A short version of this paper appears in the 34th Design Automation Conference

Multilevel Hypergraph Partitioning:
Applications in VLSI Domain*

George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar

University of Minnesota, Department of Computer Science, Minneapolis, MN 55455

{karypis, rajat, kumar, shekhar}@cs.umn.edu

Last updated on March 27, 1998 at 5:12pm

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 the performance both in terms of the size of the hyperedge cut on the bisection as well as run time on a number of VLSI circuits. Our experiments show that our multilevel hypergraph partitioning algorithm produces high quality partitioning in relatively small amount of time. The quality of the partitionings produced by our scheme are on the average 6% to 23% better than those produced by other state-of-the-art schemes. Furthermore, our partitioning algorithm is significantly faster, often requiring 4 to 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% to 30%).

1 Introduction

Hypergraph partitioning is an important problem and has extensive application to many areas, including VLSI design [16], efficient storage of large databases on disks [26], and data mining [25]. The problem is to partition the vertices of a hypergraph in 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 [29], and the size a hyperedge is the cardinality of this subset.

*This work was supported by IBM Partnership Award, NSF CCR-9423082, Army Research Office contract DA/DAAH04-95-1-0538, and Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to computing facilities was provided by AHPCRC, and the Minnesota Supercomputer Institute. Related papers are available via WWW at URL: **http://www.cs.umn.edu/˜karypis**

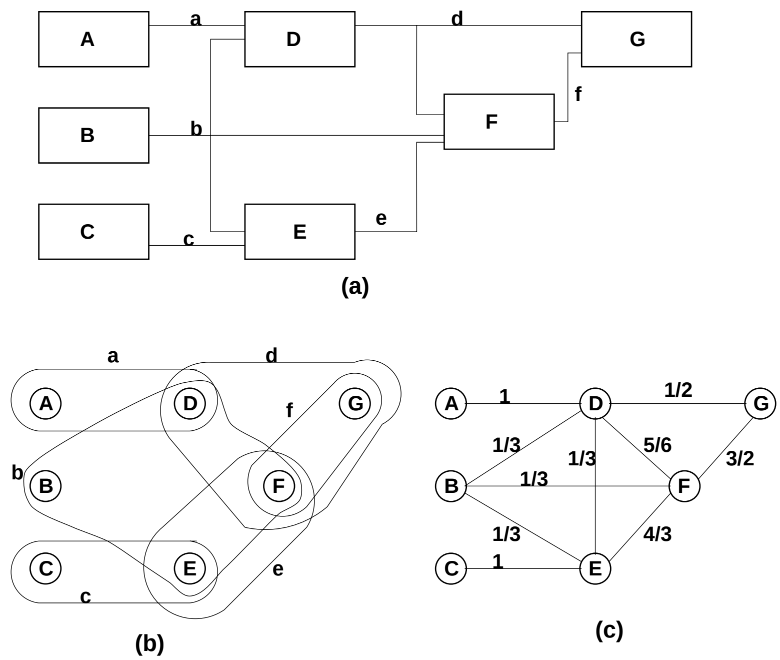


Figure 1: (a) Circuit showing cells and nets, (b) hypergraph representation of the circuit, (c) weighted clique modeling of hypergraph.

During the course of VLSI circuit design and synthesis, it is quite 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 FPGAs. Such partitioning instances are challenging because the timing, area, and 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 pins 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 FPGAs. 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 (as illustrated in Figure 1). A high quality hypergraph partitioning algorithm greatly affects the feasibility, quality, and cost of the resulting system.

Efficient storage of large databases requires information that are accessed together by individual queries to be stored on a small number of disk blocks. This significantly improves the performance of database operations when disk access time is the bottleneck. By clustering related information we can minimize the number of disk accesses. Graph partitioning is an effective method for database clustering [28] but the performance can be further improved by using hypergraph partitioning. In particular, the database is modeled as hypergraph, in which the various items stored in the database (*i.e.*, records) represent the vertices, and records that are accessed together by single queries are connected via hyperedges. This hypergraph is then partitioned into parts, so that the size of each part is smaller than the size of the disk sector, and the number of hyperedges that connect records in different disk-sectors is minimized. Other applications include clustering as well as the partitioning of the roadmap database for routing applications and declustering data in parallel databases [26].

1.1 Related Work

The problem of computing an optimal bisection of a hypergraph is at least NP-hard [30]. However, because of the importance of the problem in many application areas, many heuristic algorithms have been developed. The survey by Alpert and Khang [16] provides a detailed description and comparison of various such 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 [2] (an extension of the Kernighan-Lin (KL) heuristic [1] for hypergraphs), or the faster Fiduccia-Mattheyses (FM) [3] 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 results in 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, for a number of reasons. First, these methods choose vertices for movement based only upon local information. For example, it may be better to move a vertex with smaller gain, as it may allow many good moves later. Second, if many vertices have the same gain, then the method offers no insight on which of these vertices to move [4]. Third, a hyperedge that has more than one vertices on both sides of the partition line does not influence the computation of the gain of vertices contained in it, making the gain computation quite inexact [22]. Hence, these algorithms have been extended in a number of ways [4, 20, 22, 23].

Krishnamurthy [4] 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 moves. PROP [22], introduced by Dutt and Deng, used a probabilistic gain computation model for deciding the vertices that needs to move across the partition line. It tries to capture the implications of moving a node across the partition boundary for a hyperedge that contains many vertices on both the sides of the partition. These schemes tend to enhance the performance of the basic KLFM familly of refinement algorithms, at the expense of increased run time. Dutt and Deng [23] proposed two new methods, namely CLIP and CDIP, for computing 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 [5, 7, 13, 24] 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, they 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 [13].

Recently a new class of partitioning algorithms was developed [10, 12, 11, 17] that are 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 Figure 2. The iterative refinement schemes such as KLFM become quite powerful in this multilevel context for the following reason. First, movement of a single node across partition boundary in a coarse graph can lead to 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 [10] in the context of computing fill reducing matrix reordering, by Hendrickson and Leland [12] in the context of finite element grid partitioning, and by Hauck and Borriello [17] (called Optimized KLFM) and by Cong and Smith [11] for hypergraph partitioning. Karypis and Kumar extensively studied this paradigm in [27, 33] for 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 without compromising the overall quality. METIS [27], 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 [6, 8] for graphs.

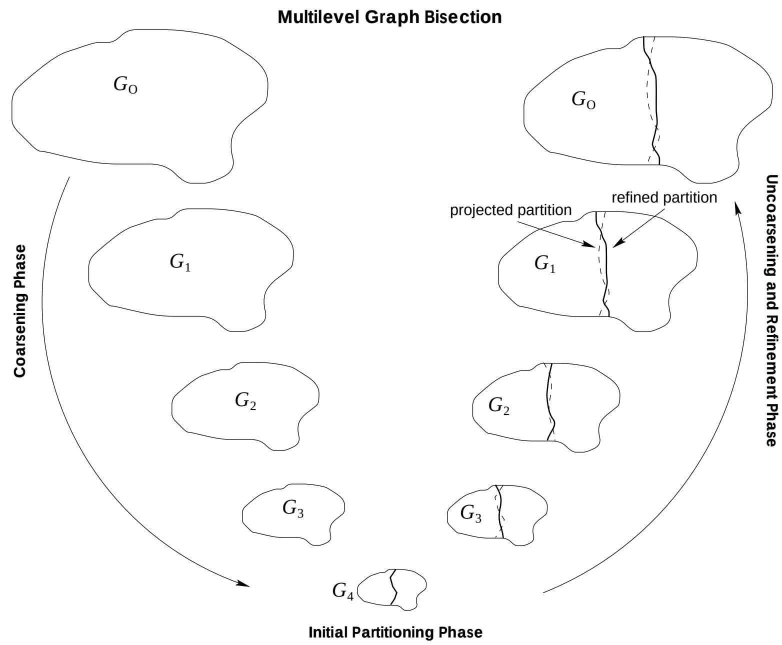


Figure 2: The various phases of the multilevel graph bisection. During the coarsening phase, the size of the graph is successively decreased; during the initial partitioning phase, a bisection of the smaller graph is computed; and during the uncoarsening and refinement phase, the bisection is successively refined as it is projected to the larger graphs. During the uncoarsening and refinement phase the dashed lines indicate projected partitionings, and dark solid indicate partitionings that were produced after refinement. G_0 is the given graph, which is the finest graph. G_{i+1} is next level coarser graph of G_i, vice versa, G_i is next level finer graph of G_{i+1}. G_4 is the coarsest graph.

The improved coarsening schemes of METIS work only for graphs, and are not directly applicable to hypergraphs. If the hypergraph is first converted into a graph (by replacing each hyperedge by a set of regular edges), then METIS [27] can be used to compute a partitioning of this graph. This technique was investigated by Alpert and Khang [21] in their algorithm called GMetis. They converted hypergraphs to graphs by simply replacing each hyperedge by a clique, and then dropped many edges from each clique randomly. They used METIS to compute a partitioning of each such random graph and 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 [15], PROP [22], and the multilevel hypergraph partitioner from Hauck and Borriello [17].

The conversion of a hypergraph into a graph by replacing each hyperedge by 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. This is illustrated in Figure 3. Figure 3(a) shows the original hyperedge and Figure 3(b) shows the graph obtained after replacing the hyperedge by its clique. The standard hyperedge to edge conversion [31] assigns a uniform weight of $1/(|e| - 1)$ to each edge in the clique, where $|e|$ is the ***size*** of the hyperegde *i.e.*, the number of vertices in the hyperedge. Thus, in our example, each edge is assigned a weight of 1/3. Figures 3(c) and 3(d) show two example bisections in which the edge-cuts are 1 and 4/3, respectively. So different partitionings of the converted graph

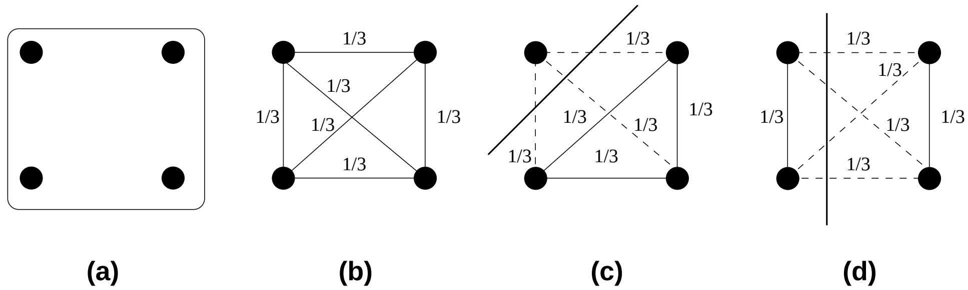


Figure 3: (a) original hypergraph, (b) graph obtained after clique conversion in which weight of each edge = 1/3, (c) an example partitioning with weight of hyperedges cut down =1, and (d) an example partitioning with weight of hyperedges cut down= 4/3

lead to different cuts in the graph, whereas for the original hypergraph both of these partitionings result in a cut-size of one. 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 [14]. This hinders the partitioning refinement algorithm since vertices are moved between partitions depending on the reduction in the number of edges they cut in the converted graph, whereas the real objective is to minimize the number of hyperedges that are 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 [21] 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 [17] operates directly on hypergraphs, and thus is able to perform accurate refinement during the uncoarsening phase. However, all coarsening schemes studied in [17] 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 [21].

1.2 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 to be simplified considerably (even beyond the plain FM refinement), making the multilevel scheme quite fast. We investigate various algorithms for the coarsening and uncoarsening phases which operates on the hypergraphs without converting them into graphs. We have also developed new multi-phase 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 multi-phase schemes further reduce the run-times as well as improve the solution quality. Figure 4 shows the general framework of our algorithm hMETIS, using both v- and V-cycles.

We evaluate the performance both in terms of the size of the hyperedge cut on the bisection as well as run time on a number of VLSI circuits. Our experiments show that our multilevel hypergraph partitioning algorithm produces high quality partitioning in relatively small amount of time. The quality of the partitionings produced by our scheme are on the average 6% to 23% better than those produced by other state-of-the-art schemes [15, 19, 21, 22, 23]. The difference in quality over other schemes becomes even greater for larger hypergraphs. Furthermore, our partitioning algorithm is significantly faster, often requiring 4 to 10 times less time than that required by the other schemes. For many circuits in the well known ACM/SIGDA benchmark set [9], our scheme is able to find better partitionings than those reported in the literature for any other hypergraph partitioning algorithm.

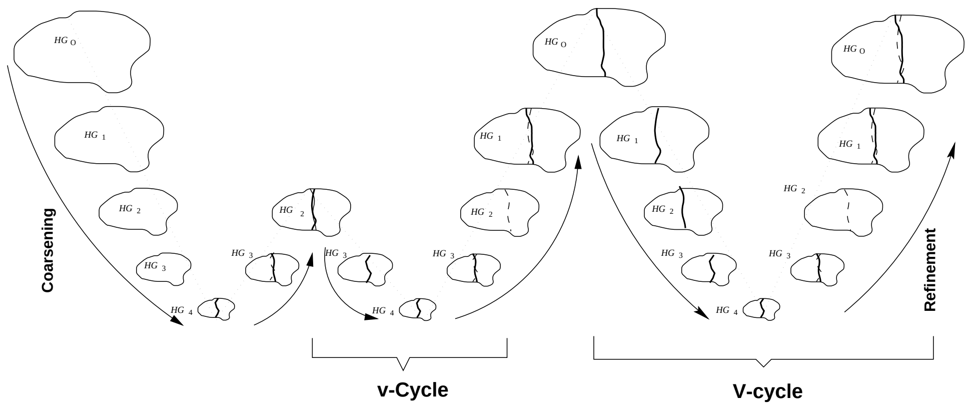


Figure 4: Multilevel Hypergraph Partitioning with one embedded v-cycles followed by a V-cycle.

The rest of this paper is organized as follows. Section 2 describes the different algorithms used in the three phases of our multilevel hypergraph partitioning algorithm. A comprehensive experimental evaluation of these algorithms is provided in Section 3. Section 4 describes and experimentally evaluates a new partitioning refinement algorithm based on the multilevel paradigm. Section 5 compares the results produced by our algorithm to those produced by earlier hypergraph partitioning algorithms.

2 Multilevel Hypergraph Bisection

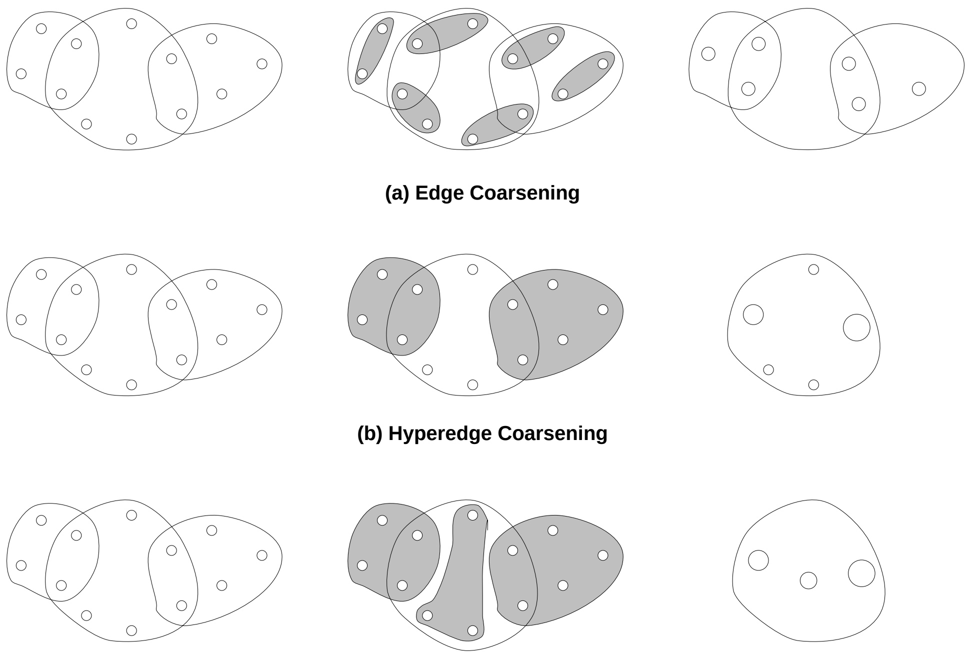
We now present the framework of hMETIS, in which the coarsening and the refinement scheme work directly with hyperedges without using the clique representation to transform them into edges. We have developed new algorithms for both the phases, which in conjunction have the capability of delivering very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

2.1 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 connecting just a few vertices. This is particularly helpful, since refinement heuristics based on the KLFM familly of algorithms [1, 2, 3] 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 merge them together, as illustrated in Figure 5(a). A second possibility is to merge together all the vertices that belong to a hyperedge as illustrated in Figure 5(b). Finally, a third possibility is to merge together a subset of the vertices belonging to a hyperedge as illustrated in Figure 5(c). These three different schemes for grouping vertices together for contraction are described below.

Edge Coarsening The heavy-edge matching scheme used in the multilevel graph bisection algorithm can also be used to obtain successive coarser hypergraphs by merging the pairs of vertices connected by many hyperedges. In



(a) Edge Coarsening

(b) Hyperedge Coarsening

(c) Modified Hyperedge Coarsening

Figure 5: Various ways of matching the vertices in the hypergraph and the coarsening they induce. (a) In the edge-coarsening connected pairs of vertices are matched together. (b) In the hyperedge-coarsening all the vertices belonging to a hyperedge are matched together. (c) In the modified hyperedge coarsening, we match together both all the vertices in a hyperedge as well as groups of vertices belonging to a hyperedge.

this ***edge coarsening*** (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 edge coarsening scheme is similar in nature to the schemes that treat the hypergraph as a graph by replacing each hyperedge with its clique representation [31]. However, this hypergraph to graph conversion is done implicitly during matching without forming the actual graph.

Hyperedge Coarsening Even though the edge coarsening scheme is able to produce successively coarse hypergraphs, it decreases the hyperedge weight of the coarse graph only for those pairs of matched vertices that are connected via a hyperedge of size two. As a result, the total hyperedge weight of successive coarse 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 coarse graph, each such group of vertices must be connected by a hyperedge.

This is the motivation behind the ***hyperedge coarsening*** (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 non-increasing hyperedge-weight order and the hyperedges of the same weight are sorted in a non-decreasing hyperedge size order. Then, the hyperedges are visited in that order, and for each hyperedge that connects vertices that have not yet been matched, they are matched together. Thus, this scheme

[1] One can also compute a maximum weight matching [32]; however that would have significantly increased the amount of time required by this phase.

gives preference to the hyperedges that have large weight and those that are of small size. After all hyperedges have been visited, the groups of vertices that have been matched are contracted together to form the next level coarse graph. The vertices that are not part of any contracted hyperedges, they are simply copied to the next level coarse graph.

Modified Hyperedge Coarsening The hyperedge coarsening algorithm is able to significantly reduce the amount of hyperedge weight that is left exposed in successive coarse graphs. However, during each coarsening phase, a large 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 successive coarse graphs become significantly different, which distorts the shape of the contracted hypergraph.

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

2.2 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 vertices) 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 [33] until half of the vertices are in this region. Then the vertices belonging to the grown region are 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 2.3. However, the partitioning of the coarsest hypergraph that has the smallest cut may not necessarily be the one that will lead to the smaller cut in the original hypergraph. It is possible that another partitioning of the coarsest hypergraph (with higher cut) leads to a better 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 runtime. One way to dampen the increase in runtime 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 with 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 coarests 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 less accurate reflection of the size of the cut of the original finest level hypergraph. Second, the 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 reported in this paper, we find 10 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 that to improve the overall partitioning quality). In our experiments we have seen that by keeping 10 partitionings, we can reduce the cut on the average by 3% to 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 vaue 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.

2.3 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 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 quality.

We have implemented two different partitioning refinement algorithms. The first is the FM algorithm [3] which repeatedly moves vertices between partitions in order to improve the cut. The second algorithm, called HER, moves groups of vertices between partitions so that an entire hyperedge is removed from the cut. These algorithms are further described in the remaining of this section.

Fiduccia-Mattheyses (FM) The partitioning refinement algorithm by Fiduccia and Mattheyses [3] 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 by moving them to other partitions, improves the quality of the partitioning (*i.e.*, the number of hyperedges being cut decreases) and the balance constraint is not violated. 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*, that is, 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 $O(|E^h|)$ [3].

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

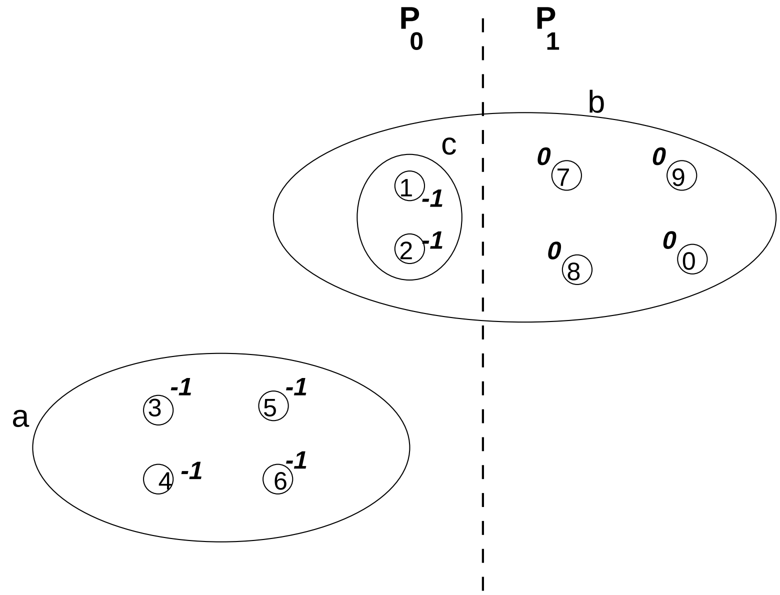


Figure 6: Initial partitioning of the given hypergraph with the gains of each vertex. Required balance factor is 40/60.

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 runtime 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-out of a local minima and reach to a better cut. However, in the context of a multilevel scheme, 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 3).

Hyperedge Refinement (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. For example, Figure 6 shows an initial partitioning of the given hypergraph. The gain for each node is either 0 or -1. There is no sequence of moves which FM can select to reduce the size of the cutset (FM always select a vertex with highest gain first to move). Whereas, if we move all the vertices of hyperedge b from partition P_0 to P_1, we can save one hyperedge without affecting the required balance factor of 40/60. We have developed such a refinement algorithm that focuses on the hyperedges that straddle partitioning boundary and tries to move them so that they are interior to either one of the partitions.

Our ***hyperedge refinement*** (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, that 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.

Note that unlike FM, HER is not a hill-climbing algorithm, as it does not perform moves that can lead to a larger cut (*i.e.*, negative cut reduction). Hence, this algorithm lacks the capability of being able to climb-out of local minima by allowing moves that increase the cut. It is possible to develop an FM-style refinement algorithm that moves entire hyperedges. This algorithm will maintain the gain of each hyperedge, and in each iteration, will move the hyperedge that leads to the highest gain. However, the computation of updating gain for each hyperedge is much more expensive than the computation of updating gain for each vertex in the original FM algorithm. The reason is that every time we move a subset of vertices we need to update the gains of not only the hyperedges that are incident on the hyperedge moved, but also of all the hyperedges that are incident on the incident hyperedges. On the other hand, our HER scheme does not need to perform such computations to update the gains, since it does not prioritize the moves according to the cut-reduction they perform. This considerably reduces the runtime of the HER algorithm.

In general, the partitioning obtained by iterative HER refinement can be further improved by FM refinement. The reason is that HER forces movement of an entire group of vertices that belongs to a hyperedge, whereas FM refinement allows movements of individual vertices across partitioning boundary. On our experiments (see Section 3) we found that the final partitioning obtained by HER refinement can be substantially improved by a round of FM refinement.

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

Table 1: The characteristics of the various hypergraphs used to evaluate the multilevel hypergraph partitioning algorithms.

3 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 [9]. The characteristics of these hypergraphs are shown in Table 1. We performed all our experiments on an SGI Challenge that has MIPS R10000 processors running at 200Mhz, and all reported run-times are in seconds. All the reported partitioning results were obtained by forcing a 45–55 balance condition.

As discussed in Sections 2.1, 2.2, and 2.3, there are many alternatives for each of the three different phases of a multilevel algorithm. It is not possible to provide an exhaustive comparison of all these possible combinations without making this paper unduly large. Instead, we provide comparisons of different alternatives for each phase after making a reasonable choice for the other two phases.

3.1 Coarsening Schemes

Table 2 shows the quality of the partitionings produced by the three coarsening schemes, edge-coarsening (EC), hyperedge coarsening (HEC), and modified hyperedge coarsening (MHEC) for all the hypergraphs in our experimental testbed. These results are the best of ten different runs using ten different random initial partitionings and FM during refinement. The column labeled "Best" in Table 2 shows the minimum of the three cuts produced by EC, HEC, and MHEC. To compare the relative performance of the three coarsening schemes, we computed the percentage by which each scheme performs worse than the "Best". We will refer to these percentages as QRBs (Quality Relative to the Best). These results are shown in the last three columns of Table 2. Furthermore, for each coarsening scheme we also computed the average QRB over all 23 test hypergraphs, and these averages are shown in the last row of the table.

	Number of Hyperedge Cut				Quality Relative to Best		
Circuit	EC	HEC	MHEC	Best	EC	HEC	MHEC
19ks	104	106	106	104		1.9	1.9
avq.large	130	127	138	127	2.4		8.7
avq.small	134	130	132	130	3.1		1.5
baluP	27	27	27	27			
biomedP	88	89	83	83	6.0	7.2	
bm1	51	52	51	51		2.0	
golem3	1848	1614	1445	1445	27.9	11.7	
industry2P	174	173	167	167	4.2	3.6	
industry3	260	255	254	254	2.4	0.4	
p1	50	50	51	50			2.0
p2	143	155	145	143		8.4	1.4
s9234P	41	40	40	40	2.5		
s13207P	58	55	64	55	5.5		16.4
s15850P	52	42	44	42	23.8		4.8
s35932	42	43	42	42		2.4	
s38417	57	54	51	51	11.8	5.9	
s38584	49	47	48	47	4.2		2.1
structP	35	33	33	33	6.0		
t2	92	90	88	88	4.5	2.3	
t3	58	58	59	58			1.7
t4	51	54	51	51		5.9	
t5	78	73	71	71	9.9	2.8	
t6	60	60	63	60			5.0
Agg. Cut	3682	3427	3253	3219			
Avg. QRB					4.97	2.37	1.98

Table 2: The size of the hyperedge cut produced by edge-coarsening (EC), hyperedge-coarsening (HEC), and modified hyperedge-coarsening schemes (MHEC). The column label "Best" shows the minimum of the cuts produced by EC, HEC, and MHEC. The last three columns show how much worse is a particular cut relative to the "Best". For example, for 19ks, the "Best" cut is 104; thus, the cut produced by MHEC (which is 106) is 1.9% worse than the "Best".

From the average QRBs, we see that all three coarsening schemes have similar performance. The difference between the average performance of any two of them is less than 3%, with EC performing worse than either HEC or MHEC. Another qualitative comparison that also takes into account the size of the problems can be performed by comparing the aggregate cuts over all 23 problems produced by the three coarsening schemes. These aggregates are also shown in Table 2. EC cuts a total of 3682 hyperedges whereas MHEC cuts only 3253, a 13% improvement.

Comparing the results in Table 2 with those in Table 8, we see that the overall performance of any of these three schemes is, on the average, better than any of the previously known state-of-the-art algorithms [15, 21, 22, 23] for hypergraph partitioning. This demonstrates the power and robustness of the multilevel paradigm for hypergraph partitioning. However, looking at the individual problems, we see that performance of the three schemes differs significantly on some of the problems. This is partly due to the random nature of the coarsening process in any of these schemes, and partly due to the suitability of some coarsening scheme for specific hypergraph structures present in some of these benchmarks. Another very interesting trend visible in Table 2 is the striking complementarity of the HEC and MHEC. On nearly every benchmark either both of these schemes or one of them has the best performance. It is clear that an algorithm that runs both of these schemes, and chooses the best cut, would be a very robust and high quality hypergraph partitioning algorithm.

The results in Table 2 also show that the EC coarsening scheme generally performs worse than the other two schemes. The reason is that, in general, EC does not produce as much reduction in the hyperedge-weight as the other two schemes. Hence, the size of the hyperedge cut of the initial bisection of the coarsest graph tends to be much higher for the EC coarsening scheme compared with the HEC and MHEC coarsening schemes. The quality of these initial bisections of the coarsest hypergraphs for some of the larger circuits is shown in Table 3, for each one of the three coarsening schemes. From these results we can see that MHEC leads to initial bisections that are 35% to 90% better than those produced by EC.

Circuit	EC	HEC	MHEC
avq.large	730	577	507
avq.small	818	623	599
golem3	5848	4074	3945
industry2P	871	655	565
industry3	1156	845	787
s38584	275	165	145

Table 3: The size of the hyperedge cut produced at the coarsest hypergraph by the region growing algorithm for each one of the three coarsening schemes.

3.2 Initial Partitioning Schemes

Table 4 shows the quality of the bisections produced by the random and the region growing algorithms for computing initial bisections. The results are shown for all three different coarsening schemes, and for the FM refinement scheme.

The results produced by each initial partitioning algorithm for the corresponding coarsening schemes are in general similar. Looking at the aggregate cuts, we see that random partitioning outperforms region growing for both EC and MHEC by 1.7% and 0.6%, respectively. However, for HEC, region growing outperforms random partitioning by 0.6%. These results show that there is no significant difference between the two initial partitioning algorithms, and either one performs reasonably. These results are consistent with those for multilevel graph partitioning of regular graphs [33]. In either case, the initial partitioning of a highly compressed graph has little effect on the overall quality of the final partition. The reasons are two-fold. First, the coarsening process reduces the number of exposed hyperedges in the coarsest hypergraph substantially. For example, for the coarsest hypergraph for “golem3”, the sum of exposed hyperedge weight is only 14368, whereas, the original graph has 144949 hyperedges. As a result, even a random partitioning is a reasonably good partitioning of the original hypergraph. Second, any partitioning of the coarsest hypergraph has plenty of opportunity to be refined at the different uncoarsening levels.

	Random			Region Growing		
Circuit	EC	HEC	MHEC	EC	HEC	MHEC
19ks	104	106	106	104	106	106
avq.large	130	127	138	134	128	138
avq.small	134	130	132	127	129	133
baluP	27	27	27	27	27	27
biomedP	88	89	83	83	83	83
bm1	51	52	51	52	47	52
golem3	1848	1614	1445	1927	1600	1484
industry2P	174	173	167	178	176	166
industry3	260	255	254	260	255	241
p1	50	50	51	47	51	47
p2	143	155	145	144	148	140
s9234P	41	40	40	42	40	42
s13207P	58	55	64	56	60	58
s15850P	52	42	44	50	43	49
s35932	42	43	42	41	44	42
s38417	57	54	51	55	52	52
s38584	49	47	48	48	47	48
structP	35	33	33	33	33	33
t2	92	90	88	88	91	89
t3	58	58	59	58	58	60
t4	51	54	51	50	54	50
t5	78	73	71	77	74	72
t6	60	60	63	65	60	61
Aggregate Cut	3682	3427	3253	3746	3406	3273

Table 4: The size of the hyperedge cut produced when “Random” and “Region Growing” initial partitioning algorithms are used in conjunction with the EC, HEC, and MHEC coarsening schemes. FM refinement is used in all cases.

3.3 Refinement Schemes

Table 5 shows the quality of the bisections produced by four different refinement schemes for the edge-coarsening (EC) and the modified hyperedge-coarsening (MHEC) schemes. The refinement schemes reported are: early-exit Fiduccia-Mattheyses (EE-FM), Fiduccia-Mattheyses (FM), hyperedge refinement (HER), and hyperedge-refinement followed by FM. Again, the reported bisections are the smallest out of ten different runs each using ten random initial partitionings.

Comparing EE-FM with FM we see, as expected, that FM performs consistently better than EE-FM. Looking at the aggregate cuts of all 23 hypergraphs, we see that for the EC coarsening scheme, EE-FM cuts a total of 3851 hyperedges while FM cuts only 3682, a difference of 4.6%. Similar results hold for the MHEC coarsening scheme; however, the difference is much smaller (1.2%). The reason that EE-FM does better for MHEC than for EC is that during coarsening, MHEC removes more hyperedge-weight than EC. Consequently, the bisection obtained initially is quite good and requires less refinement (see Table 3). On the other hand, the initial bisection of the coarsest hypergraph obtained by EC, is much worse and requires significantly more refinement. Since, EE-FM performs less refinement, the produced bisections are worse for EC than they are for MHEC.

Comparing FM and HER refinement schemes, we see that HER performs significantly worse than FM for all coarsening schemes. In the case of EC, HER is 14.5% worse than FM, and in the case of MHEC it is 6.7% worse. Again, the quality degradation is smaller for MHEC than it is for EC because the bisections produced by MHEC require less refinement. The quality of the bisection though, improves when HER is coupled with FM. For both EC and MHEC, the bisections produced by HER+FM tend to be better than those produced by any other refinement scheme.

Comparing the last column of Table 5 that corresponds to the minimum cut of the four refinement schemes and the two coarsening schemes, we see that the MHEC coupled with either FM or HER+FM performs very well. MHEC

	EC				HEC				MHEC				
Circuit	EE-FM	FM	HER	HER+ FM	EE-FM	FM	HER	HER FM	EE-FM	FM	HER	HER+ FM	Best
19ks	105	104	105	104	109	106	112	105	107	106	109	107	104
avq.large	143	130	139	134	129	127	144	128	147	138	144	135	127
avq.small	139	134	153	128	138	130	139	128	136	132	141	136	128
baluP	27	27	30	27	27	27	27	27	27	27	27	27	27
biomedP	108	88	101	85	85	89	84	83	83	83	83	83	83
bm1	53	51	54	51	52	52	52	52	51	51	52	51	51
golem3	1950	1848	2276	1843	1624	1614	1774	1580	1447	1445	1570	1451	1445
industry2P	174	174	183	175	179	173	203	193	174	167	189	170	167
industry3	262	260	262	260	255	255	255	255	257	254	255	241	241
p1	50	50	48	47	52	50	53	52	53	51	51	49	47
p2	149	143	149	144	156	155	156	155	148	145	146	143	143
s9234P	42	41	43	41	40	40	46	40	40	40	46	40	40
s13207P	61	58	74	62	55	55	65	55	65	64	72	62	55
s15850P	52	52	50	50	42	42	48	42	44	44	46	44	42
s35932	41	42	48	42	44	43	97	44	42	42	55	42	42
s38417	58	57	64	53	54	54	63	52	52	51	68	52	51
s38584	49	49	49	49	47	47	52	47	48	48	57	48	47
structP	35	35	34	34	33	33	33	33	33	33	33	33	33
t2	93	92	94	93	91	90	94	90	93	88	90	89	88
t3	60	58	59	58	58	58	58	58	60	59	58	59	58
t4	51	51	50	50	54	54	55	53	51	51	52	48	48
t5	78	78	79	77	73	73	73	71	71	71	74	72	71
t6	71	60	72	66	65	60	65	60	62	63	63	62	60
Aggregate Cut	3851	3682	4216	3673	3462	3427	3748	3403	3291	3253	3481	3244	3198

Table 5: The size of the hyperedge cut produced by different refinement schemes for EC and HEC coarsening schemes. The refinement schemes are: early-exit Fiduccia-Mattheyses (EE-FM), Fiduccia-Mattheyses (FM), hyperedge refinement (HER), and hyperedge refinement followed by FM (HER+FM).

with FM is only 1.7% worse than the "Best", and MHEC with HER+FM is only 1.4%.

4 Multi-Phase 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 Figure 7(a), and its two possible condensed versions (Figure 7(b) and 7(c)) with the same partitioning. The version in Figure 7(b) is obtained by selecting hyperedges a and b to be compressed in the hyperedge coarsening phase and then selecting pairs of nodes (4,5), (6,7), and (8,9) to be compressed in modified hyperedge coarsening phase. Similarly, version in Figure 7(c) is obtained by selecting hyperedge c to be compressed in the hyperedge coarsening phase and then selecting pairs of nodes (6,7) and (8,9) to be compressed in modified hyperedge coarsening phase. In the version of Figure 7(b) vertex A(4,5) can be moved from partition P_0 to P_1 to reduce the hyperedge-cuts by 1, but in Figure 7(c) no vertex can be moved to get reduce the hyperedge-cuts.

What this example shows is that in a multilevel setting a given initial partitioning of 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 than the previous coarsening phase (which is easily done given the random nature of all 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 ***multi-phase refinement*** algorithm is quite simple. It consists of two phases, namely a coars-

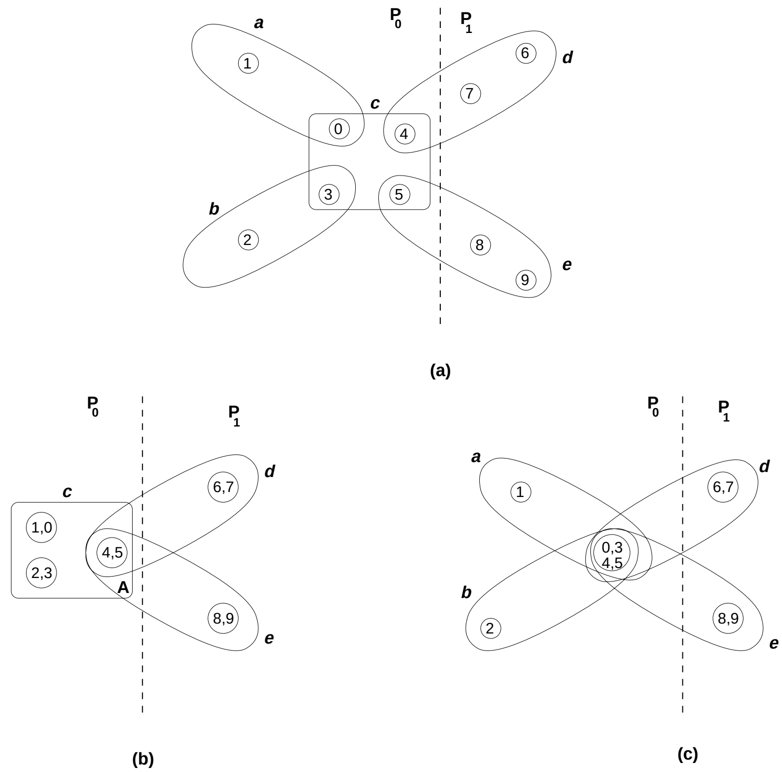


Figure 7: Effect of *Restricted coarsening*. (a) example hypergraph with a given partitioning with required balance of 40/60, (b) a possible condensed version of (a), and (c) another condensed version of hypergraph.

ening and an uncoarsening phase. The uncoarsening phase of the multi-phase refinement algorithm is identical to the uncoarsening phase of the multilevel hypergraph partitioning algorithm described in Section 2.3. 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 ***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 $i = 1, 2, \ldots, m$, 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 only to 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 2.1, namely edge-coarsening, hyperedge-coarsening, or modified hyperedge-coarsening.

Due to the randomization in the coarsening phase, successive runs of the multi-phase refinement algorithm can lead to additional reductions in the hyperedge cut. Thus the multi-phase refinement algorithm can be performed iteratively. Note that during the refinement phase, we only propagate a single partitioning; thus, multi-phase refinement is quite fast.

In the context of our multilevel hypergraph partitioning algorithm, this new multi-phase refinement can be used in a number of ways. In the rest of this section we describe three such approaches.

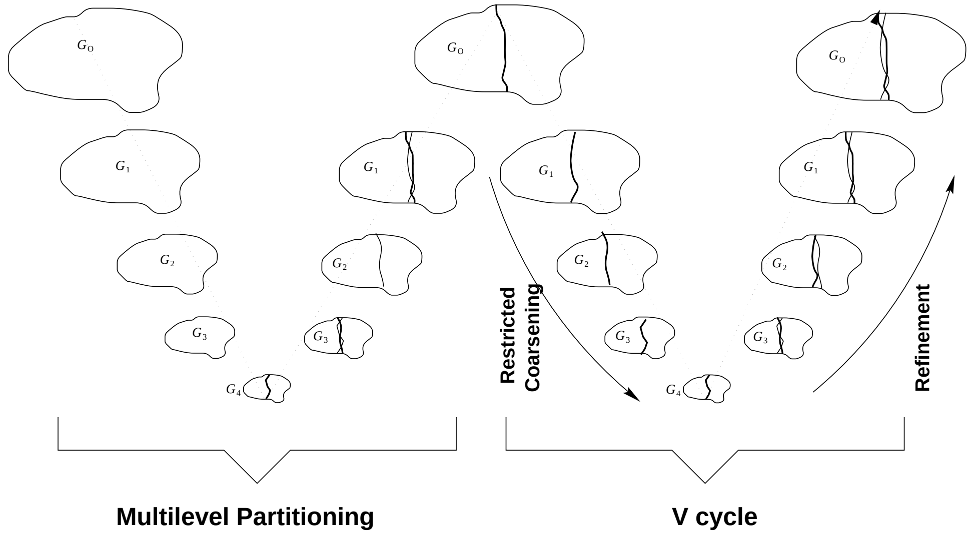


Figure 8: Multilevel Partitioning followed by a V-cycle.

V-Cycle In this scheme we take the best solution obtained from the multilevel partitioning algorithm (P_b) and we improve it using multi-phase refinement repeatedly. We stop the multi-phase refinement when the solution quality cannot be improved further. The number of multi-phase refinement steps performed is problem dependent and in general increases as the size of the hypergraph increases. This is because of the larger solution space of the large hypergraphs. Figure 8 shows multilevel partitioning followed by a V-cycle.

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 $H_{m/2}$. Then we use ($H_{m/2}$, $P_{m/2}$) as the input to multi-phase refinement. This is illustrated in Figure 9. Since $H_{m/2}$ is relatively small as compared to H_m, multi-phase 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 $H_{m/2}$.

vV-Cycle We can combine both V-cycles and v-cycles in the algorithm to obtain high quality partitioning in small amount 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.

4.1 Experimental Results with Multi-Phase Refinement Schemes

Table 6 shows the effect of the V-cycles and vV-cycles on the results of the three coarsening schemes, namely, edge-coarsening (EC), hyperedge coarsening (HEC), and modified hyperedge coarsening (MHEC). The experiments were run on all the benchmarks. For each coarsening scheme we ran two sets of experiments. In the first set we selected the best result out of the ten runs from the multilevel partitioning and used the V-cycles to further improve the partitioning. The results are shown in the Table 6 under the column “V”. In the second experiment, we ran ten runs of the multilevel partitioning with v-cycles embedded in them. The best result out of these 10 runs is passed through the V-cycles. The

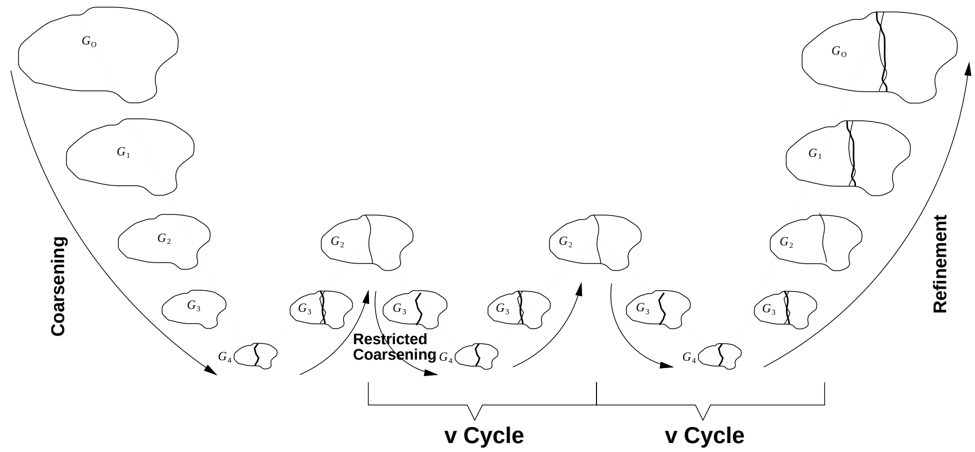


Figure 9: Multilevel Partitioning with two embedded v-cycles.

results from these experiments are shown in Table 6 under the column "vV".

The number of v-cycles and V-cycles in these experiments are determined dynamically. When there is no refinement in 3 consecutive cycles, the program exits after reporting the current result. During these experiments, we saw that the number of V-cycles performed depends on the problem size. For the large hypergraphs such as "golem3", it required about 7-8 V-cycles, whereas, for the small hypergraphs such as "bm1", it required only about 3-4 V-cycles.

For all these experiments, we used the random initial partitioning and FM refinement. The column labeled "Best" in Table 6 shows the minimum of the six cuts produced by EC, HEC, and MHEC for both V-cycles and vV-cycles. To compare relative performance of the six schemes, we computed the "quality relative to best", as was done in Section 3.1. The row labeled "Aggregate Cut" shows the total number of hyperedges cut for all the benchmarks. Comparing aggregate cuts in Tables 2 and 6, we see that the use of multi-phase refinement improves the quality of the solution. When vV-refinement is used, the total number of hyperedges being cut reduces by 2.28%, 2.07% and 1.66% for EC, HEC and MHEC coarsening schemes, respectively. At the same time, the use of vV-refinement results in reduction of overall amount of time by about 30% to 50%. Thus, the use of vV-cycles improves both the solution quality as well as the runtimes. Comparing the schemes that use V- and vV-cycles, we see that vV-cycles perform better, since it cuts fewer hyperedges and requires less time. Also note that the MHEC coarsening scheme performs better for both V- and vV-cycles. The complementarity of the performance of HEC and MHEC quite visible in Table 2, is still present in Table 6, but the spread is much smaller. This is because multi-phase refinement eliminates some of the variability inherent in the multilevel framework.

Table 7 shows the effect of the V-cycles and vV-cycles on the results of the two refinement schemes namely, Early-Exit FM (EE-FM) and FM. For each of the refinement schemes we ran two experiment with V-cycles and vV-cycles as explained above. For all the experiments, we choose MHEC as the coarsening scheme and random initial partitioning. Quality relative to best is also reported, where "Best" is the best of the 4 schemes using the combinations of EE-FM and FM with V-cycles and vV-cycles. By looking at the "Aggregate Cut" we see that FM produces partitioning that cut fewer hyperedges. In particular, FM cuts 2.49% and 0.81% fewer hyperedges as compared to EE-FM for V-cycles and vV-cycles respectively. However, EE-FM requires half the amount of time compared to that required by FM. Note also that in the case of EE-FM there is no difference between the runtimes of vV-cycles and V-cycles. This is because EE-FM is very fast which reduces the total time required to refine multiple solutions all the way to the top when no v-cycles are employed. Hence, the increase in runtime due to the v-cycles offsets the reduction due to the propagation of only one partitioning. In general, EE-FM with vV-cycles is a very good choice when runtime is the major consideration.

	Number of Hyperedge Cut							Quality Relative to the Best					
	EC		HEC		MHEC			EC		HEC		MHEC	
Circuit	V	vV	V	vV	V	vV	Best	V	vV	V	vV	V	vV
19ks	104	104	104	109	105	105	104				4.8	1.0	1.0
avq.large	127	127	127	129	127	127	127				1.6		
avq.small	127	127	127	127	132	130	127					3.9	2.4
baluP	27	27	27	27	27	27	27						
biomedP	84	84	83	83	83	83	83	1.2	1.2				
bm1	49	49	52	52	51	51	49			6.1	6.1	4.1	4.1
golem3	1750	1801	1607	1576	1422	1424	1422	23.1	26.7	13.0	10.8		0.1
industry2P	176	179	177	175	169	168	168	4.8	6.5	5.4	4.2	0.6	
industry3	260	262	255	241	249	244	241	7.9	8.7	5.8		3.3	1.2
p1	47	47	49	49	51	51	47			4.3	4.3	8.5	8.5
p2	142	142	144	145	145	145	142			1.4	2.1	2.1	2.1
s9234P	40	40	40	40	40	40	40						
s13207P	57	62	54	53	58	58	53	7.5	17.0	1.9		9.4	9.4
s15850	46	44	42	42	41	42	41	12.2	7.3	2.4	2.4		2.4
s35932	40	40	44	44	42	42	40			10.0	10.0	5.0	5.0
s38417	50	52	53	50	53	52	50		4.0	6.0		6.0	4.0
s38584	48	48	47	48	48	48	47	2.1	2.1		2.1	2.1	2.1
structP	34	34	33	33	33	33	33	3.0	3.0				
t2	92	92	90	88	88	88	88	4.5	4.5	2.3			
t3	57	57	58	58	59	59	57			1.8	1.8	3.5	3.5
t4	49	49	54	54	48	48	48	2.1	2.1	12.5	12.5		
t5	71	71	73	73	71	71	71			2.8	2.8		
t6	60	60	60	60	63	63	60					5.0	5.0
Aggregate Cut	3537	3598	3400	3356	3205	3199	3165						
Average QRB								2.9	3.6	3.2	2.8	2.3	2.2
Total Time	1263	814	1094	731	1150	781							

Table 6: The Effect of the V-cycles and vV-cycles on the coarsening schemes

5 Comparison With Other Partitioning Algorithms

In this section, we will compare our scheme with other partitioning schemes available in the literature. We compare our complete framework with other partitioning schemes rather than the coarsening or the refinement schemes. Complete framework gives us the complete picture, unlike the comparison of only the coarsening or the refinement schemes.

To compare the performance of the bisections produced by our multilevel hypergraph bisection and multi-phase refinement algorithms, both in terms of bisection quality as well as runtime, we created Table 8. Table 8 shows the size of the hyperedge cut produced by our algorithms (hMETIS) and those reported by various previously developed hypergraph bisection algorithms. In particular, Table 8 contains results for the following algorithms: PROP [22], CDIP-LA3$_f$ and CLIP-PROP$_f$ [23], PARABOLI [15], GFM [18], GMetis [21], and Optimized KLFM (scheme by Hauck and Borriello [17]). 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 the algorithms have been run and the best partition was selected.

The last four columns of Table 8 shows the partitionings produced by our multilevel hypergraph bisection and refinement algorithms. In particular, the column labeled "hMETIS-EE$_{20}$" corresponds to the best partitioning produced from 20 runs of our multilevel algorithm that uses early-exit FM during refinement (FM-EE). Of these twenty runs, ten runs are using hyperedge coarsening (HEC) and ten runs are using modified hyperedge coarsening (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 10 runs of our multilevel

	Number of Hyperedge Cut					Quality Relative to the Best			
	EE-FM		FM			EE-FM		FM	
Circuit	V	vV	V	vV	Best	V	vV	V	vV
19ks	106	106	105	105	105	1.0	1.0		
avq.large	127	134	127	127	127		5.5		
avq.small	128	128	132	130	128			3.1	1.6
baluP	27	27	27	27	27				
biomedP	83	83	83	83	83				
bm1	51	51	51	51	51				
golem3	1497	1425	1422	1424	1422	5.3	0.2		0.1
industry2P	168	169	169	168	168		0.6	0.6	
industry3	252	252	249	244	244	3.3	3.3	2.0	
p1	49	49	51	51	49			4.1	4.1
p2	148	148	145	145	145	2.1	2.1		
s9234P	40	40	40	40	40				
s13207P	58	61	58	58	58		5.2		
s15850	42	42	41	42	41	2.4	2.4		2.4
s35932	41	42	42	42	41		2.4	2.4	2.4
s38417	54	54	53	52	52	3.8	3.8	1.9	
s38584	48	48	48	48	48				
structP	33	33	33	33	33				
t2	92	92	88	88	88	4.5	4.5		
t3	59	59	59	59	59				
t4	48	48	48	48	48				
t5	71	71	71	71	71				
t6	63	63	63	63	63				
Aggregate Cut	3285	3225	3205	3199	3191				
Average QRB						0.9	1.3	0.6	0.4
Total Time	409	408	1150	781					

Table 7: The Effect of the V-cycles and vV-cycles on the refinement schemes

partitioning algorithm that uses the vV-cycle refinement scheme. These results were obtained using the modified hyperedge coarsening scheme (MHEC) and EE-FM for refinement. Finally, the column labeled hMETIS-FM$_{20vV}$ corresponds to the best partition produced from 20 runs that uses 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 CLIP-PROP$_f$, 5.3% better than CDIP-LA3$_f$, 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 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 the previous schemes combined. In particular, hMETIS-FM$_{20}$ was able to improve the best-known bisections for 8 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 about 2.0% better than hMETIS-EE$_{20}$ and 0.9% better that hMETIS-FM$_{20}$. 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 subtable of Table 8 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 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

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

Table 8: Performance of our multilevel hypergraph bisection algorithm (hMETIS) against various previously developed algorithms.

about four times more time than when it is running on R10000. Taking into consideration a scaling factor of four, we see that both hMETIS-EE_{20} and hMETIS-FM_{20} require less time than either PROP, CDIP-$LA3_f$, CLIP-$PROP_f$, PARABOLI, or GFM. In particular, hMETIS-EE_{20} is about four times faster than PROP, nine times faster than CDIP-$LA3_f$ and CLIP-$PROP_f$, and much faster than PARABOLI, GFM and Optimized KLFM. Comparing against GMetis, we see that hMETIS-EE_{20} requires roughly the same time, whereas 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 hMETIS-EE_{20} and hMETIS-FM_{20} produce bisections that cut substantially fewer hyperedges than GMetis.

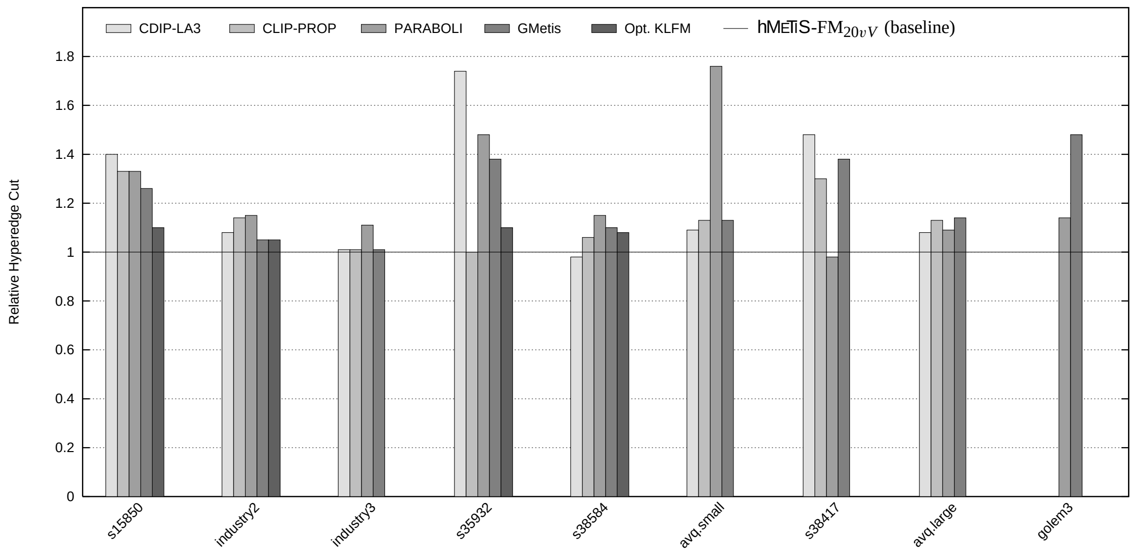
Looking at the amount of time required by hMETIS-EE_{10vV} and hMETIS-FM_{20vV}, we see that by using multi-phase refinement we were in general able to further reduce the amount of time required by our partitioning algorithms. In particular, hMETIS-EE_{10vV} requires only 409 sec to partition all 23 circuits whereas hMETIS-FM_{20vV} requires 1513 seconds.

6 Conclusions and Future Work

As the experiments in Section 3 show, the multilevel paradigm is very successful in producing high quality hypergraph partitionings in relatively small amount of time. The multilevel paradigm is successful because of the following reasons. The coarsening phase is able to generate a sequence of hypergraphs that are good approximations of the original hypergraph. Then, the initial partitioning algorithm is 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 runtime of the scheme increases linearly with hypergraph size. Furthermore, the scheme appears to be more powerful relative to the other schemes for larger hypergraph (please refer to Figure 10). Restricting to only the larger hypergraphs (with 10K 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-$LA3_f$, CLIP-$PROP_f$, PARABOLI, GFM, GMetis, and Optimized KLFM, respectively. Note that hypergraph-based multilevel scheme as presented in this paper significantly outperforms the graph based multilevel scheme GMetis [21] that used METIS [27] to compute bisections of graph approximations of hypergraph. The reasons for this performance difference are as follows. First hypergraph-based coarsening cause 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 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 10 runs of hMETIS-FM_{20vV}. hMETIS also significantly outperforms Optimized KLFM [17] by Hauck and Borriello even though they used powerful refinement schemes (FM



centerline

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

with LA_3 [4]). This is primarily due to the more powerful hyperedge coarsening 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.*, [22, 23]. It will be instructive to include such a refinement scheme during the uncoarsening phase to see if it makes the multi-level 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 can be made to potentially improve the bisection.

A version of hMETIS is available on the WWW at the following URL: **http://www.cs.umn.edu/˜metis**.

References

[1] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[2] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 57–62, 1972.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *In Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[4] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, Vol. C-33, May 1984.

[5] T. Bui et al. Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithm. In *Proc. ACM/IEEE Design Automation Conference*, pages 775–778, 1989.

[6] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[7] Lars Hagen and Andrew Kahng. A new approach to effective circuit clustering. In *Proceedings of IEEE International Conference on Computer Aided Design*, pages 422–427, 1992.

[8] Stephen T. Barnard and Horst D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. In *Proceedings of the sixth SIAM conference on Parallel Processing for Scientific Computing*, pages 711–718, 1993.

[9] F. Brglez. ACM/SIGDA design automation benchmarks: Catalyst or anathema? *IEEE Design & Test*, 10(3):87–91, 1993. Available on the WWW at *http://vlsicad.cs.ucla.edu/˜cheese/benchmarks.html.*

[10] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[11] 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 Conference*, pages 755–760, 1993.

[12] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[13] H. Shin and C. Kim. A simple yet effective technique for partitioning. *IEEE Transactions on VLSI Systems*, 1(3), 1993.

[14] E. Ihler, D. Wagner, and F. Wagner. Modeling hypergraphs by graphs with the same mincut properties. *Information Processing Letters*, 45(4), March 1993.

[15] B. M. Riess, K. Doll, and F. M. Johannes. Partitioning very large circuits using analytical placement techniques. In *Proceedings ACM/IEEE Design Automation Conference*, pages 646–651, 1994.

[16] Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning. *Integration, the VLSI Journal*, 19(1-2):1–81, 1995.

[17] S. Hauck and G. Borriello. An evaluation of bipartitioning technique. In *Proc. Chapel Hill Conference on Advanced Research in VLSI*, 1995.

[18] J. Li, J. Lillis, and C. Cheng. Linear decomposition algorithm for vlsi design applications. In *Proceedings of IEEE Intl. Conf. Computer Aided Design*, pages 223–228, 1995.

[19] J. Li, J. Lillis, and C. K. Cheng. Linear decomposition algorithm for VLSI design applications. In *Proceedings of IEEE Intl. Conf. Computer Aided Design*, pages 223–228, 1995.

[20] Y. Saab. A fast and robust network bisection algorithm. *IEEE Transactions on Computers*, 44(7):903–913, 1995.

[21] C. Alpert and A. Kahng. A hybrid multilevel/genetic approach for circuit partitioning. In *Proceedings of the Fifth ACM/SIGDA Physical Design Workshop*, pages 100–105, 1996.

[22] S. Dutt and W. Deng. A probability-based approach to VLSI circuit partitioning. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1996.

[23] S. Dutt and W. Deng. VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In *Proc. Physical Design Workshop*, 1996.

[24] Charles J. Alpert Lars W. Hagen and Andrew B. Kahng. A general framework for vertex orderings, with applications to netlist clustering. *IEEE Transactions on VLSI*, 4(2):240–246, 1996.

[25] B. Mobasher, N. Jain, E.H. Han, and J. Srivastava. Web mining: Pattern discovery from world wide web transactions. Technical Report TR-96-050, Department of Computer Science, University of Minnesota, Minneapolis, 1996.

[26] S. Shekhar and D. R. Liu. Partitioning similarity graphs: A framework for declustering problmes. *Information Systems Journal*, 21(4), 1996.

[27] G. Karypis and V. Kumar. METIS 3.0: Unstructured graph partitioning and sparse matrix ordering system. Technical Report 97-061, Department of Computer Science, University of Minnesota, 1997. Available on the WWW at URL *http://www.cs.umn.edu/˜metis*.

[28] S. Shekhar and D. R. Liu. CCAM: A connectivity-clustered access method for networks and network computations. *IEEE Transactions on Knowledge and Data Engineering*, 9(1), 1997.

[29] C. Berge. *Graphs and Hypergraphs*. American Elsevier, New york, 1976.

[30] Michael R. Garey and David S. Johnson. *Computers and Instractability: A Guide to the Theory of NP–Completeness*. W.H Freeman, San Francisco, CA, 1979.

[31] T. Lengauer. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, Boston, MA, 1976.

[32] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[33] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998 (to appear). Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.