Multilevel Hypergraph Partitioning: Applications in VLSI Domain

George Karypis, Rajat Aggarwal, Vipin Kumar, *Senior Member, IEEE*, and Shashi Shekhar, *Senior Member, IEEE*

***Abstract*— In this paper, we present a new hypergraph-partitioning algorithm that is based on the multilevel paradigm. In the multilevel paradigm, a sequence of successively coarser hypergraphs is constructed. A bisection of the smallest hypergraph is computed and it is used to obtain a bisection of the original hypergraph by successively projecting and refining the bisection to the next level finer hypergraph. We have developed new hypergraph coarsening strategies within the multilevel framework. We evaluate their performance both in terms of the size of the hyperedge cut on the bisection, as well as on the run time for a number of very large scale integration circuits. Our experiments show that our multilevel hypergraph-partitioning algorithm produces high-quality partitioning in a relatively small amount of time. The quality of the partitionings produced by our scheme are on the average 6%–23% better than those produced by other state-of-the-art schemes. Furthermore, our partitioning algorithm is significantly faster, often requiring 4–10 times less time than that required by the other schemes. Our multilevel hypergraph-partitioning algorithm scales very well for large hypergraphs. Hypergraphs with over 100 000 vertices can be bisected in a few minutes on today's workstations. Also, on the large hypergraphs, our scheme outperforms other schemes (in hyperedge cut) quite consistently with larger margins (9%–30%).**

***Index Terms*— Circuit partitioning, hypergraph partitioning, multilevel algorithms.**

I. Introduction

HYPERGRAPH partitioning is an important problem with extensive application to many areas, including very large scale integration (VLSI) design [1], efficient storage of large databases on disks [2], and data mining [3]. The problem is to partition the vertices of a hypergraph into k roughly equal parts, such that the number of hyperedges connecting vertices in different parts is minimized. A hypergraph is a generalization of a graph, where the set of edges is replaced by a set of hyperedges. A hyperedge extends the notion of an edge by allowing more than two vertices to be connected by a hyperedge. Formally, a hypergraph $H = (V, E^h)$ is defined as a set of vertices V and a set of hyperedges E^h, where each hyperedge is a subset of the vertex set V [4], and the size of a hyperedge is the cardinality of this subset.

Manuscript received April 29, 1997; revised March 23, 1998. This work was supported under IBM Partnership Award NSF CCR-9423082, by the Army Research Office under Contract DA/DAAH04-95-1-0538, and by the Army High Performance Computing Research Center, the Department of the Army, Army Research Laboratory Cooperative Agreement DAAH04-95-2-0003/Contract DAAH04-95-C-0008.

G. Karypis, V. Kumar, and S. Shekhar are with the Department of Computer Science and Engineering, Minneapolis, University of Minnesota, Minneapolis, MN 55455-0159 USA.

R. Aggarwal is with the Lattice Semiconductor Corporation, Milpitas, CA 95131 USA.

Publisher Item Identifier S 1063-8210(99)00695-2.

During the course of VLSI circuit design and synthesis, it is important to be able to divide the system specification into clusters so that the inter-cluster connections are minimized. This step has many applications including design packaging, HDL-based synthesis, design optimization, rapid prototyping, simulation, and testing. In particular, many rapid prototyping systems use partitioning to map a complex circuit onto hundreds of interconnected field-programmable gate arrays (FPGA's). Such partitioning instances are challenging because the timing, area, and input/output (I/O) resource utilization must satisfy hard device-specific constraints. For example, if the number of signal nets leaving any one of the clusters is greater than the number of signal p-i-n's available in the FPGA, then this cluster cannot be implemented using a single FPGA. In this case, the circuit needs to be further partitioned, and thus implemented using multiple FPGA's. Hypergraphs can be used to naturally represent a VLSI circuit. The vertices of the hypergraph can be used to represent the cells of the circuit, and the hyperedges can be used to represent the nets connecting these cells. A high quality hypergraph-partitioning algorithm greatly affects the feasibility, quality, and cost of the resulting system.

A. Related Work

The problem of computing an optimal bisection of a hypergraph is at least NP-hard [5]. However, because of the importance of the problem in many application areas, many heuristic algorithms have been developed. The survey by Alpert and Khang [1] provides a detailed description and comparison of such various schemes. In a widely used class of *iterative refinement partitioning algorithms*, an initial bisection is computed (often obtained randomly) and then the partition is refined by repeatedly moving vertices between the two parts to reduce the hyperedge cut. These algorithms often use the Schweikert–Kernighan heuristic [6] (an extension of the Kernighan–Lin (KL) heuristic [7] for hypergraphs), or the faster Fiduccia–Mattheyses (FM) [8] refinement heuristic, to iteratively improve the quality of the partition. In all of these methods (sometimes also called KLFM schemes), a vertex is moved (or a vertex pair is swapped) if it produces the greatest reduction in the edge cuts, which is also called the gain for moving the vertex. The partition produced by these methods is often poor, especially for larger hypergraphs. Hence, these algorithms have been extended in a number of ways [9]–[12].

Krishnamurthy [9] tried to introduce intelligence in the tie-breaking process from among the many possible moves with the same high gain. He used a *Look Ahead* (LA_r) algorithm, which looks ahead up to r-level of gains before making

a move. PROP [11], introduced by Dutt and Deng, used a probabilistic gain computation model for deciding which vertices need to move across the partition line. These schemes tend to enhance the performance of the basic KLFM family of refinement algorithms, at the expense of increased run time. Dutt and Deng [12] proposed two new methods, namely, CLIP and CDIP, for computing the gains of hyperedges that contain more than one node on either side of the partition boundary. CDIP in conjunction with LA_3 and CLIP in conjunction with PROP are two schemes that have shown the best results in their experiments.

Another class of hypergraph-partitioning algorithms [13]–[16] performs partitioning in two phases. In the first phase, the hypergraph is coarsened to form a small hypergraph, and then the FM algorithm is used to bisect the small hypergraph. In the second phase, these algorithms use the bisection of this contracted hypergraph to obtain a bisection of the original hypergraph. Since FM refinement is done only on the small coarse hypergraph, this step is usually fast, but the overall performance of such a scheme depends upon the quality of the coarsening method. In many schemes, the projected partition is further improved using the FM refinement scheme [15].

Recently, a new class of partitioning algorithms was developed [17]–[20] based upon the multilevel paradigm. In these algorithms, a sequence of successively smaller (coarser) graphs is constructed. A bisection of the smallest graph is computed. This bisection is now successively projected to the next-level finer graph and, at each level, an iterative refinement algorithm such as KLFM is used to further improve the bisection. The various phases of multilevel bisection are illustrated in Fig. 1. Iterative refinement schemes such as KLFM become quite powerful in this multilevel context for the following reason. First, the movement of a single node across a partition boundary in a coarse graph can lead to the movement of a large number of related nodes in the original graph. Second, the refined partitioning projected to the next level serves as an excellent initial partitioning for the KL or FM refinement algorithms. This paradigm was independently studied by Bui and Jones [17] in the context of computing fill-reducing matrix reordering, by Hendrickson and Leland [18] in the context of finite-element mesh-partitioning, and by Hauck and Borriello (called Optimized KLFM) [20], and by Cong and Smith [19] for hypergraph partitioning. Karypis and Kumar extensively studied this paradigm in [21] and [22] for the partitioning of graphs. They presented new graph coarsening schemes for which even a good bisection of the coarsest graph is a pretty good bisection of the original graph. This makes the overall multilevel paradigm even more robust. Furthermore, it allows the use of simplified variants of KLFM refinement schemes during the uncoarsening phase, which significantly speeds up the refinement process without compromising overall quality. METIS [21], a multilevel graph partitioning algorithm based upon this work, routinely finds substantially better bisections and is often two orders of magnitude faster than the hitherto state-of-the-art spectral-based bisection techniques [23], [24] for graphs.

The improved coarsening schemes of METIS work only for graphs and are not directly applicable to hypergraphs. If the

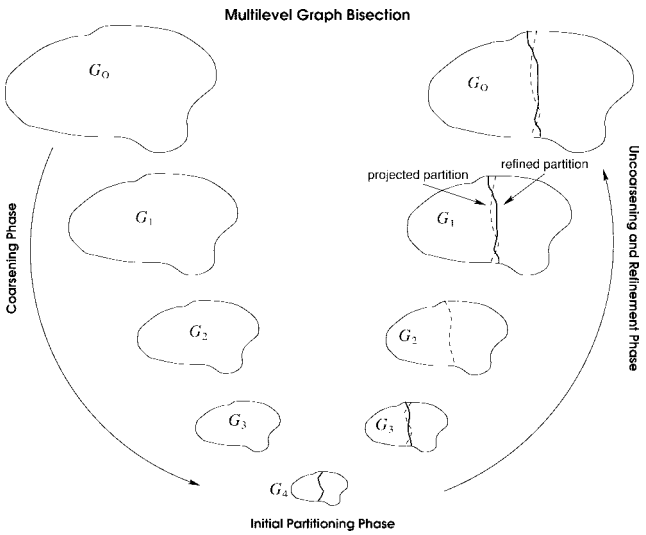


Fig. 1. The various phases of the multilevel graph bisection. During the coarsening phase, the size of the graph is successively decreased; during the initial partitioning phase, a bisection of the smaller graph is computed, and during the uncoarsening and refinement phase, the bisection is successively refined as it is projected to the larger graphs. During the uncoarsening and refinement phase, the dashed lines indicate projected partitionings and dark solid lines indicate partitionings that were produced after refinement. G_0 is the given graph, which is the finest graph. G_{i+1} is the next level coarser graph of G_i, and vice versa, G_i is the next level finer graph of G_{i+1}. G_4 is the coarsest graph.

hypergraph is first converted into a graph (by replacing each hyperedge by a set of regular edges), then METIS [21] can be used to compute a partitioning of this graph. This technique was investigated by Alpert and Khang [25] in their algorithm called GMetis. They converted hypergraphs to graphs by simply replacing each hyperedge with a clique, and then they dropped many edges from each clique randomly. They used METIS to compute a partitioning of each such random graph and then selected the best of these partitionings. Their results show that reasonably good partitionings can be obtained in a reasonable amount of time for a variety of benchmark problems. In particular, the performance of their resulting scheme is comparable to other state-of-the art schemes such as PARABOLI [26], PROP [11], and the multilevel hypergraph partitioner from Hauck and Borriello [20].

The conversion of a hypergraph into a graph by replacing each hyperedge with a clique does not result in an equivalent representation since high-quality partitionings of the resulting graph do not necessarily lead to high-quality partitionings of the hypergraph. The standard hyperedge-to-edge conversion [27] assigns a uniform weight of $1/(|e|-1)$ to each edge in the clique, where $|e|$ is the size of the hyperedge, i.e., the number of vertices in the hyperedge. However, the fundamental problem associated with replacing a hyperedge by its clique is that there exists no scheme to assign weight to the edges of the clique that can correctly capture the cost of cutting this hyperedge [28]. This hinders the partitioning refinement algorithm since vertices are moved between partitions depending on how much this reduces the number of edges they cut in the converted graph, whereas the real objective is to minimize the number of hyperedges cut in the original hypergraph. Furthermore, the hyperedge-to-clique conversion destroys the natural sparsity of the hypergraph, significantly increasing the

run time of the partitioning algorithm. Alpert and Khang [25] solved this problem by dropping many edges of the clique randomly, but this makes the graph representation even less accurate. A better approach is to develop coarsening and refinement schemes that operate directly on the hypergraph. Note that the multilevel scheme by Hauck and Borriello [20] operates directly on hypergraphs and, thus, is able to perform accurate refinement during the uncoarsening phase. However, all coarsening schemes studied in [20] are edge-oriented; i.e., they only merge pairs of nodes to construct coarser graphs. Hence, despite a powerful refinement scheme (FM with the use of look-ahead LA_3) during the uncoarsening phase, their performance is only as good as that of GMetis [25].

B. Our Contributions

In this paper, we present a multilevel hypergraph-partitioning algorithm `hMETIS` that operates directly on the hypergraphs. A key contribution of our work is the development of new hypergraph coarsening schemes that allow the multilevel paradigm to provide high-quality partitions quite consistently. The use of these powerful coarsening schemes also allows the refinement process to be simplified considerably (even beyond plain FM refinement), making the multilevel scheme quite fast. We investigate various algorithms for the coarsening and uncoarsening phases which operate on the hypergraphs without converting them into graphs. We have also developed new multiphase refinement schemes (v- and V-cycles) based on the multilevel paradigm. These schemes take an initial partition as input and try to improve them using the multilevel scheme. These multiphase schemes further reduce the run times, as well as improve the solution quality. We evaluate their performance both in terms of the size of the hyperedge cut on the bisection, as well as on run time on a number of VLSI circuits. Our experiments show that our multilevel hypergraph-partitioning algorithm produces high-quality partitioning in a relatively small amount of time. The quality of the partitionings produced by our scheme are on the average 6%–23% better than those produced by other state-of-the-art schemes [11], [12], [25], [26], [29]. The difference in quality over other schemes becomes even greater for larger hypergraphs. Furthermore, our partitioning algorithm is significantly faster, often requiring 4–10 times less time than that required by the other schemes. For many circuits in the well-known ACM/SIGDA benchmark set [30], our scheme is able to find better partitionings than those reported in the literature for any other hypergraph-partitioning algorithm. The remainder of this paper is organized as follows. Section II describes the different algorithms used in the three phases of our multilevel hypergraph-partitioning algorithm. Section III describes a new partitioning refinement algorithm based on the multilevel paradigm. Section IV compares the results produced by our algorithm to those produced by earlier hypergraph-partitioning algorithms.

II. Multilevel Hypergraph Bisection

We now present the framework of `hMETIS`, in which the coarsening and refinement scheme work directly with hyperedges without using the clique representation to transform them into edges. We have developed new algorithms for both the phases, which, in conjunction, are capable of delivering very good quality solutions.

A. Coarsening Phase

During the coarsening phase, a sequence of successively smaller hypergraphs are constructed. As in the case of multilevel graph bisection, the purpose of coarsening is to create a small hypergraph, such that a good bisection of the small hypergraph is not significantly worse than the bisection directly obtained for the original hypergraph. In addition to that, hypergraph coarsening also helps in successively reducing the sizes of the hyperedges. That is, after several levels of coarsening, large hyperedges are contracted to hyperedges that connect just a few vertices. This is particularly helpful, since refinement heuristics based on the KLFM family of algorithms [6]–[8] are very effective in refining small hyperedges, but are quite ineffective in refining hyperedges with a large number of vertices belonging to different partitions.

Groups of vertices that are merged together to form single vertices in the next-level coarse hypergraph can be selected in different ways. One possibility is to select pairs of vertices with common hyperedges and to merge them together, as illustrated in Fig. 2(a). A second possibility is to merge together all the vertices that belong to a hyperedge, as illustrated in Fig. 2(b). Finally, a third possibility is to merge together a subset of the vertices belonging to a hyperedge, as illustrated in Fig. 2(c). These three different schemes for grouping vertices together for contraction are described below.

1) Edge Coarsening (EC): The heavy-edge matching scheme used in the multilevel-graph bisection algorithm can also be used to obtain successively coarser hypergraphs by merging the pairs of vertices connected by many hyperedges. In this EC scheme, a heavy-edge maximal[1] matching of the vertices of the hypergraph is computed as follows. The vertices are visited in a random order. For each vertex v, all unmatched vertices that belong to hyperedges incident to v are considered, and the one that is connected via the edge with the largest weight is matched with v. The weight of an edge connecting two vertices v and u is computed as the sum of the *edge weights* of all the hyperedges that contain v and u. Each hyperedge e of size $|e|$ is assigned an edge-weight of $1/(|e|-1)$, and as hyperedges collapse on each other during coarsening, their edge weights are added up accordingly.

This EC scheme is similar in nature to the schemes that treat the hypergraph as a graph by replacing each hyperedge with its clique representation [27]. However, this hypergraph-to-graph conversion is done implicitly during matching without forming the actual graph.

2) Hyperedge Coarsening (HEC): Even though the EC scheme is able to produce successively coarser hypergraphs, it decreases the hyperedge weight of the coarser graph only for those pairs of matched vertices that are connected via a hyperedge of size two. As a result, the total hyperedge

[1] One can also compute a maximum weight matching [31]; however, that would have significantly increased the amount of time required by this phase.

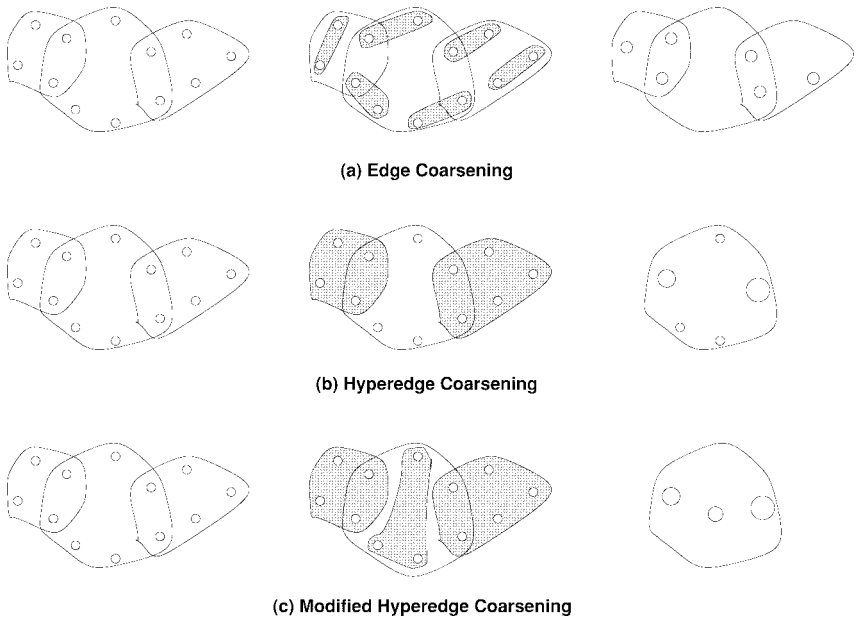


Fig. 2. Various ways of matching the vertices in the hypergraph and the coarsening they induce. (a) In edge-coarsening, connected pairs of vertices are matched together. (b) In hyperedge-coarsening, all the vertices belonging to a hyperedge are matched together. (c) In MHEC, we match together all the vertices in a hyperedge, as well as all the groups of vertices belonging to a hyperedge.

weight of successively coarser graphs does not decrease very fast. In order to ensure that for every group of vertices that are contracted together, there is a decrease in the hyperedge weight in the coarser graph, each such group of vertices must be connected by a hyperedge.

This is the motivation behind the HEC scheme. In this scheme, an independent set of hyperedges is selected and the vertices that belong to individual hyperedges are contracted together. This is implemented as follows. The hyperedges are initially sorted in a nonincreasing hyperedge-weight order and the hyperedges of the same weight are sorted in a nondecreasing hyperedge size order. Then, the hyperedges are visited in that order, and for each hyperedge that connects vertices that have not yet been matched, the vertices are matched together. Thus, this scheme gives preference to the hyperedges that have large weight and those that are of small size. After all of the hyperedges have been visited, the groups of vertices that have been matched are contracted together to form the next level coarser graph. The vertices that are not part of any contracted hyperedges are simply copied to the next level coarser graph.

3) Modified Hyperedge Coarsening (MHEC): The HEC algorithm is able to significantly reduce the amount of hyperedge weight that is left exposed in successively coarser graphs. However, during each coarsening phase, a majority of the hyperedges do not get contracted because vertices that belong to them have been contracted via other hyperedges. This leads to two problems. First, the size of many hyperedges does not decrease sufficiently, making FM-based refinement difficult. Second, the weight of the vertices (i.e., the number of vertices that have been collapsed together) in successively coarser graphs becomes significantly different, which distorts the shape of the contracted hypergraph.

To correct this problem, we implemented a MHEC scheme as follows. After the hyperedges to be contracted have been selected using the HEC scheme, the list of hyperedges is traversed again, and for each hyperedge that has not yet been contracted, the vertices that do not belong to any other contracted hyperedge are contracted together.

B. Initial Partitioning Phase

During the initial partitioning phase, a bisection of the coarsest hypergraph is computed, such that it has a small cut, and satisfies a user-specified balance constraint. The balance constraint puts an upper bound on the difference between the relative size of the two partitions. Since this hypergraph has a very small number of vertices (usually less than 200), the time to find a partitioning using any of the heuristic algorithms tends to be small. Note that it is not useful to find an optimal partition of this coarsest graph, as the initial partition will be substantially modified during the refinement phase. We used the following two algorithms for computing the initial partitioning.

The first algorithm simply creates a random bisection such that each part has roughly equal vertex weight. The second algorithm starts from a randomly selected vertex and grows a region around it in a breadth-first fashion [22] until half of the vertices are in this region. The vertices belonging to the grown region are then assigned to the first part, and the rest of the vertices are assigned to the second part. After a partitioning is constructed using either of these algorithms, the partitioning is refined using the FM refinement algorithm.

Since both algorithms are randomized, different runs give solutions of different quality. For this reason, we perform a small number of initial partitionings. At this point, we can select the best initial partitioning and project it to the original hypergraph, as described in Section II-C. However, the partitioning of the coarsest hypergraph that has the smallest cut may not necessarily be the one that will lead to the smallest cut in the original hypergraph. It is possible that another partitioning of the coarsest hypergraph (with a higher cut) will lead to a bet-

ter partitioning of the original hypergraph after the refinement is performed during the uncoarsening phase. For this reason, instead of selecting a single initial partitioning (i.e., the one with the smallest cut), we propagate all initial partitionings.

Note that propagation of i initial partitionings increases the time during the refinement phase by a factor of i. Thus, by increasing the value of i, we can potentially improve the quality of the final partitioning at the expense of higher run time. One way to dampen the increase in run time due to large values of i is to drop unpromising partitionings as the hypergraph is uncoarsened. For example, one possibility is to propagate only those partitionings whose cuts are within x% of the best partitionings at the current level. If the value of x is sufficiently large, then all partitionings will be maintained and propagated in the entire refinement phase. On the other hand, if the value of x is sufficiently small then, on average, only one partitioning will be maintained, as all other partitionings will be eliminated at the coarsest level. For moderate values of x, many partitionings may be available at the coarsest graph, but the number of such available partitionings will decrease as the graph is uncoarsened. This is useful for two reasons. First, it is more important to have many alternate partitionings at the coarser levels, as the size of the cut of a partitioning at a coarse level is a less accurate reflection of the size of the cut of the original finest level hypergraph. Second, refinement is more expensive at the fine levels, as these levels contain far more nodes than the coarse levels. Hence, by choosing an appropriate value of x, we can benefit from the availability of many alternate partitionings at the coarser levels and avoid paying the high cost of refinement at the finer levels by keeping fewer candidates on average.

In our experiments, as reported in this paper, we find ten initial partitionings at the coarsest graph, and we drop all partitionings whose cut is 10% worse than the best cut at that level. This allows us to both filter out the really bad partitionings (and thus reduce the amount of time spent in refinement) and at the same time keep more than just one promising partitioning (so as to improve the overall partitioning quality). In our experiments, we have seen that by keeping ten partitionings, we can reduce the cut on the average by 3%–4%, whereas the partitioning time increases only by a factor of two. Computing and propagating more partitionings does not further reduce the cut significantly. In our experiments, keeping 20 partitionings further reduces the cut by a factor less than 0.5%, on the average. Increasing the value of parameter x (from 10% to a higher value such as 20%) did not significantly improve the quality of the partitionings, although it did increase the run time.

C. Uncoarsening and Refinement Phase

During the uncoarsening phase, a partitioning of the coarser hypergraph is successively projected to the next-level finer hypergraph, and a partitioning refinement algorithm is used to reduce the cut set (and thus to improve the quality of the partitioning) without violating the user specified balance constraints. Since the next-level finer hypergraph has more degrees of freedom, such refinement algorithms tend to improve the solution quality.

We have implemented two different partitioning refinement algorithms. The first is the FM algorithm [8], which repeatedly moves vertices between partitions in order to improve the cut. The second algorithm, called hyperedge refinement (HER), moves groups of vertices between partitions so that an entire hyperedge is removed from the cut. These algorithms are further described in the remainder of this section.

1) FM: The partitioning refinement algorithm by Fiduccia and Mattheyses [8] is iterative in nature. It starts with an initial partitioning of the hypergraph. In each iteration, it tries to find subsets of vertices in each partition, such that moving them to other partitions improves the quality of the partitioning (i.e., the number of hyperedges being cut decreases) and this does not violate the balance constraint. If such subsets exist, then the movement is performed and this becomes the partitioning for the next iteration. The algorithm continues by repeating the entire process. If it cannot find such a subset, then the algorithm terminates since the partitioning is at a local minima and no further improvement can be made by this algorithm.

In particular, for each vertex v, the FM algorithm computes the ***gain***, which is the reduction in the hyperedge cut achieved by moving v to the other partition. Initially all vertices are *unlocked*, i.e., they are free to move to the other partition. The algorithm iteratively selects an unlocked vertex v with the largest gain (subject to balance constraints) and moves it to the other partition. When a vertex v is moved, it is *locked*, and the gain of the vertices adjacent to v are updated. After each vertex movement, the algorithm also records the size of the cut achieved at this point. Note that the algorithm does not allow locked vertices to be moved since this may result in thrashing (i.e., repeated movement of the same vertex). A single pass of the FM algorithm ends when there are no more unlocked vertices (i.e., all the vertices have been moved). Then, the recorded cut sizes are checked, and the point where the minimum cut was achieved is selected, and all vertices that were moved after that point are moved back to their original partition. Now, this becomes the initial partitioning for the next pass of the algorithm. With the use of appropriate data structures, the complexity of each pass of the FM algorithm is $O(|E^h|)$ [8].

For refinement in the context of multilevel schemes, the initial partitioning obtained from the next level coarser graph is actually a very good partition. For this reason, we can make a number of optimizations to the original FM algorithm. The first optimization limits the maximum number of passes performed by the FM algorithm to only two. This is because the greatest reduction in the cut is obtained during the first or second pass and any subsequent passes only marginally improve the quality. Our experience has shown that this optimization significantly improves the run time of FM without affecting the overall quality of the produced partitionings. The second optimization aborts each pass of the FM algorithm before actually moving all the vertices. The motivation behind this is that only a small fraction of the vertices being moved actually lead to a reduction in the cut and, after some point, the cut tends to increase as we move more vertices. When FM is applied to a random initial partitioning, it is quite likely that after a long sequence of *bad* moves, the algorithm will climb

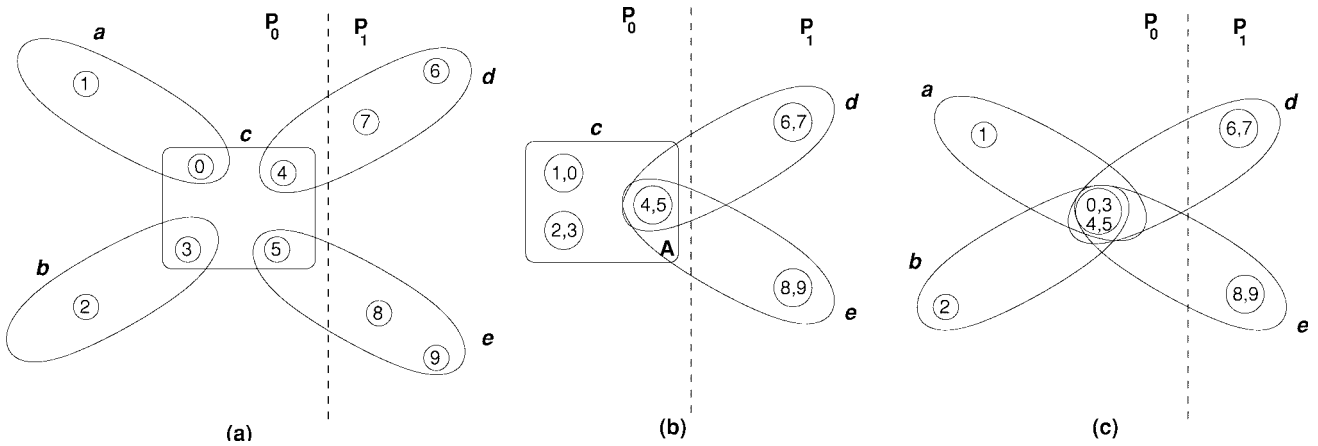


Fig. 3. Effect of *restricted coarsening*. (a) Example hypergraph with a given partitioning with the required balance of 40/60. (b) Possible condensed version of (a). (c) Another condensed version of a hypergraph.

out of a local minima and reach to a better cut. However, in the context of a multilevel scheme, a long sequence of cut-increasing moves rarely leads to a better local minima. For this reason, we stop each pass of the FM algorithm as soon as we have performed k vertex moves that did not improve the cut. We choose k to be equal to 1% of the number of vertices in the graph we are refining. This modification to FM, called ***early-exit FM*** (FM-EE), does not significantly affect the quality of the final partitioning, but it dramatically improves the run time (see Section IV).

2) HER: One of the drawbacks of FM (and other similar vertex-based refinement schemes) is that it is often unable to refine hyperedges that have many nodes on both sides of the partitioning boundary. However, a refinement scheme that moves all the vertices that belong to a hyperedge can potentially solve this problem. Our HER works as follows. It randomly visits all the hyperedges and, for each one that straddles the bisection, it determines if it can move a subset of the vertices incident on it, so that this hyperedge will become completely interior to a partition. In particular, consider a hyperedge e, which straddles the partitioning boundary, and let V_e^0 and V_e^1 be the vertices of e that belong to partition 0 and partition 1, respectively. Our algorithm computes the gain $g_{0\to1}$, which is the reduction in the cut achieved by moving the vertices in V_e^0 to partition 1, and the gain $g_{1\to0}$, which is the reduction in the cut achieved by moving the vertices in V_e^1 to partition 0. Now, depending on these gains and subject to balance constraints, it may move one of the two sets V_e^0 or V_e^1. In particular, if $g_{0\to1}$ is positive and $g_{0\to1} > g_{1\to0}$, it moves V_e^0, and if $g_{1\to0}$ is positive and $g_{1\to0} > g_{0\to1}$, it moves V_e^1.

III. Multiphase Refinement with Restricted Coarsening

Although the multilevel paradigm is quite robust, randomization is inherent in all three phases of the algorithm. In particular, the random choice of vertices to be matched in the coarsening phase can disallow certain hyperedge cuts, reducing refinement in the uncoarsening phase. For example, consider the example hypergraph in Fig. 3(a) and its two possible condensed versions [Fig. 3(b) and (c)] with the same partitioning. The version in Fig. 3(b) is obtained by selecting hyperedges a and b to be compressed in the HEC phase and then selecting pairs of nodes $(4,5)$, $(6,7)$, and $(8,9)$ to be compressed in the modified HEC phase. Similarly, the version shown in Fig. 3(c) is obtained by selecting hyperedge c to be compressed in the HEC phase and then selecting pairs of nodes $(6,7)$ and $(8,9)$ to be compressed in the MHEC phase. In the version of Fig. 3(b), vertex $A(4,5)$ can be moved from partition P_0 to P_1 to reduce the hyperedge cuts by 1, but in Fig. 3(c), no vertex can be moved to reduce the hyperedge cuts.

What this example shows is that, in a multilevel setting, a given initial partitioning of a hypergraph can be potentially refined in many different ways depending upon how the coarsening is performed. Hence, a partitioning produced by a multilevel partitioning algorithm can be potentially further refined if the two partitions are again coarsened in a manner different from the previous coarsening phase (which is easily done given the random nature of all of the coarsening schemes described here). The power of iterative refinement at different coarsening levels can also be used to develop a partitioning refinement algorithm based on the multilevel paradigm. The idea behind this ***multiphase refinement*** algorithm is quite simple. It consists of two phases, namely a coarsening and an uncoarsening phase. The uncoarsening phase of the multiphase refinement algorithm is identical to the uncoarsening phase of the multilevel hypergraph-partitioning algorithm described in Section II-C. The coarsening phase, however, is somewhat different, as it preserves the initial partitioning that is input to the algorithm. We will refer to this as the ***restricted coarsening*** scheme. Given a hypergraph H and a partitioning P, during the coarsening phase, a sequence of successively coarser hypergraphs and their partitionings is constructed. Let (H_i, P_i) for $i = 1, 2, \cdots, m$, be the sequence of hypergraphs and partitionings. Given a hypergraph H_i and its partitioning P_i, restricted coarsening will collapse vertices together that belong to only one of the two partitions. That is, if A and B are the two partitions, we only collapse together vertices that either belong to partition A or partition B. The partitioning P_{i+1} of the next level coarser hypergraph H_{i+1} is computed by simply inheriting the partition from H_i. For example, if a set of vertices $\{v_1, v_2, v_3\}$ from partition A are collapsed together to form vertex u_i of H_{i+1}, then vertex u_i belong

to partition A as well. By constructing H_{i+1} and P_{i+1} in this way, we ensure that the number of hyperedges cut by the partitioning is identical to the number of hyperedges cut by P_i in H_i. The set of vertices to be collapsed together in this restricted coarsening scheme can be selected by using any of the coarsening schemes described in Section II-A, namely, edge coarsening, hyperedge coarsening, or modified hyperedge coarsening. Due to the randomization in the coarsening phase, successive runs of the multiphase refinement algorithm can lead to additional reductions in the hyperedge cut. Thus, the multiphase refinement algorithm can be performed iteratively. Note that during the refinement phase, we only propagate a single partitioning; thus, multiphase refinement is quite fast. In the context of our multilevel hypergraph-partitioning algorithm, this new multiphase refinement can be used in a number of ways. In the remainder of this section, we describe three such approaches.

1) V-Cycle: In this scheme, we take the best solution obtained from the multilevel partitioning algorithm (P_b) and we improve it using multiphase refinement repeatedly. We stop the multiphase refinement when the solution quality cannot be improved further. The number of multiphase refinement steps performed is problem dependent and, in general, it increases as the size of the hypergraph increases. This is due to the larger solution space of the large hypergraphs.

2) v-Cycle: Our experience with the multilevel partitioning algorithm has shown that refining multiple solutions is expensive, especially during the final uncoarsening levels when the size of the contracted hypergraphs is large. One way to reduce the high cost of refining multiple solutions during the final uncoarsening levels is to select the best partitioning at some point in the uncoarsening phase and further refine only this best partitioning using multiphase refinement. This is the idea behind the v-cycle refinement. In particular, let $H_{m/2}$ be the coarse hypergraph at the midpoint between H_0 (original hypergraph) and H_m (coarsest hypergraph). Let $P_{m/2}$ be the best partitioning at $H_{m/2}$. We then use ($H_{m/2}$, $P_{m/2}$) as the input to multiphase refinement. Since $H_{m/2}$ is relatively small, as compared to H_m, multiphase refinement converges in a small number of iterations. By using v-cycles, we can significantly reduce the amount of time spent in the refinement phase, especially for large hypergraphs. However, the overall quality can potentially decrease because we may have not picked up the best overall partitioning at $H_{m/2}$.

3) vV-Cycle: We can combine both V-cycles and v-cycles in the algorithm to obtain high-quality partitioning in a small amount of time. In this scheme, we use v-cycles to partition the hypergraph followed by the V-cycles to further improve the partition quality. V-cycles used in this way are particularly effective in significantly improving the hyperedge cut.

IV. Experimental Results

We experimentally evaluated the quality of the bisections produced by our multilevel hypergraph-partitioning algorithm on a large number of hypergraphs that are part of the widely used ACM/SIGDA circuit partitioning benchmark suite [30]. The characteristics of these hypergraphs are shown in Table I.

TABLE I
Characteristics of the Various Hypergraphs Used to Evaluate the Multilevel Hypergraph Partitioning Algorithms

Benchmark	No. of vertices	No. of hyperedges
balu	801	735
p1	833	902
bm1	882	903
t4	1515	1658
t3	1607	1618
t2	1663	1720
t6	1752	1541
struct	1952	1920
t5	2595	2750
19ks	2844	3282
p2	3014	3029
s9234	5866	5844
biomed	6514	5742
s13207	8772	8651
s15850	10470	10383
industry2	12637	13419
industry3	15406	21923
s35932	18148	17828
s38584	20995	20717
avq.small	21918	22124
s38417	23849	23843
avq.large	25178	25384
golem3	103048	144949

We performed all of our experiments on an SGI Challenge that has MIPS R10000 processors running at 200 MHz, and all of the reported run times are in seconds. All of the reported partitioning results were obtained by forcing a 45–55 balance condition.

As discussed in Sections II-A, II-B, and II-C, there are many alternatives for each of the three different phases of a multilevel algorithm. Due to space limitations, we are not able to provide a comprehensive comparison of the various parameters. However, this comparison can be found in the full version of this paper, which is available on the World Wide Web at: http//www.cs.umn.edu/~karypis/publications.

In the remainder of this section, we present comparisons of our scheme with other partitioning schemes available in the literature.

A. Comparison with Other Partitioning Algorithms

To compare the performance of the bisections produced by our multilevel hypergraph bisection and multiphase refinement algorithms, both in terms of bisection quality and run time, we created Table II. Table II shows the sizes of the hyperedge cuts produced by our algorithms (`hMETIS`)and those reported by various previously developed hypergraph bisection algorithms. In particular, Table II contains results for the following algorithms: PROP [11], $\mathrm{CDIP-LA3}_f$ and $\mathrm{CLIP-PROP}_f$ [12], Optimized KLFM (scheme by Hauck and Borriello [20]), GMetis [25], PARABOLI [26], and GFM [32]. Note that for certain circuits, there are missing results for some of the algorithms. This is because no results were reported for these circuits. The column labeled "Best" shows the minimum cut obtained for each circuit by any of the earlier algorithms. Essentially, this column represents the quality that would have been obtained if all of the algorithms had been run and the best partition was selected.

The last four columns of Table II shows the partitionings produced by our multilevel hypergraph bisection and refine-

TABLE II
PERFORMANCE OF OUR MULTILEVEL HYPERGRAPH BISECTION ALGORITHM (hMETIS) AGAINST VARIOUS PREVIOUSLY DEVELOPED ALGORITHMS

Benchmark	PROP	CDIP-LA3$_f$	CLIP-PROP$_f$	PARABOLI	GFM	GMetis	Opt. KLFM	Best	hMETIS-EE$_{20}$	hMETIS-FM$_{20}$	hMETIS-EE$_{10vV}$	hMETIS-FM$_{20vV}$
balu	27	27	27	41	27	27	–	27	27	27	27	27
p1	47	47	51	53	47	47	–	47	52	50	49	49
bm1	50	47	47	–	–	48	–	47	51	51	51	51
t4	52	48	52	–	–	49	–	48	51	51	48	48
t3	59	57	57	–	–	62	–	57	58	58	59	58
t2	90	89	87	–	–	95	–	87	91	88	92	88
t6	76	60	60	–	–	94	–	60	62	60	63	60
struct	33	36	33	40	41	33	–	33	33	33	33	33
t5	79	74	77	–	–	104	–	74	71	71	71	71
19ks	105	104	104	–	–	106	–	104	107	106	106	105
p2	143	151	152	146	139	142	–	139	148	145	148	145
s9234	41	44	42	74	41	43	45	41	40	40	40	40
biomed	83	83	84	135	84	102	–	83	83	83	83	83
s13207	75	69	71	91	66	74	62	62	55	55	61	53
s15850	65	59	56	91	63	53	46	46	42	42	42	42
industry2	220	182	192	193	211	177	–	177	174	167	169	168
industry3	–	243	243	267	241	243	–	241	255	254	252	241
s35932	–	73	42	62	41	57	46	41	42	42	42	42
s38584	–	47	51	55	47	53	52	47	47	47	48	48
avq.small	–	139	144	224	–	144	–	139	136	130	128	127
s38417	–	74	65	49	81	69	–	49	52	51	54	50
avq.large	–	137	143	139	–	145	–	137	129	127	134	127
golem3	–	–	–	1629	–	2111	–	1629	1447	1445	1425	1424
Sum of Hyperedge-cuts												
5 circuits							251	237	226	226	233	225
13 circuits					1129			1033	1050	1036	1048	1021
16 circuits	1245							1132	1145	1127	1142	1121
16 circuits				3289				2938	2762	2738	2735	2699
22 circuits		1890	1880					1786	1806	1778	1800	1756
23 circuits						4078		3415	3253	3223	3225	3180
hMETIS	**Quality improvement**											
EE$_{20}$	6.2%	5.3%	4.1%	21.4%	7.8%	10.0%	9.9%	0.3%				
FM$_{20}$	7.2%	6.4%	5.2%	22.4%	8.7%	11.0%	9.9%	1.4%	1.1%			
EE$_{10vV}$	6.4%	5.4%	4.1%	21.3%	7.5%	10.1%	7.6%	0.3%	-0.1%	-1.2%		
FM$_{20vV}$	7.9%	7.3%	6.1%	23.1%	9.4%	11.9%	10.1%	2.3%	2.0%	0.9%	2.0%	
Runtime Comparison. The times are in seconds on the specified machines												
	Sparc5	Sparc5	Sparc5	Dec3000 500AXP	Sparc10	Sparc5	Sparc IPX		SGI R10000	SGI R10000	SGI R10000	SGI R10000
5 circuits							5606		95	125	62	180
13 circuits					46376				283	390	173	508
16 circuits	2383								158	224	103	303
16 circuits				37570					874	1593	382	1442
22 circuits		15850	16206						445	637	249	733
23 circuits						3357			913	1654	409	1513

ment algorithms. In particular, the column labeled "hMETIS-EE$_{20}$" corresponds to the best partitioning produced from 20 runs of our multilevel algorithm that uses FM-EE during refinement. Of these 20 runs, ten runs used HEC and ten runs used MHEC. The column labeled "hMETIS-FM$_{20}$" corresponds to the best partitioning produced from 20 runs when FM is used during refinement and coarsening is performed similarly to "hMETIS-EE$_{20}$." In both of these schemes, we used random initial partitionings during the initial partitioning phase.

The column labeled hMETIS-EE$_{10vV}$ corresponds to the best partitioning produced from ten runs of our multilevel partitioning algorithm that uses the vV-cycle refinement scheme. These results were obtained using the MHEC and EE-FM for refinement. Finally, the column labeled hMETIS-FM$_{20vV}$ corresponds to the best partition produced from 20 runs that use the vV-cycles refinement scheme. Out of these runs, ten used HEC and ten used MHEC for coarsening; and the refinement was done using FM.

To make the comparison with previous algorithms easier, we computed the total number of hyperedges cut by each algorithm, as well as the percentage improvement in the cut achieved by our algorithms over previous algorithms. This cut improvement was computed as the average improvement on a circuit-by-circuit level. Looking at these results, we see that all four of our algorithms produce partitionings whose quality is better than that produced by any of the previous algorithms. In particular, hMETIS-EE$_{20}$ is 4.1% better than $\mathrm{CLIP-PROP}_f$, 5.3% better than $\mathrm{CDIP-LA}_f$, 6.2% better than PROP, 7.8% better than GFM, 9.9% better than Optimized KLFM, 10.0% better than GMetis, and 21.4% better than PARABOLI. If all of these algorithms are considered together, hMETIS-EE$_{20}$ is still better by 0.3%. Comparing hMETIS-EE$_{20}$ with hMETIS-FM$_{20}$, we see that hMETIS-FM$_{20}$ is about 1.1% better than hMETIS-EE$_{20}$, and about 1.4% better than all of the previous schemes combined. In particular, hMETIS-FM$_{20}$ was able to improve the best-known bisections for eight out of the 23 test circuits.

Looking at the quality of the partitionings produced by the two schemes that use the multilevel hypergraph refinement (vV-cycles), we see that these schemes are able to produce very good results. In particular, hMETIS-FM$_{20vV}$ is

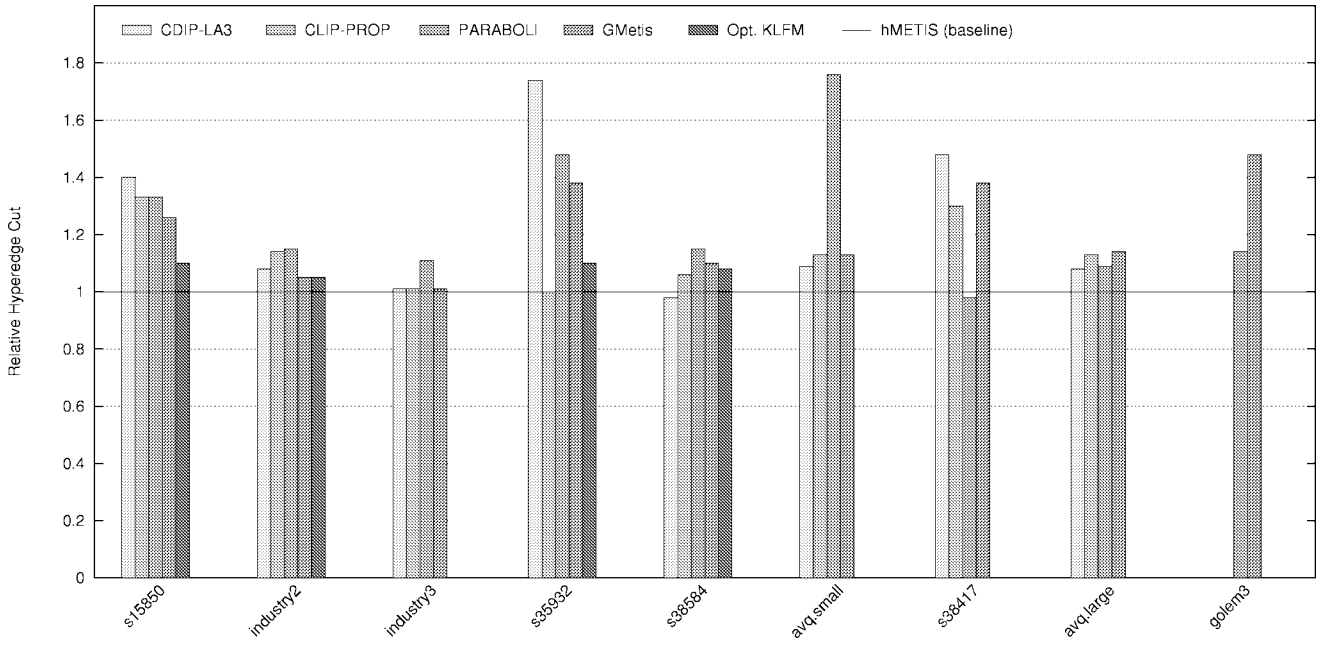


Fig. 4. The relative performance of $\texttt{hMETIS-FM}_{20vV}$ compared to rest of the schemes on the large benchmarks (with 10 K or more nodes).

about 2.0% better than $\texttt{hMETIS-EE}_{20}$ and 0.9% better that $\texttt{hMETIS-FM}_{20}$. $\texttt{hMETIS-FM}_{20vV}$ seems to be the overall best scheme, producing partitionings whose quality is better than any of the previous schemes and 2.3% better that the "Best."

The last sub-table of Table II shows the total amount of time required by the various partitioning algorithms. These run times are in seconds on the respective architectures. Because of the difference in central processing unit (CPU) speed at the various machines, it is hard to make direct comparisons. However, we tested our code on Sparc5 and we found that it requires about four times more time than when it is running on R10000. Taking into consideration a scaling factor of four, we see that both $\texttt{hMETIS-EE}_{20}$ and $\texttt{hMETIS-FM}_{20}$ require less time than either PROP, $\mathrm{CDIP-LA}_f$, $\mathrm{CLIP-PROP}_f$, PARABOLI, or GFM. In particular, $\texttt{hMETIS-EE}_{20}$ is about four times faster than PROP, nine times faster than $\mathrm{CDIP-LA}_f$ and $\mathrm{CLIP-PROP}_f$, and much faster than PARABOLI, GFM and Optimized KLFM. Compared to GMetis, we see that $\texttt{hMETIS-EE}_{20}$ requires roughly the same time, whereas $\texttt{hMETIS-FM}_{20}$ is about twice as slow. Note that GMetis runs `METIS` 100 times on each graph, but each of these runs is substantially faster than `hMETIS`, partly because `METIS` is a highly optimized code for graphs, and partly because coarsening and refinement on hypergraphs is more complex than the refinement schemes used in `METIS` for graphs. However, both $\texttt{hMETIS-EE}_{20}$ and $\texttt{hMETIS-FM}_{20}$ produce bisections that cut substantially fewer hyperedges than GMetis.

Looking at the amount of time required by $\texttt{hMETIS-EE}_{10vV}$ and $\texttt{hMETIS-FM}_{20vV}$, we see that, by using multiphase refinement, we were in general able to further reduce the amount of time required by our partitioning algorithms. In particular, $\texttt{hMETIS-EE}_{10vV}$ requires only 409 s to partition all 23 circuits, whereas $\texttt{hMETIS-FM}_{20vV}$ requires 1513 s.

V. Conclusions and Future Work

As the experiments in Section IV show, the multilevel paradigm is very successful in producing high-quality hypergraph partitionings in a relatively small amount of time. The multilevel paradigm is successful for the following reasons. The coarsening phase is able to generate a sequence of hypergraphs that are good approximations of the original hypergraph. The initial partitioning algorithm is then able to find a good partitioning by essentially exploiting global information of the original hypergraph. Finally, the iterative refinement at each uncoarsening level is able to significantly improve the partitioning quality because it moves successively smaller subsets of vertices between the two partitions. Thus, in the multilevel paradigm, a good coarsening scheme results in a coarse graph that provides a global view that permits computations of a good initial partitioning, and the iterative refinement performed during the uncoarsening phase provides a local view to further improve the quality of the partitioning.

The multilevel hypergraph-partitioning algorithm presented here is quite fast and robust. Even a single run of the algorithm is able to find reasonably good bisections. With a small number of runs (e.g., 20), our algorithm is able to find better bisections than those found by all previously known algorithms for many of the well-known benchmarks.

Our algorithm scales quite well for large hypergraphs. Due to the multilevel paradigm, the number of runs required to obtain high-quality bisections does not increase as the size of the hypergraph increases. High-quality bisections of hypergraphs with over 100 000 vertices are obtained in a few minutes on today's workstations. Also, since the coarsening phase runs in time proportional to the size of the hypergraph, the run time of the scheme increases linearly with hypergraph size. Furthermore, the scheme appears to be more powerful relative to the other schemes for larger hypergraphs (refer to Fig. 4). Restricting our comparisons to

only the larger hypergraphs (with 10 K or more nodes) in the benchmark set, we find that hMETIS-FM$_{20vV}$ performs 29.5%, 15.8%, 11.3%, 20.4%, 14.6%, 16.3%, and 8.4% better than PROP, CDIP $-$ LA$_f$, CLIP $-$ PROP$_f$, PARABOLI, GFM, GMetis, and Optimized KLFM, respectively. Note that the hypergraph-based multilevel scheme, as presented in this paper, significantly outperforms the graph-based multilevel scheme GMetis [25] that used METIS [21] to compute bisections of graph approximations of a hypergraph. The reasons for this performance difference are as follows. First, hypergraph-based coarsening causes a much greater reduction of the exposed hyperedge weight of the coarsest level hypergraph and, thus, provides much better initial partitions than those obtained with edge-based coarsening. Second, the refinement in the hypergraph-based multilevel scheme directly minimizes the size of the hyperedge cut rather than the edge cut of the inaccurate graph approximation of the hypergraph. The power of hMETIS over GMetis is much more visible on the largest benchmark golem3, on which even the best of 100 different runs produced a cut that is 50% worse than ten runs of hMETIS-FM$_{20vV}$. hMETIS also significantly outperforms Optimized KLFM [20] by Hauck and Borriello even though they used powerful refinement schemes (FM with LA$_3$ [9]). This is primarily due to the more powerful HEC schemes used in hMETIS.

It may be possible to improve the quality of the bisection produced by this algorithm in many ways. Further research may identify better coarsening schemes that are suitable for a wider class of hypergraphs. New powerful variants of the FM refinement schemes have been developed recently by Dutt *et al*. [11], [12]. It will be instructive to include such a refinement scheme during the uncoarsening phase to see if it makes the multilevel scheme more robust. However, it is unclear if the added cost of these more powerful refinement schemes will result in a cost-effective improvement in the size of the bisection because additional trials of the multilevel scheme could potentially improve the bisection.

ACKNOWLEDGMENT

Access to computing facilities was provided by AHPCRC and the Minnesota Supercomputer Institute. The algorithms described in this paper are part of the hMETIS hypergraph-partitioning package available via the World Wide Web at URL: http://www.cs.umn.edu/~metis.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning," *Integr. VLSI J.*, vol. 19, no. 1–2, pp. 1–81, 1995.
[2] S. Shekhar and D. R. Liu, "Partitioning similarity graphs: A framework for declustering problems," *Inf. Syst. J.*, vol. 21, no. 4, pp. 475–496, 1996.
[3] B. Mobasher, N. Jain, E. H. Han, and J. Srivastava, "Web mining: Pattern discovery from world wide web transactions," Dept. Comput. Sci., Univ. Minnesota, Minneapolis, MN, Tech. Rep. TR-96-050, 1996.
[4] C. Berge, *Graphs and Hypergraphs.* Amsterdam, The Netherlands: Elsevier, 1976.
[5] M. R. Garey and D. S. Johnson, *Computers and Instractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman, 1979.
[6] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. ACM/IEEE Design Automation Conf.,* 1972, pp. 57–62.
[7] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
[8] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. 19th IEEE Design Automation Conf.,* 1982, pp. 175–181.
[9] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438–446, May 1984.
[10] Y. Saab, "A fast and robust network bisection algorithm," *IEEE Trans. Comput.*, vol. 44, pp. 903–913, July 1995.
[11] S. Dutt and W. Deng, "A probability-based approach to VLSI circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.,* 1996.
[12] _____, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. Physical Design Workshop,* 1996.
[13] T. Bui *et al*., "Improving the performance of the Kernighan–Lin and simulated annealing graph bisection algorithm," in *Proc. ACM/IEEE Design Automation Conf.,* 1989, pp. 775–778.
[14] L. Hagen and A. Kahng, "A new approach to effective circuit clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design,* 1992, pp. 422–427.
[15] H. Shin and C. Kim, "A simple yet effective technique for partitioning," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 380–386, Sept. 1993.
[16] C. J. Alpert, L. W. Hagen, and A. B. Kahng, "A general framework for vertex orderings, with applications to netlist clustering," *IEEE Trans. VLSI Syst.*, vol. 4, pp. 240–246, June 1996.
[17] T. Bui and C. Jones, "A heuristic for reducing fill in sparse matrix factorization," in *6th SIAM Conf. Parallel Processing Sci. Computing,* 1993, pp. 445–452.
[18] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Sandia Nat. Labs., Tech. Rep. SAND93-1301, 1993.
[19] J. Cong and M. L. Smith, "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design," in *Proc. ACM/IEEE Design Automation Conf.,* 1993, pp. 755–760.
[20] S. Hauck and G. Borriello, "An evaluation of bipartitioning technique," in *Proc. Chapel Hill Conf. Advanced Res. VLSI,* 1995.
[21] G. Karypis and V. Kumar, "METIS 3.0: Unstructured graph partitioning and sparse matrix ordering system," Dept. Computer Sci., Univ. Minnesota, Tech. Rep. 97-061, 1997.
[22] G. Karypis and V. Kumar, "A fast and highly quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, to be published.
[23] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Analysis Applicat.*, vol. 11, no. 3, pp. 430–452, 1990.
[24] S. T. Barnard and H. D. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," in *Proc. 6th SIAM Conf. Parallel Processing Sci. Computing,* 1993, pp. 711–718.
[25] C. Alpert and A. Kahng, "A hybrid multilevel/genetic approach for circuit partitioning," in *Proc. 5th ACM/SIGDA Physical Design Workshop,* 1996, pp. 100–105.
[26] B. M. Riess, K. Doll, and F. M. Johannes, "Partitioning very large circuits using analytical placement techniques," in *Proc. ACM/IEEE Design Automation Conf.,* 1994, pp. 646–651.
[27] T. Lengauer, *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart and Winston, 1976.
[28] E. Ihler, D. Wagner, and F. Wagner, "Modeling hypergraphs by graphs with the same mincut properties," *Info. Process. Lett.*, vol. 45, no. 4, pp. 171–175, Mar. 1993.
[29] J. Li, J. Lillis, and C. K. Cheng, "Linear decomposition algorithm for VLSI design applications," in *Proc. IEEE Int. Conf. Computer-Aided Design,* 1995, pp. 223–228.
[30] F. Brglez, "ACM/SIGDA design automation benchmarks: Catalyst or anathema?," *IEEE Design & Test,* vol. 10, no. 3, pp. 87–91, 1993.
[31] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization.* Englewood Cliffs, NJ: Prentice-Hall, 1982.
[32] J. Li, J. Lillis, and C. Cheng, "Linear decomposition algorithm for VLSI design applications," in *Proc. IEEE Int. Conf. Computer-Aided Design,* 1995, pp. 223–228.

George Karypis received the Ph.D. degree in computer science from the University of Minnesota, Minneapolis.

He is currently an Assistant Professor in the Department of Computer Science and Engineering, University of Minnesota. He has co-authored several journal articles and conference papers on these topics and *Introduction to Parallel Computing* (Reading, MA: Addison-Wesley, 1994). His current research interests spans the areas of parallel algorithm design, applications of parallel processing in scientific computing and optimization, sparse matrix computations, and data mining. His research has resulted in the development of software libraries for serial and parallel unstructured graph partitioning (METIS and ParMETIS), and for parallel Cholesky factorization (PSPASES).

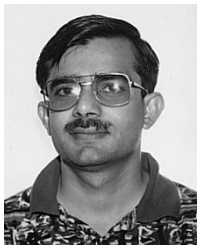
Rajat Aggarwal received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi, India, in 1995, and the M.Sc. degree in computer science from the University of Minnesota, Minneapolis, MN, in 1997.

He is currently with the Lattice Semiconductor Corporation, Milpitas, CA, where he is involved in the development of logic optimization, mapping, and placement algorithms for the CPLD's and FPGA's.

Vipin Kumar (S'78–M'82–SM'91) received the Ph.D. degree in computer science from the University of Maryland at College Park.

He is currently a Professor in the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN. His current research interests include parallel computing, parallel algorithms for scientific computing problems, and data mining. His research has resulted in the development of highly efficient parallel algorithms and software for sparse matrix factorization (PSPASES), graph partitioning, (METIS and ParMETIS), and dense hierarchical solvers. His research in performance analysis resulted in the development of the isoefficiency metric for analyzing the scalability of parallel algorithms. He has authored over 100 research papers and coauthored *Introduction to Parallel Computing* (Reading, MA: Addison-Wesley, 1994). He has presented over 50 invited talks at various conferences, workshops, national labs, and has served as chair/co-chair for many conferences/workshops in the area of parallel computing and high-performance data mining. He serves on the editorial boards of *Parallel Computing* and the *Journal of Parallel and Distributed Computing*.

Dr. Kumar is a member of the Society of Industrial and Applied Mathematics (SIAM) and the Association for Computing Machinery (ACM). He serves on the editorial board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He has also served on the editorial board of IEEE TRANSACTIONS OF DATA AND KNOWLEDGE ENGINEERING (1993–1997).



Shashi Shekhar (S'86–M'89–SM'96) received the B.Tech. degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1985, and the M.S. degree in business administration and the Ph.D. degree in computer science from the University of California at Berkeley, Berkeley, CA, in 1989.

He is currently an Associate Professor in the Department of Computer Science and Engineering, and an active member of the Army High Performance Computing Research Center, as well as the Center for Transportation Studies, University of Minnesota, Minneapolis, MN. His research interests include databases, geographic information systems (GIS's), and intelligent transportation systems. He has published over 100 research papers in refereed journals, conferences, workshops, and edited books. He was program co-chair of the 1996 ACM International Workshop on Advances in GIS.

Dr. Shekhar is a senior member of the IEEE Computer Society, and a member of the ACM and AAAI. He is an editorial board member of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and of the IEEE Computer Society Computer Science and Engineering Practice Board.