	5.552	18.25	2.66
	64	48, 580	40,285	50, 910	42,097	17.206	6.695	17.31	2.57
	128	73, 857	61,670	76, 913	63,410	19.544	8.078	17.56	2.42
	256	112, 224	92,946	114, 197	94,975	21.818	10.050	16.83	2.17
bmw3_2	32	42, 905	32,222	45, 457	34,446	14.068	5.718	24.22	2.46
	64	60, 947	49,109	65, 546	51,274	16.512	6.982	21.77	2.36
	128	94, 851	73,221	101, 070	77,762	18.881	8.500	23.06	2.22
	256	148, 610	114,732	157, 599	118,964	21.160	10.881	24.51	1.94
Average	32	1.000	0.777	1.000	0.776	1.000	0.691	22.36	1.66
	64	1.000	0.797	1.000	0.785	1.000	0.697	21.55	1.63
	128	1.000	0.807	1.000	0.800	1.000	0.723	20.03	1.55
	256	1.000	0.807	1.000	0.800	1.000	0.779	19.98	1.42

Table 6
Performance of PaToH and kPaToH in partitioning hypergraphs with four partitioning constraints

Data set	K	Min. cutsize		Avg. cutsize		Avg. time		Improvement	
		PaToH	kPaToH	PaToH	kPaToH	PaToH	kPaToH	%Cutsize	Spdup
dawson5	32	13, 737	8, 037	14, 781	8, 600	1.439	0.768	41.82	1.87
	64	19, 318	13, 124	22, 841	13, 909	1.692	1.037	39.11	1.63
	128	33, 860	22, 513	36, 084	23, 885	1.941	1.425	33.81	1.36
	256	51, 794	38, 509	56, 280	39, 956	2.161	1.917	29.01	1.13
language	32	139, 353	101, 153	141, 895	102, 487	9.580	9.756	27.77	0.98
	64	148, 714	113, 760	151, 633	114, 933	10.229	10.014	24.20	1.02
	128	156, 375	125, 636	163, 899	127, 946	10.842	10.161	21.94	1.07
	256	165, 782	141, 125	175, 689	145, 699	11.365	11.131	17.07	1.02
Lin	32	91, 234	44, 949	98, 966	46, 184	5.019	4.709	53.33	1.07
	64	120, 349	62, 380	125, 700	63, 297	5.730	5.431	49.64	1.06
	128	152, 362	83, 325	157, 968	84, 282	6.399	6.432	46.65	0.99
	256	187, 114	109, 471	192, 952	110, 604	7.031	7.754	42.68	0.91
poisson3Db	32	64, 204	39, 871	72, 387	42, 610	8.029	7.953	41.14	1.01
	64	92, 385	58, 116	95, 745	61, 079	8.965	9.080	36.21	0.99
	128	124, 979	85, 237	129, 528	88, 489	9.775	10.361	31.68	0.94
	256	170, 152	121, 870	175, 514	125, 055	10.496	11.642	28.75	0.90
helm2d03	32	24, 307	13, 639	27, 429	14, 398	6.701	2.907	47.51	2.31
	64	37, 354	20, 599	38, 828	21, 319	7.642	3.298	45.09	2.32
	128	51, 410	30, 342	53, 462	31, 087	8.541	3.856	41.85	2.21
	256	69, 835	44, 079	73, 373	44, 834	9.420	4.782	38.90	1.97
stomach	32	47, 275	26, 410	51, 908	28, 022	6.038	3.378	46.02	1.79
	64	65, 598	39, 246	69, 666	40, 304	7.063	4.172	42.15	1.69
	128	85, 852	56, 244	89, 528	57, 189	8.092	5.225	36.12	1.55
	256	115, 517	80, 401	118, 783	81, 813	9.097	6.830	31.12	1.33
barrier2-1	32	87, 700	57, 078	93, 946	59, 268	8.633	7.066	36.91	1.22
	64	113, 469	80, 217	121, 148	83, 535	9.817	8.405	31.05	1.17
	128	150, 990	113, 913	159, 412	116, 518	10.841	10.080	26.91	1.08
	256	203, 583	161, 848	208, 792	164, 935	11.864	11.711	21.00	1.01
Hamrle3	32	105, 671	38, 587	115, 453	39, 776	20.145	9.071	65.55	2.22
	64	139, 438	52, 876	146, 122	55, 972	22.900	10.126	61.70	2.26
	128	175, 186	77, 831	181, 742	79, 631	25.409	11.685	56.18	2.17
	256	216, 312	110, 674	222, 124	111, 749	27.646	13.905	49.69	1.99
pre2	32	222, 989	88, 430	240, 992	92, 342	21.903	16.071	61.68	1.36
	64	280, 190	115, 309	288, 496	128, 862	25.308	18.233	55.33	1.39
	128	333, 089	178, 247	349, 267	189, 243	28.463	20.495	45.82	1.39
	256	407, 435	237, 041	418, 959	251, 744	31.515	23.375	39.91	1.35
cage13	32	734, 084	370, 648	780, 736	384, 668	34.957	46.229	50.73	0.76
	64	881, 612	492, 560	928, 273	505, 252	38.757	50.003	45.57	0.78
	128	1, 040, 360	643, 380	1, 073, 786	654, 515	42.286	52.965	39.05	0.80
	256	1, 222, 315	824, 847	1, 257, 893	839, 154	45.385	56.801	33.29	0.80
hood	32	46, 844	25, 305	50, 503	27, 537	14.786	5.575	45.47	2.65
	64	68, 600	41, 293	74, 043	43, 224	17.212	6.695	41.62	2.57
	128	97, 104	63, 427	102, 604	65, 988	19.536	8.063	35.69	2.42
	256	140, 910	96, 733	145, 102	98, 625	21.798	10.026	32.03	2.17
bmw3_2	32	56, 881	33, 049	64, 026	35, 743	14.105	5.816	44.17	2.43
	64	83, 150	51, 511	89, 492	54, 225	16.518	7.085	39.41	2.33
	128	116, 628	76, 555	127, 693	81, 583	18.853	8.888	36.11	2.12
	256	177, 088	120, 850	185, 200	125, 376	21.093	11.153	32.30	1.89
Average	32	1.000	0.549	1.000	0.532	1.000	0.716	46.84	1.64
	64	1.000	0.585	1.000	0.574	1.000	0.725	42.59	1.60
	128	1.000	0.634	1.000	0.624	1.000	0.757	37.65	1.51
	256	1.000	0.680	1.000	0.670	1.000	0.817	32.98	1.37

Table 7
Properties of the hypergraphs used in the experiments on partitioning hypergraphs with fixed vertices

Data set	# of vertices	# of nets	# of pins	Avg. net size
`BF-32`	28, 930	4, 800	688,018	143.34
`BF-64`	28, 962	9, 600	930,412	96.92
`BF-128`	29, 026	19, 200	1,335,049	69.53
`CC-32`	56, 374	4, 800	1,133,858	236.22
`CC-64`	56, 406	9, 600	1,472,295	153.36
`CC-128`	56, 470	19, 200	2,094,107	109.07
`OP-32`	68, 190	4, 800	1,276,595	265.96
`OP-64`	68, 222	9, 600	1,629,169	169.70
`OP-128`	68, 286	19, 200	1,924,807	100.25

Table 8
Performance of PaToH and kPaToH in partitioning hypergraphs with fixed vertices

Data set	*K*	Min. cutsize		Avg. cutsize		Avg. time		Improvement	
		PaToH	kPaToH	PaToH	kPaToH	PaToH	kPaToH	%Cutsize	Spdup
`BF-32`	32	9,474	7,507	9,639	7,604	5.394	2.018	21.11	2.67
	64	11,343	9,379	11,799	9,623	5.906	2.186	18.44	2.70
	128	14,962	12,695	15,212	12,916	6.309	2.373	15.09	2.66
`BF-64`	32	17,790	13,538	18,625	13,691	5.152	2.088	26.49	2.47
	64	21,473	16,583	22,010	16,867	5.726	2.309	23.37	2.48
	128	25,548	21,354	26,406	21,652	6.284	2.585	18.00	2.43
`BF-128`	32	34,522	24,234	35,751	24,568	5.770	2.709	31.28	2.13
	64	39,837	28,855	41,521	29,366	6.569	3.003	29.28	2.19
	128	47,448	36,180	48,652	36,589	7.006	3.298	24.79	2.12
`CC-32`	32	9,534	8,438	9,668	8,619	4.865	2.248	10.85	2.16
	64	12,608	10,931	12,927	11,123	5.547	2.472	13.96	2.24
	128	17,635	14,796	17,873	14,956	6.172	2.771	16.32	2.23
`CC-64`	32	17,466	15,349	17,952	15,512	4.623	2.671	13.59	1.73
	64	21,397	19,161	21,740	19,316	5.344	3.040	11.15	1.76
	128	28,088	24,685	28,729	25,010	6.012	3.282	12.94	1.83
`CC-128`	32	33,201	28,803	34,298	28,986	5.407	3.677	15.49	1.47
	64	40,036	34,947	40,677	35,225	6.233	4.218	13.40	1.48
	128	49,454	43,960	50,315	44,321	6.965	4.556	11.91	1.53
`OP-32`	32	8,717	6,953	8,935	7,042	18.714	6.159	21.18	3.04
	64	10,367	8,530	10,804	8,622	19.485	6.372	20.19	3.06
	128	13,155	11,135	13,463	11,313	21.275	6.697	15.97	3.18
`OP-64`	32	15,693	12,507	16,402	12,668	17.462	6.010	22.76	2.91
	64	18,823	15,028	19,399	15,178	19.317	6.291	21.76	3.07
	128	22,972	18,745	23,404	19,238	20.020	6.757	17.80	2.96
`OP-128`	32	30,418	22,440	31,076	22,717	13.119	5.073	26.90	2.59
	64	34,735	25,876	35,157	26,505	14.981	5.425	24.61	2.76
	128	39,643	31,807	40,642	32,133	16.047	5.971	20.94	2.69
Average	32	1.000	0.802	1.000	0.789	1.000	0.448	21.07	2.35
	64	1.000	0.814	1.000	0.804	1.000	0.437	19.57	2.42
	128	1.000	0.835	1.000	0.829	1.000	0.437	17.09	2.40

In Table 9, we report performance results in terms of the cutsize on a subset of the hypergraphs selected from Table 1. In each hypergraph, we randomly fix $F=128, 256, 512$ vertices to $K=128, 256, 512$ parts in a round-robin fashion and partition the hypergraph using both PaToH and kPaToH. For kPaToH, we explore the percent improvement due to bipartite graph matching (BGM) and hence try it both with and without BGM. In Table 9, "kPaToH w/ BGM" corresponds to an implementation

Table 9
Performance of kPaToH with and without bipartite graph matching (BGM) in partitioning hypergraphs with a randomly selected number (F) of fixed vertices

Data set	F	K	Avg. cutsize			%Improvement		%Share of BGM in the improvement
			PaToH	kPaToH w/o BGM	kPaToH w/ BGM	kPaToH w/o BGM	kPaToH w/ BGM	
dawson5	256	128	26,227	23,010	20,992	12.26	19.96	38.56
		256	36,828	33,651	30,285	8.63	17.77	51.44
		512	56,948	49,299	45,409	13.43	20.26	33.71
	512	128	33,075	28,005	25,345	15.33	23.37	34.42
		256	43,058	38,715	34,038	10.09	20.95	51.85
		512	62,390	54,470	47,475	12.69	23.91	46.90
	1024	128	46,633	37,819	34,654	18.90	25.69	26.42
		256	56,410	48,648	43,244	13.76	23.34	41.04
		512	73,354	64,288	55,273	12.36	24.65	49.86
Lin	256	128	91,505	82,284	81,888	10.08	10.51	4.12
		256	118,981	107,683	106,572	9.50	10.43	8.95
		512	155,004	140,019	138,752	9.67	10.48	7.79
	512	128	94,202	84,155	83,421	10.67	11.44	6.80
		256	120,629	109,462	108,008	9.26	10.46	11.53
		512	155,286	141,911	139,816	8.61	9.96	13.55
	1024	128	99,117	87,418	86,626	11.80	12.60	6.34
		256	124,302	112,920	111,368	9.16	10.41	12.00
		512	158,567	145,222	142,332	8.42	10.24	17.80
poisson3Db	256	128	96,136	89,682	86,009	6.71	10.53	36.27
		256	132,245	123,777	118,546	6.40	10.36	38.19
		512	184,110	168,285	162,525	8.60	11.72	26.69
	512	128	104,717	96,503	92,188	7.84	11.96	34.44
		256	140,800	130,267	124,592	7.48	11.51	35.02
		512	191,542	174,627	164,621	8.83	14.05	37.17
	1024	128	126,271	109,786	104,960	13.06	16.88	22.65
		256	158,922	144,133	135,716	9.31	14.60	36.27
		512	208,279	188,168	175,242	9.66	15.86	39.13
barrier2-1	256	128	115,348	108,922	105,160	5.57	8.83	36.93
		256	154,860	144,915	139,278	6.42	10.06	36.18
		512	212,822	194,371	187,355	8.67	11.97	27.55
	512	128	126,627	116,548	112,418	7.96	11.22	29.07
		256	166,085	153,156	145,217	7.78	12.56	38.04
		512	221,905	202,422	189,665	8.78	14.53	39.57
	1024	128	151,989	133,302	127,497	12.29	16.11	23.70
		256	188,254	169,670	159,889	9.87	15.07	34.48
		512	243,276	218,533	202,652	10.17	16.70	39.09
hood	256	128	84,003	70,391	64,793	16.20	22.87	29.14
		256	112,839	99,714	90,857	11.63	19.48	40.29
		512	159,561	142,146	131,641	10.91	17.50	37.62
	512	128	107,231	82,145	75,616	23.39	29.48	20.65
		256	133,568	111,861	100,191	16.25	24.99	34.96
		512	175,277	154,121	136,864	12.07	21.92	44.93
	1024	128	149,171	106,871	99,501	28.36	33.30	14.84
		256	172,577	136,753	122,510	20.76	29.01	28.45
		512	212,997	179,269	156,507	15.83	26.52	40.29
bmw3_2	256	128	89,754	78,213	72,883	12.86	18.80	31.59
		256	121,386	112,222	103,552	7.55	14.69	48.61
		512	174,406	161,878	152,451	7.18	12.59	42.94
	512	128	110,871	90,326	83,413	18.53	24.77	25.18
		256	141,551	125,157	113,760	11.58	19.63	41.01
		512	189,793	174,442	156,349	8.09	17.62	54.10
	1024	128	150,756	115,426	107,445	23.43	28.73	18.43
		256	178,616	149,789	136,220	16.14	23.74	32.01
		512	223,685	200,207	176,784	10.50	20.97	49.94

Table 10
Averages of the results provided in Table 9 over F and K

Averages over F				Averages over K			
F	%Improvement		% Share of BGM in the improvement	K	% Improvement		% Share of BGM in the improvement
	kPaToH w/o BGM	kPaToH w/ BGM			kPaToH w/o BGM	kPaToH w/ BGM	
256	9.57	14.38	32.03	128	14.18	18.73	24.42
512	11.40	17.46	33.29	256	10.64	16.61	34.46
1024	14.10	20.24	29.60	512	10.25	16.75	36.03

of the approach described in Section 4, whereas "kPaToH w/o BGM" corresponds to an implementation in which the fixed vertex sets are arbitrarily matched with the ordinary vertex parts.

Basically, kPaToH provides superior results over PaToH in partitioning hypergraphs with fixed vertices due to (i) K-way refinement and (ii) BGM performed during the the initial partitioning phase. We provide Table 9 to illustrate the share of these two factors in the overall cutsize improvement. According to Table 9, the share of BGM in the total cutsize improvement is quite variable, ranging between 4.12% and 54.10%. In general, better results are achieved by BGM for hypergraphs having large net sizes. It is hard to make a judgment about the behavior with increasing F and K values. This is due to the fact that many other factors affect the performance of BGM. Among these factors, the performance is dependent on the connectivity of the regular and fixed vertices, distribution of fixed vertices on the parts, the ratio of F to the number of regular vertices, and the ratio of F to K. These are hard to assess (also see [1] for comments on the bipartitioning case). The averages over F and K values are provided in Table 10.

6. Conclusion

We argued that the hypergraph partitioning with multiple constraints and fixed vertices problems should be tackled with a direct K-way refinement approach. In order to support our claim, we presented a careful implementation of a multi-level direct K-way refinement algorithm. We discussed extensions of this algorithm for partitioning hypergraphs with multiple constraints and fixed vertices. Extension to the multi-constraint case adopts standard hypergraph partitioning techniques. In order to extend the algorithm to the fixed vertex case, we proposed specialized coarsening and initial partitioning with a novel formulation that uses bipartite graph matching. The experiments conducted on benchmark data sets indicate that the proposed algorithm is quite fast and effective in minimizing the cutsize compared to the state-of-the-art hypergraph partitioning tool PaToH. Especially, in the multi-constraint and fixed vertices domain, the obtained results are quite promising in terms of both execution time and solution quality. On average, the cutsize improvement of kPaToH over PaToH is around 20% and 40% at the 2-constraint and 4-constraint cases, respectively. Similarly, in partitioning hypergraphs with fixed vertices, kPaToH outperforms PaToH by up to 33% in cutsize as it runs 2.3 times faster on average.

References

[1] C.J. Alpert, A.E. Caldwell, A.B. Kahng, I.L. Markov, Hypergraph partitioning with fixed vertices, IEEE Trans. Comput.-Aided Design 19 (2) (2000) 267–272.
[2] C.J. Alpert, A.B. Kahng, Recent directions in netlist partitioning: a survey, VLSI J. 19 (1–2) (1995) 1–81.
[3] C. Ashcraft, Compressed graphs and the minimum degree algorithm, SIAM J. Sci. Comput. 16 (6) (1995) 1404–1411.
[4] C. Aykanat, A. Pinar, Ü.V. Çatalyürek, Permuting sparse rectangular matrices into block-diagonal form, SIAM J. Sci. Comput. 25 (6) (2004) 1860–1879.
[5] C. Berge, Graphs and Hypergraphs, North-Holland Publishing Company, Amsterdam, 1973.
[6] R.H. Bisseling, J. Byrka, S. Cerav-Erbas, N. Gvozdenovic, M. Lorenz, R. Pendavingh, C. Reeves, M. Roger, A. Verhoeven, Partitioning a call graph, in: Second International Workshop on Combinatorial Scientific Computing, 2005.
[7] R.H. Bisseling, I. Flesch, Mondriaan sparse matrix partitioning for attacking cryptosystems by a parallel block Lanczos algorithm: a case study, Parallel Comput. 32 (7) (2006) 551–567.
[8] T.N. Bui, C. Jones, A heuristic for reducing fill in sparse matrix factorization, in: Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 445–452.
[9] A. Caldwell, A. Kahng, I. Markov, Improved algorithms for hypergraph bipartitioning, in: Proceedings of the IEEE ACM Asia and South Pacific Design Automation Conference, 2000, pp. 661–666.
[10] B.B. Cambazoglu, C. Aykanat, Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids, IEEE Trans. Parallel Distributed Systems 18 (1) (2007) 3–16.
[11] C. Chang, T.M. Kurc, A. Sussman, Ü.V. Çatalyürek, J.H. Saltz, A hypergraph-based workload partitioning strategy for parallel data aggregation, in: SIAM Conference on Parallel Processing for Scientific Computing, 2001.
[12] G. Chartrand, O.R. Oellermann, Applied and Algorithmic Graph Theory, McGraw-Hill, New York, 1993.
[13] C. Clifton, R. Cooley, J. Rennie, TopCat: data mining for topic identification in a text corpus, IEEE Trans. Knowledge Data Eng. 16 (8) (2004) 949–964.
[14] Ü.V. Çatalyürek, ISPD98 benchmark ⟨http://bmi.osu.edu/∼umit/PaToH/ispd98.html⟩.
[15] Ü.V. Çatalyürek, C. Aykanat, Decomposing irregularly sparse matrices for parallel matrix–vector multiplication, Lecture Notes in Computer Science, vol. 1117, 1996, pp. 75–86.
[16] Ü.V. Çatalyürek, C. Aykanat, PaToH: partitioning tool for hypergraphs, Technical Report, Department of Computer Engineering, Bilkent University, 1999.
[17] Ü.V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Trans. Parallel Distributed Systems 10 (7) (1999) 673–693.
[18] Ü.V. Çatalyürek, C. Aykanat, A fine-grain hypergraph model for 2D decomposition of sparse matrices, in: Proceedings of the 15th International Parallel and Distributed Processing Symposium, 2001, p. 118.

[19] Ü.V. Çatalyürek, C. Aykanat, A hypergraph-partitioning approach for coarse-grain decomposition, in: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, 2001, p. 28.
[20] A. Dasdan, C. Aykanat, Two novel multiway circuit partitioning algorithms using relaxed locking, IEEE Trans. Comput.-Aided Design Integrated Circuits Systems 16 (2) (1997) 169–178.
[21] T. Davis, University of Florida Sparse Matrix Collection ⟨http://www.cise.ufl.edu/research/sparse/matrices⟩, NA Digest 97 (23) (June 7, 1997).
[22] E. Demir, C. Aykanat, B.B. Cambazoglu, Clustering spatial networks for aggregate query processing: a hypergraph approach, Inform. Systems, in press.
[23] E. Demir, C. Aykanat, B.B. Cambazoglu, A link-based storage scheme for efficient aggregate query processing on clustered road networks, Technical Report, BU-CE-0707, Department of Computer Engineering, Bilkent University, 2007.
[24] K.D. Devine, E.G. Boman, R.T. Heaphy, R. Bisseling, Ü.V. Çatalyürek, Parallel hypergraph partitioning for scientific computing, in: Proceedings of the IEEE International Parallel and Distributed Processing Symposium, 2006.
[25] K.D. Devine, E.G. Boman, R.T. Heaphy, B. Hendrickson, C. Vaughan, Zoltan data management services for parallel dynamic applications, Comput. Sci. Eng. 4 (2) (2002) 90–97.
[26] N.J. Dingle, P.G. Harrison, W.J. Knottenbelt, Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models, J. Parallel Distributed Comput. 64 (8) (2004) 908–920.
[27] I.S. Duff, S. Riyavong, M.B. van Gijzen, Parallel preconditioners based on partitioning sparse matrices, Technical Report, TR/PA/04/114, CERFACS, 2004.
[28] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the 19th ACM/IEEE Design Automation Conference, 1982, pp. 175–181.
[29] M.K. Goldberg, M. Burnstein, Heuristic improvement technique for bisection of VLSI networks, in: Proceedings of the IEEE International Conference on Computer Design, 1983, pp. 122–125.
[30] B. Hendrickson, R. Leland, The Chaco user's guide: version 2.0, Technical Report, SAND94-2692, Sandia National Laboratories, 1994.
[31] B. Hendrickson, E. Rothberg, Improving the run time and quality of nested dissection ordering, SIAM J. Sci. Comput. 20 (2) (1998) 468–489.
[32] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Multilevel hypergraph partitioning: applications in VLSI domain, IEEE Trans. Very Large Scale Integration Systems 7 (1) (1999) 69–79.
[33] G. Karypis, V. Kumar, hMETIS: a hypergraph partitioning package, Technical Report, Department of Computer Science, University of Minnesota, 1998.
[34] G. Karypis, V. Kumar, MeTiS: a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices, Technical Report, Department of Computer Science, University of Minnesota, 1998.
[35] G. Karypis, V. Kumar, Multilevel algorithms for multi-constraint graph par-titioning, in: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, 1998, pp. 1–13.
[36] G. Karypis, V. Kumar, Multilevel *k*-way hypergraph partitioning, VLSI Design 11 (3) (2000) 285–300.
[37] K. Kaya, C. Aykanat, Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master–slave environments, IEEE Trans. Parallel Distributed Systems 17 (8) (2006) 883–896.
[38] K. Kaya, B. Ucar, C. Aykanat, Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories, J. Parallel Distributed Comput. 67 (3) (2007) 271–285.
[39] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical J. 49 (1970) 291–307.
[40] G. Khanna, N. Vydyanathan, T.M. Kurc, Ü.V. Çatalyürek, P. Wyckoff, J. Saltz, P. Sadayappan, A hypergraph partitioning based approach for scheduling of tasks with batch-shared IO, in: Proceedings of Cluster Computing and Grid, 2005.
[41] M. Koyuturk, C. Aykanat, Iterative-improvement-based declustering heuristics for multi-disk databases, Inform. Systems 30 (1) (2005) 47–70.
[42] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley-Teubner, Chichester, 1990.
[43] D.R. Liu, M.Y. Wu, A hypergraph based approach to declustering problems, Distributed Parallel Databases 10 (3) (2001) 269–288.
[44] M.M. Ozdal, C. Aykanat, Hypergraph models and algorithms for data-pattern-based clustering, Data Mining Knowledge Discovery 9 (1) (2004) 29–57.
[45] D.G. Schweikert, B.W. Kernighan, A proper model for the partitioning of electrical circuits, in: Proceedings of the 9th Workshop on Design Automation, 1972, pp. 57–62.
[46] S. Shekhar, C.-T. Lu, S. Chawla, S. Ravada, Efficient join-index-based spatial-join processing: a clustering approach, IEEE Trans. Knowledge Data Eng. 14 (6) (2002) 1400–1421.
[47] H.D. Simon, S.-H. Teng, How good is recursive bisection?, SIAM J. Sci. Comput. 18 (5) (1997) 1436–1445.
[48] A. Trifunovic, W.J. Knottenbelt, Parkway2.0: a parallel multilevel hypergraph partitioning tool, in: Proceedings of the International Symposium on Computer and Information Sciences, 2004, pp. 789–800.
[49] B. Uçar, C. Aykanat, Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix–vector multiplies, SIAM J. Sci. Comput. 25 (6) (2004) 1837–1859.
[50] B. Uçar, C. Aykanat, Revisiting hypergraph models for sparse matrix partitioning, SIAM Rev. 49 (4) (2007) 543–732.
[51] B. Uçar, C. Aykanat, Partitioning sparse matrices for parallel preconditioned iterative methods, SIAM J. Sci. Comput. 29 (4) (2007) 1683–1709.
[52] B. Uçar, C. Aykanat, M.C. Pınar, T. Malas, Parallel image restoration using surrogate constraints methods, J. Parallel Distributed Comput. 67 (2) (2007) 186–204.
[53] B. Vastenhouw, R.H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix-vector multiplication, SIAM Rev. 47 (1) (2005) 67–95.
[54] C. Walshaw, M. Cross, K. McManus, Multiphase mesh partitioning, Appl. Math. Modelling 25 (2000) 123–140.

Cevdet Aykanat received the BS and MS degrees from Middle East Technical University, Ankara, Turkey, both in electrical engineering, and the PhD degree from Ohio State University, Columbus, in electrical and computer engineering. He was a Fulbright scholar during his PhD studies. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Ankara, Turkey, where he is currently a professor. His research interests mainly include parallel computing, parallel scientific computing and its combinatorial aspects, parallel computer graphics applications, parallel data mining, graph and hypergraph partitioning, load balancing, neural network algorithms, high performance information retrieval systems, parallel and distributed web crawling, parallel and distributed databases, and grid computing. He has (co)authored over 40 technical papers published in academic journals indexed in SCI. He is the recipient of the 1995 Young Investigator Award of The Scientific and Technological Research Council of Turkey. He is a member of the ACM and the IEEE Computer Society. He has been recently appointed as a member of IFIP Working Group 10.3 (Concurrent Systems).

Berkant Barla Cambazoglu received his BS, MS, and PhD degrees all in computer engineering from the Computer Engineering Department of Bilkent University in 1997, 2000, and 2006, respectively. He has worked in several research projects funded by the Scientific and Technological Research Council of Turkey, the European Union Sixth Framework Program, and the National Cancer Institute. He is currently working as a postdoctoral researcher at the Biomedical Informatics Department of the Ohio State University. His research interests include high performance computing, scientific visualization, information retrieval, data mining, and grid middleware.

Bora Ucar received the PhD degree (2005) in computer engineering from Bilkent University, Ankara, Turkey. He is currently working at the Parallel Algorithms Project, CERFACS, France. His research interests are combinatorial scientific computing, parallel and high performance computing, and sparse matrix computations.