A short version of this paper appears in the 36th Design Automation Conference

The algorithms described in this paper are implemented by the
'hMETIS: Hypergraph Partitioning Package'.
hMETIS is available on WWW at URL: http://www.cs.umn.edu/~metis

Multilevel k-way Hypergraph Partitioning*

George Karypis and Vipin Kumar

Department of Computer Science & Engineering
Army HPC Research Center
University of Minnesota, Minneapolis, MN 55455
Technical Report #98–036

{karypis, kumar}@cs.umn.edu

Abstract

In this paper, we present a new multilevel k-way hypergraph partitioning algorithm that substantially outperforms the existing state-of-the-art K-PM/LR algorithm for multi-way partitioning. both for optimizing local as well as global objectives. Experiments on the ISPD98 benchmark suite show that the partitionings produced by our scheme are on the average 15% to 23% better than those produced by the K-PM/LR algorithm, both in terms of the hyperedge cut as well as the $(K-1)$ metric. Furthermore, our algorithm is significantly faster, requiring 4 to 5 times less time than that required by K-PM/LR.

1 Introduction

Hypergraph partitioning is an important problem with extensive application to many areas, including VLSI design [10], efficient storage of large databases on disks [14], and data mining [13]. The problem is to partition the vertices of a hypergraph into k roughly equal parts, such that a certain objective function defined over the hyperedges is optimized. A commonly used objective function is to minimize the number of hyperedges that span different partitions; however, a number of other objective functions are also considered useful [10].

The most commonly used approach for computing a k-way partitioning is based on the recursive bisection paradigm, that reduces the problem of computing a k-way partitioning to that of performing a sequence of bisections. The problem of computing an optimal bisection of a hypergraph is at least NP-hard [24]; however, many heuristic algorithms

*This work was supported by IBM Partnership Award, NSF CCR-9423082, Army Research Office contract DA/DAAG55-98-1-0441, and the 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**

have been developed. The survey by Alpert and Kahng [10] provides a detailed description and comparison of various such schemes. Recently a new class of hypergraph bisection algorithms has been developed [11, 26, 15, 22], that are based upon the multilevel paradigm. In these algorithms, a sequence of successively smaller (coarser) hypergraphs is constructed. A bisection of the smallest hypergraph is computed. This bisection is then successively projected to the next level finer hypergraph, and at each level an iterative refinement algorithm (*e.g.*, KL [1] or FM [3]) is used to further improve the bisection. Experiments presented in [26, 15, 22] have shown that multilevel hypergraph bisection algorithms can produce substantially better partitionings than those produced by non-multilevel schemes. In particular, hMETIS [20], a multilevel hypergraph bisection algorithm based upon the work in [26] has been shown to find substantially better bisections than current state-of-the-art iterative refinement algorithms for the ISPD98 benchmark set that contains many large circuits [18].

Despite the success of multilevel recursive bisection algorithms, there are a number of advantages of computing the k-way partitioning directly (rather than computing it successively via recursive bisection). First, a recursive bisection algorithm does not allows us to directly optimize objectives that are global in nature and depend on having a direct view of all k partitions. Some examples of such objectives are the sum of external degrees (SOED), scaled cost, and absorption [10]. Second, a k-way partitioning algorithm is capable of enforcing tighter balancing constraints while retaining the ability to sufficiently explore the feasible solution space to optimize the partitioning objective. This is especially true when the partitioning solution must simultaneously satisfy multiple balancing constraints [21]. Third, a method that obtains a k-way partitioning directly can potentially produce much better partitionings than a method that computes a k-way partitioning via recursive bisection [8].

For these reasons, researchers have investigated a number of k-way partitioning algorithms that try to compute a k-way partitioning directly, rather than via recursive bisection. The most notable of them are the generalization of the FM algorithm for k-way partitioning [4, 7], the spectral multi-way ratio-cut [6], the primal-dual algorithm of [5], the geometric embedding [9], the dual-net method [12], and the K-PM/LR algorithm [19]. A key problem faced by some of these algorithms is that the k-way FM refinement algorithm easily gets trapped in local minima. The recently developed K-PM/LR algorithm by Cong and Lim [19] attempts to solve this problem by refining a k-way partitioning by applying a sequence of 2-way FM refinement to pairs of domains. The pairing of domains is based on the gain of the last pass, and the pairwise cell movement passes continues until no further gain can be obtained. The experiments presented in [19] have shown that K-PM/LR outperforms the k-way FM partitioning algorithm of Sanchis [4, 7] by up to 86.2% and outperforms the recursive FM partitioning algorithm by up to 17.3%. Nevertheless, all of the above partitioners tend to produce solutions that are inferior to those produced by the state-of-the-art multilevel recursive bisection algorithms, especially when they are used to optimize an objective that can directly be optimized by the recursive bisection framework (*e.g.*, minimize the hyperedge cut) [18].

In this paper we present a new k-way partitioning algorithm that is based on the multilevel paradigm. The multilevel paradigm can be used to directly construct a k-way partitioning of a hypergraph using the framework illustrated in Figure 1. The hypergraph is coarsened successively as before. But the coarsest hypergraph is now directly partitioned into k parts, and this k-way partitioning is successively refined as the partitioning is projected back into the original hypergraph. A key contribution of our work is a simple and yet powerful scheme for refining a k-way partitioning in the multilevel context. This k-way partitioning refinement scheme is substantially simpler and faster than either the k-way FM [4], or the K-PM/LR algorithm [19], but is equally effective in the multilevel context. Furthermore, this new k-way refinement algorithm is inherently parallel [17] making it possible to develop high-quality parallel hypergraph partitioning algorithms.

We evaluate the performance of our multilevel k-way partitioning algorithm both in terms of the partitioning quality as well as computational requirements on the ISPD98 benchmark [18]. Our experiments show that the multilevel k-way 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 15% to 23% better than those produced by the

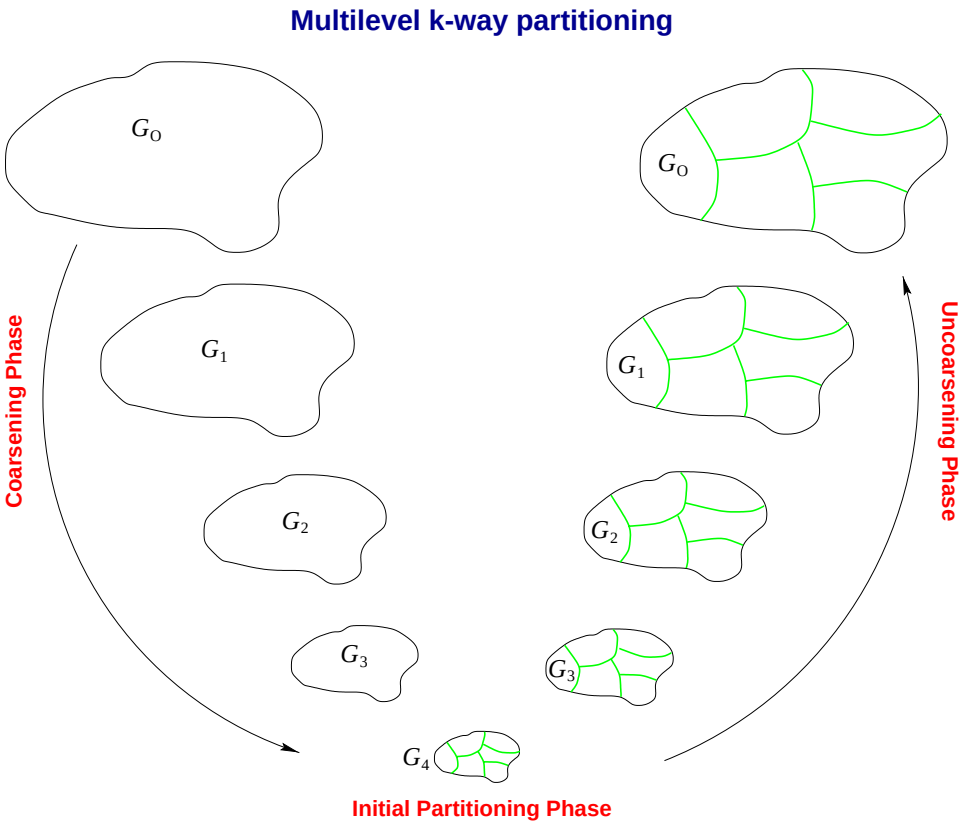


Figure 1: The various phases of the multilevel k-way partitioning algorithm. During the coarsening phase, the size of the hypergraph is successively decreased; during the initial partitioning phase, a k-way partitioning of the smaller hypergraph is computed (a 6-way partitioning in this example); and during the uncoarsening phase, the partitioning is successively refined as it is projected to the finer (larger) hypergraphs.

K-PM/LR [19] algorithm, both in terms of the hyperedge cut as well as the $(K-1)$ metric. Furthermore, our algorithm is significantly faster, requiring 4 to 5 times less time than that required by K-PM/LR and provides partitions that adhere to tighter balancing constraints. Compared to the state-of-the-art multilevel recursive bisection, our experiments show that with respect to the hyperedge cut, our algorithm produces partitions of comparable quality, whereas with respect to the SOED, our algorithm produces partitions that are up to 18% better. Furthermore, our multilevel k-way partitioning algorithm is in general two times faster than multilevel recursive bisection, and this ratio increases with the size of the hypergraph.

The rest of this paper is organized as follows. Section 2 describes the different algorithms used in the three phases of our multilevel k-way hypergraph partitioning algorithm. Section 3 compares the results produced by our algorithm to those produced by earlier hypergraph partitioning algorithms. Finally, Section 4 provides some concluding remarks.

2 Multilevel k-way Hypergraph Partitioning

Formally, a hypergraph $G = (V, E)$ is defined as a set of vertices V and a set of hyperedges E, where each hyperedge is a subset of the vertex set V [23], and the size of a hyperedge is the cardinality of this subset. The ***k-way*** hypergraph partitioning problem is defined as follows: Given a hypergraph $G = (V, E)$ (where V is the set of vertices and E is the set of hyperedges) and an overall load imbalance tolerance c such that $c \geq 1.0$, the goal is to partition the set V into k disjoint subsets, $V_1, V_2, \ldots, V_k$ such that the number of vertices in each set V_i is bounded by $|V|/(ck) \leq |V_i| \leq c|V|/k$, and a function defined over the hyperedges is optimized.

The requirement that the size of each partition is bounded is referred to as the ***partitioning constraint***, and the requirement that a certain function is optimized is referred to as the ***partitioning objective***. Over the years, a number of partitioning objective functions have been developed. The survey by Alpert and Kahng [10] provides a comprehensive description of a variety of objective functions that are commonly used for hypergraph partitioning in the context of VLSI design.

One of the most commonly used objective function is to ***minimize the hyperedge-cut*** of the partitioning; *i.e.*, the total number of hyperedges that span multiple partitions. Another objective that is often used is to ***minimize the sum of external degrees*** (SOED) of all hyperedges that span multiple partitions. Given a k-way partitioning and a hyperedge e, the external degree of e is defined to be 0, if e is not cut by the partitioning, otherwise it is equal to the number of partitions that is spanned by e. Then, the goal of the partitioning algorithm is to compute a k-way partitioning that minimizes the sum of external degrees of the hyperedges. An objective related to SOED is to ***minimize the*** $(K-1)$ ***metric*** [10, 19]. In the case of the $(K-1)$ metric, the cost of a hyperedge that spans K partitions is $(K-1)$, whereas for the SOED metric, the cost is K.

Next we describe the three phases of the multilevel k-way partitioning algorithm in detail.

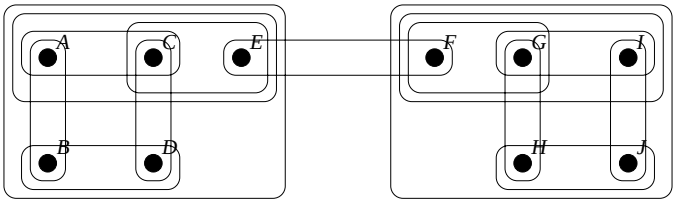
2.1 Coarsening Phase

During the coarsening phase, a sequence of successively smaller hypergraphs is constructed. As in the case of the multilevel hypergraph bisection algorithm [16], the coarsening phase servers the following three purposes. First it leads to a small hypergraph such that a good k-way partitioning of the small hypergraph is not significantly worse than the k-way partitioning directly obtained for the original hypergraph. Second, the different successively coarsened versions of the hypergraph allow local refinement techniques such as FM to become effective. Third, hypergraph coarsening also helps in successively reducing the sizes of the hyperedges. That is, at each level of coarsening, large hyperedges are contracted to smaller hyperedges. This is particularly helpful, since refinement heuristics based on the KLFM family 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.

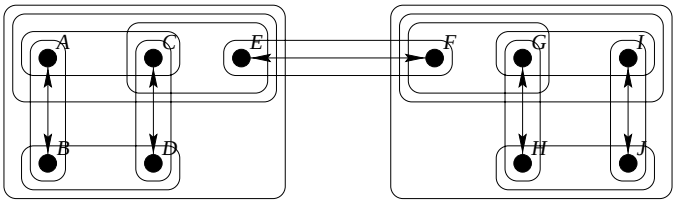
Two primary schemes have been developed for selecting what groups of vertices will be merged together to form single vertices in the next level coarse hypergraphs. The first scheme called ***edge-coarsening*** (EC) [16, 15, 22], selects the groups by finding a maximal set of pairs of vertices (*i.e.*, matching) that belong in many hyperegdes. In this scheme, each group consists of at most two vertices (some vertices are not combined at all), and each vertex belongs to exactly one group. The second scheme that is called ***hyperedge-coarsening*** (HEC) [16] finds a maximal independent set of hyperedges, and the sets of vertices that belong to each hyperedge becomes a group of vertices to be merged together. In this scheme, each group can have an arbitrary number of vertices (even though preference is given to smaller groups), and each vertex also belongs to exactly one group. Experiments in [16] show that for certain problems, the hyperedge-coarsening scheme and its variations tend to outperform the edge-coarsening scheme, as they do a better job of reducing the total hyperedge weight of successively coarser hypergraphs.

However, both the edge- and the hyperedge-coarsening schemes share one characteristic that can potentially lead to less than ideal coarse representations of the original hypergraph, especially for hypergraphs corresponding to circuits. This common characteristic is that the grouping schemes employed by both approaches find maximal independent groups. That is, both the edge- and the hyperedge-coarsening schemes will find as many groups of vertices as they can, that are pair- or hyperedge-wise independent. The potential problem with this approach is that the independence (and to a certain degree, the maximality) requirement may destroy some clusters of vertices that naturally exist in the hypergraph. To see that consider the example shown in Figure 2(a). As we can see from this figure there are two natural clusters. The first cluster contains the five vertices on the left and the second cluster contains the five vertices on the right. These two clusters are connected by a single hyperedge; thus the *natural* cut for this hypergraph is one. Figure 2(b) shows the pairs of vertices that are found by the edge-coarsening scheme. In the edge-coarsening scheme,

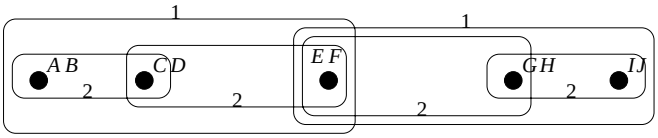
vertex F will prefer to merge with vertex G, but vertex G had already been grouped with vertex H, consequently, vertex E is grouped together with vertex E. Once the hypergraph is coarsened as shown in Figure 2(c), we can see that the natural separation point in this hypergraph has been eliminated, as it has been contracted in the vertex that resulted from merging E and F. A similar kind of example can be constructed using the hyperedge-coarsening as well.



(a) Initial Hypergraph



(b) Groups Determined by Edge-Coarsning



(c) Coarse Hypergraph

Figure 2: An example in which the edge-coarsening scheme can lead to a coarse representation in which the natural clusters of the hypergraph have been obscured. The weights on the hyperedges of the coarse hypergraph (c) represent the number of hyperedges in the original hypergraph that span the same set of vertices in the coarse representation.

The above observation, led us to develop a new coarsening scheme that we will refer to as the ***FirstChoice*** (FC) coarsening scheme. The FC coarsening scheme is based up on the edge-coarsening scheme, and understanding how the EC scheme works is essential in understanding FC. In the rest of this section we briefly describe the EC scheme (further details can be found in [16, 15]), and describe how FC can be derived by modifying EC.

In the EC coarsening scheme, 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 [25]. However, this hypergraph to graph conversion is done implicitly during matching without forming the actual graph.

The FC coarsening scheme is derived from the EC coarsening scheme by relaxing the requirement that a vertex is matched only with another unmatched vertex. Specifically, in the FC coarsening scheme, the vertices are again visited

in a random order. However, for each vertex v, all vertices (both matched and unmatched) 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, breaking ties in favor of unmatched vertices. As a result, each group of vertices to be merged together can contain an arbitrarily large number of vertices. The one potential problem with this coarsening scheme is that the number of vertices in successive coarse graphs may decrease by a large factor,* potentially limiting the effect of refinement [15]. For this reason, at each coarsening level, we stop the FC coarsening scheme as soon as the size of the resulting coarse graph has been reduced by a factor of 1.7. Our experiments have shown that by controlling the rate of coarsening in this fashion, we can ensure that there are sufficiently many coarsening levels, and that the refinement algorithm is effective in improving the partitioning quality during the uncoarsening phase.

The coarsening phase ends when the coarsest hypergraph has a small number of vertices. Since our goal is to compute a k-way partitioning, the number of vertices in this smaller hypergraph should be a function of k, to ensure that a reasonably balanced partitioning can be computed by the initial partitioning algorithm. In our experiments, for a k-way partition, we stop the coarsening process when the number of vertices becomes less than ck, where $c = 100$.

2.2 Initial Partitioning Phase

The second phase of a multilevel k-way partitioning algorithm is to compute a k-way partitioning of the coarsest hypergraph such that the balancing constraint is satisfied and the partitioning objective is optimized. Since during coarsening, the weights of the vertices and hyperedges of the coarser hypergraph were set to reflect the weights of the vertices and hyperedges of the finer hypergraph, the coarsest hypergraph contains sufficient information to intelligently enforce the balancing constraint and optimize the partitioning objective.

One way to produce the initial k-way partitioning is to keep coarsening the hypergraph until it has only k vertices left. These coarse k vertices can serve as the initial k-way partitioning of the original hypergraph. There are two problems with this approach. First, for many hypergraphs, the reduction in the size of the hypergraph in each coarsening step becomes very small after some coarsening steps, making it very expensive to continue with the coarsening process. Second, even if we are able to coarsen the hypergraph down to only k vertices, the weights of these vertices are likely to be quite different, making the initial partitioning highly unbalanced.

In our algorithm, the k-way partitioning of the coarsest hypergraph is computed using our multilevel hypergraph bisection algorithm [16], that is available in the hMeTiS package [20].

2.3 Uncoarsening 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 optimize the objective function without violating the partitioning balancing constraints. Since the next level finer hypergraph has more degrees of freedom, such refinement algorithms tend to improve the solution quality.

In the case of bisection refinement, the FM algorithm [3] has been shown to produce very good results [16]. For each vertex v, the FM algorithm computes the ***gain*** which is the reduction in the value of the objective function achieved by moving v to the other partition. These vertices are inserted into two priority queues, one for each partition, according to their gains. 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 from one of the two priority queues 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 value of the objective function achieved at this point. A single pass of the algorithm ends when there are no more unlocked vertices. Then, the recorded values of the objective function

*In the case of the EC coarsening scheme, the size of successive coarse graphs can be reduced by at most a factor of two.

are checked, and the point where the minimum value 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.

However, refining a k-way partitioning is significantly more complicated because vertices can move from a partition to many other partitions; thus, increasing the optimization space combinatorially. An extension of the FM refinement algorithm in the case of k-way refinement is described in [4]. This algorithm uses $k(k-1)$ priority queues, one for each type of move. In each step of the algorithm, the moves with the highest gain are found from each of these $k(k-1)$ queues, and the move with the highest gain that preserves or improves the balance, is performed. After the move, all of the $k(k-1)$ priority queues are updated. The complexity of k-way refinement is significantly higher than that of 2-way refinement, and is only practical for small values of k. Furthermore, as the experiments in [19] suggest, the k-way FM algorithm is also very susceptible of being trapped into a local minima that is far from being optimal.

Benchmark	No. of vertices	No. of hyperedges
ibm01	12506	14111
ibm02	19342	19584
ibm03	22853	27401
ibm04	27220	31970
ibm05	28146	28446
ibm06	32332	34826
ibm07	45639	48117
ibm08	51023	50513
ibm09	53110	60902
ibm10	68685	75196
ibm11	70152	81454
ibm12	70439	77240
ibm13	83709	99666
ibm14	147088	152772
ibm15	161187	186608
ibm16	182980	190048
ibm17	184752	189581
ibm18	210341	201920

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

The hill-climbing capability of the FM algorithm serves a very important purpose. It allows movement of an entire cluster of vertices across a partition boundary. Note that it is quite possible that as the cluster is moved across the partition boundary, the value of the objective function increases, but after the entire cluster of vertices moves across the partition, then the overall value of the objective function comes down. In the context of multilevel schemes, this hill-climbing capability becomes less important. The reason is that these clusters of vertices are coarsened into a single vertex at successive coarsening phases. Hence, movement of a vertex at a coarse level really corresponds to the movement of a group of vertices in the original hypergraph.

If the hill-climbing part of the FM algorithm is eliminated (*i.e.*, if vertices are moved only if they lead to positive gain), then it becomes less useful to maintain a priority queue. This is because vertices whose move results in a large positive gain will most likely be moved anyway even if they are not moved earlier (in the priority order). Hence, a variation of the FM algorithm that simply visits the vertices in a random order and moves them if they result in a positive gain is likely to work well in the multilevel context. Furthermore, the complexity of this algorithm will be independent of the number of partitions being refined, leading to a fast algorithm. This observation has lead to us to develop a ***greedy refinement*** algorithm. It consists of a number of iterations. In each iteration all the vertices are checked to see if they can be moved so that the partitioning objective function is optimized, subject to the partitioning balancing constraint (as described in Section 2). As the results in Section 3 show, despite the simplicity of our refinement algorithms, they produce high quality partitionings in small amount of time.

More precisely, our greedy k-way refinement algorithm works as follows. Consider a hypergraph $G_i = (V_i, E_i)$, and its partitioning vector P_i. The vertices are visited in a random order. Let v be such a vertex, let $P_i[v] = a$ be the partition that v belongs to. If v is a node internal to partition a then v is not moved. If v is at the boundary of the partition, then v can potentially be moved to one of the partitions $N(v)$ that vertices adjacent to v belong to (the set $N(v)$ is often refer to as the ***neighborhood*** of v). Let $N'(v)$ be the subset of $N(v)$ that contains all partitions b such that movement of vertex v to partition b does not violate the balancing constraint. Now the partition $b \in N'(v)$ that leads to the greatest positive reduction (gain) in the objective function is selected and v is moved to that partition.

The above greedy refinement algorithm can be used to compute a partitioning that minimizes a variety of objective functions, by appropriately computing the gain achieved in moving a vertex. Our current implementation allows a choice of two different objective functions. The first minimizes the hyperedge cut and the second minimizes the sum of external degrees (SOED) (Section 2).

Experiments with this greedy k-way refinement algorithm show that it converges after a small number of iterations. In our experiments, we found that for most hypergraphs, the algorithm converged within four to eight iterations.

3 Experimental Results

We experimentally evaluated the quality of the partitionings produced by our multilevel k-way hypergraph partitioning algorithm (hMETiS-Kway) on the 18 hypergraphs that are part of the ISPD98 circuit partitioning benchmark suite [18]. The characteristics of these hypergraphs are shown in Table 1. In addition to the circuits, the ISPD98 benchmark also contains the actual areas for each one of the cell. However, to make it easy to compare our results with those of other researchers [19], we used only unit cell-areas in our experiments. Furthermore, for some circuits, the actual areas of some cells is higher than 1/8 of the overall area, making it impossible to produced balanced 8-, 16-, and 32-way partitionings. We performed all of our experiments on a 300MHz Pentium II–based Linux workstation.

3.1 Comparison with the Multilevel Recursive Bisection

In our first set of experiments, we compare the performance of our multilevel k-way partitioning algorithm to that of the multilevel recursive bisection algorithm for computing 8-, 16-, and 32-way partitionings. Our multilevel k-way partitioning algorithm was compared against the multilevel bisection algorithm [26] that is part of the hMETiS [20] hypergraph partitioning package. For the rest of this paper, we will refer to this recursive bisection algorithm as hMETiS-RB, and we will refer to our multilevel k-way partitioning algorithm as hMETiS-Kway.

Both hMETiS-RB and hMETiS-Kway used the FC scheme for coarsening (Section 2.1). For refinement, hMETiS-RB used the FM algorithm whereas the hMETiS-Kway used the greedy refinement algorithm described in Section 2.3. To compute a bisection using hMETiS-RB, we performed a total of 20 different runs, and then we further improved the best bisection using the V-cycle refinement technique [26]. To ensure that the overall k-way partitioning does not become significantly unbalanced, each bisection was computed using a $[48, 52]$ balancing constraint (*i.e.*, the smaller part must contain at least 48% of the vertices). Consequently, the effective overall balancing constraints for the 8-, 16-, and 32-way partitionings were $[0.48^3 = 0.111, .52^3 = 0.141]$, $[0.48^4 = 0.053, .52^4 = 0.073]$, and $[0.48^5 = 0.025, .52^5 = 0.038]$, respectively. In other words, these balancing constraints allow an overall maximum load imbalance of 12.5%, 17.0%, and 21.7%, for the 8-, 16-, and 32-way partitionings, respectively. We also performed a total of 20 different runs for hMETiS-Kway, and we also used the V-cycle refinement technique to further improve the quality of the best k-way partitioning. In all the experiments, hMETiS-Kway used an overall load imbalance tolerance of 1.10, meaning that the weight of the heaviest partition will be less than 10% higher than the average weight of the k partitions.

Table 2 shows the number of hyperedges that are cut by both hMETiS-RB and hMETiS-Kway for an 8-, 16-, and 32-way partitioning for all the circuits of the ISPD98 benchmark. For this set of experiments, the objective of hMETiS-

Circuit	hMETiS-RB			hMETiS-Kway		
	8-way	16-way	32-way	8-way	16-way	32-way
ibm01	760	1258	1723	795	1283	1702
ibm02	1720	3150	4412	1790	3210	4380
ibm03	2503	3256	4064	2553	3317	4120
ibm04	2857	3989	5094	2902	3896	5050
ibm05	4548	5465	6211	4464	5612	5948
ibm06	2452	3356	4343	2397	3241	4231
ibm07	3454	4804	6300	3422	4764	6212
ibm08	3696	4916	6489	3544	4718	6154
ibm09	2756	3902	5502	2680	3968	5490
ibm10	4301	6190	8659	4263	6209	8612
ibm11	3592	5260	7514	3713	5371	7534
ibm12	5913	8540	11014	6183	8569	11392
ibm13	3042	5522	7541	2744	5329	7610
ibm14	5501	8362	12681	5244	8293	12838
ibm15	6816	8691	13342	6855	9201	13853
ibm16	6871	10230	15589	6737	10250	15335
ibm17	9341	15088	20175	9420	15206	19812
ibm18	5310	8860	13410	5540	9025	13102
ARQ	1.002	0.996	1.006	0.998	1.004	0.994
Run-time	21872.22	25941.12	30325.48	10551.7	14227.52	19572.45

Table 2: The number of hyperedges that are cut by the multilevel recursive bisection algorithm (hMETiS-RB) and the multilevel k-way partitioning algorithm (hMETiS-Kway) for 8-, 16-, and 32-way partitionings. The row labeled 'ARQ' shows the Average Relative Quality of one scheme versus the other. For example, the ARQ value of 1.002 for the 8-way partitioning of hMETiS-RB means that the cuts produced by hMETiS-RB are on the average 0.2% higher than the corresponding cuts produced by hMETiS-Kway. An ARQ value that is less than 1.0 indicates that the particular scheme on the average performs better. The last row shows the total amount of time required by each of the partitioners for all 18 circuits (the times are in seconds).

Kway algorithm was to minimize the hyperedge cut. The same set of data was also used to plot the bar-charts shown in Figure 3 that show the cut obtained by hMETiS-Kway relative to that obtained by hMETiS-RB. These bars were obtained by dividing the cut obtained by hMETiS-Kway to the cut obtained by hMETiS-RB. Any bars lower than 1.0 indicate that hMETiS-Kway performs better. As can be seen from Figure 3, hMETiS-Kway produces partitions whose cut is comparable to those produced by hMETiS-RB. On the average, hMETiS-Kway performs 0.2% and 0.6% better than hMETiS-RB for the 8- and 32-way partitionings, respectively, and 0.4% worse for the 16-way partitioning. The fact that hMETiS-Kway cuts the same number of hyperedges as hMETiS-RB, is especially interesting if we consider (i) the simplicity of the greedy refinement scheme used by hMETiS-Kway as opposed to the much more sophisticated FM algorithm used by hMETiS-RB, and (ii) the fact that compared to hMETiS-Kway, hMETiS-RB operates under more relaxed balancing constraints.

The last row of Table 2 shows the total amount of time required by the two algorithms in order to compute the 8-, 16-, and 32-way partitionings. As we can see, hMETiS-Kway is 2.07, 1.82, and 1.55 times faster than hMETiS-RB for computing an 8-, 16-, and a 32-way partitioning, respectively. Note that this relative speed advantage of hMETiS-Kway decreases as k increases. This is primarily due to the fact that the recursive bisection algorithm used in the initial partitioning takes a larger fraction of the overall time (as the size of the coarsest hypergraph is proportional to the number of partitions). hMETiS-Kway will continue running faster than hMETiS-RB if the size of the hypergraph is increased proportionally to the number of partitions.

To test the effectiveness of hMETiS-Kway for optimizing the SOED, we ran another set of experiments in which the objective of hMETiS-Kway was to minimize the SOED. Table 3 shows the sum of external degrees (SOED) of the partitionings produced by both hMETiS-RB and hMETiS-Kway for an 8-, 16-, and 32-way partitioning for all the circuits of the ISPD98 benchmark. The same set of data was also used to plot the bar-charts shown in Figure 4 that show the SOED obtained by hMETiS-Kway relative to that obtained by hMETiS-RB. From this figure we can see that for all cases, hMETiS-Kway produces partitionings whose SOEDs are better than those produced by hMETiS-RB. On

Circuit	hMETIS-RB			hMETIS-Kway		
	8-way	16-way	32-way	8-way	16-way	32-way
ibm01	1768	2938	4566	1750	2883	4149
ibm02	3940	8040	13039	3850	7556	11821
ibm03	5909	8719	11667	5820	8205	11077
ibm04	6461	9595	13008	6214	8992	12495
ibm05	11572	16070	22708	10749	15206	20020
ibm06	6160	9631	13988	5784	8661	12779
ibm07	7885	12116	16806	7586	11040	15559
ibm08	9031	13040	18819	7979	10976	15327
ibm09	6073	9016	13193	5822	8634	12460
ibm10	9458	14543	21060	9144	13130	19941
ibm11	7940	12023	17857	7874	11706	17118
ibm12	12975	19563	27026	12910	17848	25228
ibm13	7010	12792	18484	6079	11819	17350
ibm14	12360	19189	30484	11258	18232	29699
ibm15	15198	21314	32039	14586	20826	31874
ibm16	14853	23237	37234	14616	22924	34879
ibm17	20423	34177	48256	19930	33344	45961
ibm18	12940	21765	34069	12177	19598	30558
ARQ	1.048	1.068	1.076	0.954	0.936	0.929

Table 3: The sum of external degrees (SOED) of the hyperedges that are cut by the partitionings produced by the multilevel recursive bisection algorithm (hMETIS-RB) and the multilevel k-way partitioning algorithm (hMETIS-Kway) for 8-, 16-, and 32-way partitionings. The row labeled 'ARQ' shows the Average Relative Quality of one scheme versus the other. For example, the ARQ value of 1.048 for the 8-way partitioning of hMETIS-RB means that the SOEDs produced by hMETIS-RB are on the average 4.8% higher than the corresponding SOEDs produced by hMETIS-Kway. An ARQ value that is less than 1.0 indicates that the particular scheme on the average performs better.

the average, hMETIS-Kway performs 4.8%, 6.8%, and 7.6% better than hMETIS-RB for the 8-way, 16-way, and 32-way partitionings, respectively. These results show that hMETIS-Kway is effective in incorporating global objective functions which can only be optimized in the context of a k-way refinement algorithm.

3.2 Comparison with K-PM/LR

We compared the performance of our multilevel k-way partitioning algorithm against the multi-way partitioning algorithm K-PM/LR developed by Cong and Lim [19].

Table 4 shows the number of hyperedges that are cut by both hMETIS-Kway and K-PM/LR for an 8- and a 16-way partitioning.[♣] In these experiments, for both hMETIS-Kway and K-PM/LR, the partitioning objective was to minimize the hyperedge cut. The results for hMETIS-Kway are the same as shown in Table 2, whereas the results from K-PM/LR are taken from [19]. Note that the results for K-PM/LR were obtained by using balancing constraints that correspond to those obtained by recursive bisection if it used a $[0.45, 0.55]$ balancing constraint at each level. Consequently, the balancing constraints for the 8- and 16-way partitioning are $[0.45^3 = 0.091, 0.55^3 = 0.166]$ and $[0.45^4 = 0.041, 0.55^4 = 0.092]$, respectively. Note that these balancing constraints are considerably more relaxed than the 10% overall load imbalanced used by hMETIS-Kway. If we translate the balancing constraints enforced by K-PM/LR to maximum allowable load imbalances for k-way partitioning, we see that K-PM/LR allows up to 32.8% and 47.2% load imbalance, for the 8-, and 16-way partitionings, respectively.

The data in Table 4 was also used to plot the bar-charts shown in Figure 5 that compares the cut obtained by hMETIS-Kway relative to those obtained by K-PM/LR. From this figure we can see that hMETIS-Kway produces partitionings that cut significantly fewer hyperedges than those cut by K-PM/LR. In fact, on the average, hMETIS-Kway cuts 20% and 23% fewer hyperedges than K-PM/LR for the 8- and 16-way partitionings, respectively. Thus, even though

[♣] We were not able to compare results for 32-way partitioning, because they are not reported in [19].

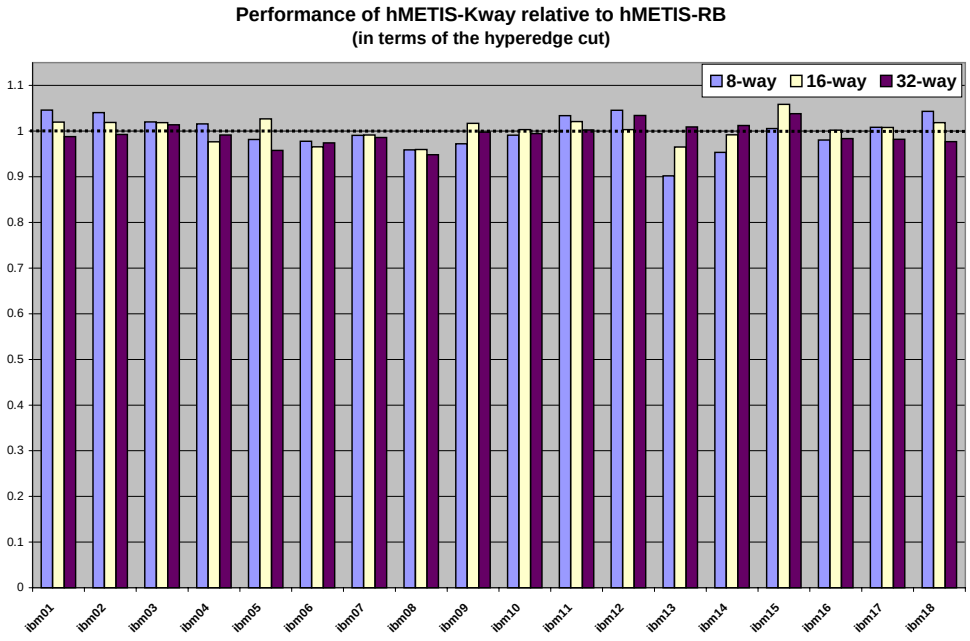


Figure 3: The quality of the partitionings in terms of the cuts produced by hMETIS-Kway relative to those produced by hMETIS-RB for an 8-, 16-, and 32-way partitioning. Bars bellow the 1.0 line indicate that hMETIS-Kway performs better than hMETIS-RB.

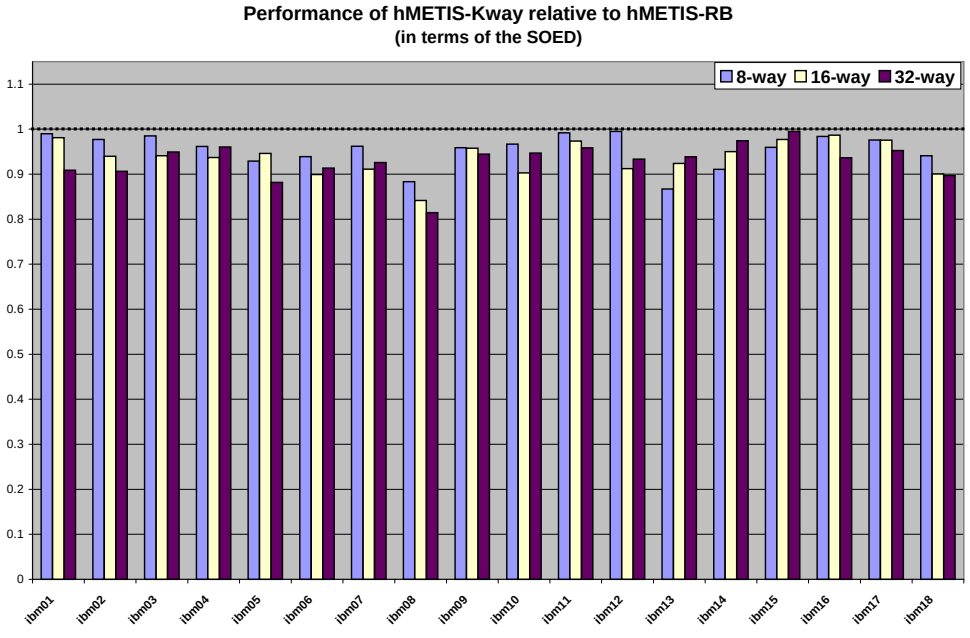


Figure 4: The quality of the partitionings in terms of the SOEDs produced by hMETIS-Kway relative to those produced by hMETIS-RB for an 8-, 16-, and 32-way partitioning. Bars bellow the 1.0 line indicate that hMETIS-Kway performs better than hMETIS-RB.

hMETiS-Kway operates under tighter balancing constraints, it is able to produce partitionings that cut substantially fewer hyperedges than K-PM/LR.

	hMETiS-Kway		K-PM/LR	
Circuit	8-way	16-way	8-way	16-way
ibm01	795	1283	1020	1699
ibm02	1790	3210	1751	3592
ibm03	2553	3317	3882	5736
ibm04	2902	3896	3559	5349
ibm05	4464	5612	4834	6419
ibm06	2397	3241	3198	4815
ibm07	3422	4764	4398	6854
ibm08	3544	4718	4466	6477
ibm09	2680	3968	4115	6046
ibm10	4263	6209	5252	8559
ibm11	3713	5371	6086	8871
ibm12	6183	8569	7736	11000
ibm13	2744	5329	3570	7066
ibm14	5244	8293	6753	9854
ibm15	6855	9201	8965	11345
ibm16	6737	10250	7543	10456
ibm17	9420	15206	10654	17653
ibm18	5540	9025	5765	9653
ARQ	0.802	0.771	1.247	1.297
Run-time	10551.7	14227.52	105840	134640

Table 4: The number of hyperedges that are cut by hMETiS-Kway and the K-PM/LR partitioning algorithms for 8- and 16-way partitionings. The row labeled 'ARQ' shows the Average Relative Quality of one scheme versus the other. For example, the ARQ value of 1.247 for the 8-way partitioning of K-PM/LR means that the cuts produced by K-PM/LR are on the average 24.7% higher (worse) than the corresponding cuts produced by hMETiS-Kway. An ARQ value that is less than 1.0 indicates that the particular scheme on the average performs better. The last row shows the total amount of time required by each of the partitioners for all 18 circuits (the times are in seconds). Note that hMETiS-Kway was run on a Pentium II@300Mhz, whereas K-PM/LR was run on a Ultra Sparc1@143Mhz.

The last row of Table 4 shows the amount of time required by hMETiS-Kway and K-PM/LR. Note that the K-PM/LR was run on a Sun Ultra Sparc1 running at 143Mhz. Our experiments have shown that the Sun Ultra Sparc1 running at 143Mhz is about twice as slow than the Pentium II running at 300Mhz that we used for our hMETiS-Kway experiments). Taking this CPU performance difference into account, we see that hMETiS-Kway is 5 times faster for the 8-way partitioning, and 4.7 times faster for the 16-way partitioning. Thus, compared to K-PM/LR, hMETiS-Kway not only cuts substantially fewer hyperedges but it is also significantly faster than K-PM/LR.

Finally, Cong and Lim [19] also reported results using the minimization of the $(K-1)$ metric as the objective function of K-PM/LR. Table 5 shows the cost of the solutions with respect to the $(K-1)$ metric obtained by both hMETiS-Kway and K-PM/LR for an 8- and a 16-way partitioning. Note that for hMETiS-Kway, the value for the $(K-1)$ metric was obtained by performing the partitioning using the minimization of the SOED as the objective. The data in Table 5 was also used to plot the bar-charts shown in Figure 6 that compares the cost of the partitions obtained by hMETiS-Kway relative to those obtained by K-PM/LR. From this figure we can see that hMETiS-Kway also produces partitionings that are consistently and significantly better than those produced by K-PM/LR. In particular, the $(K-1)$-metric cost of hMETiS-Kway is, on the average, 15% and 14% smaller than the cost of K-PM/LR for the 8- and 16-way partitionings, respectively.

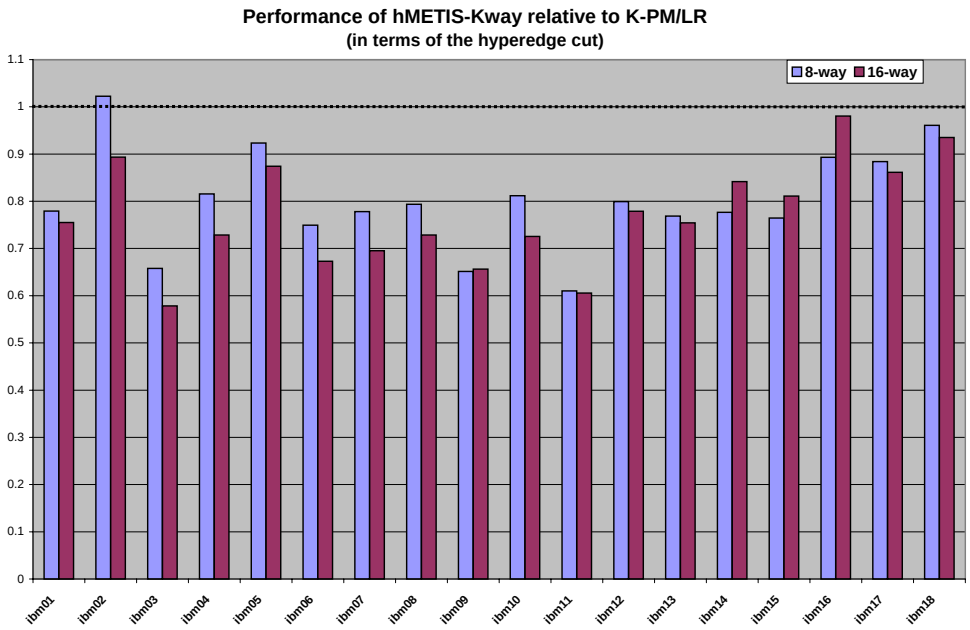


Figure 5: The quality of the partitionings in terms of the cuts produced by hMETIS-Kway relative to those produced by K-PM/LR for an 8- and 16-way partitioning. Bars bellow the 1.0 line indicate that hMETIS-Kway performs better than K-PM/LR.

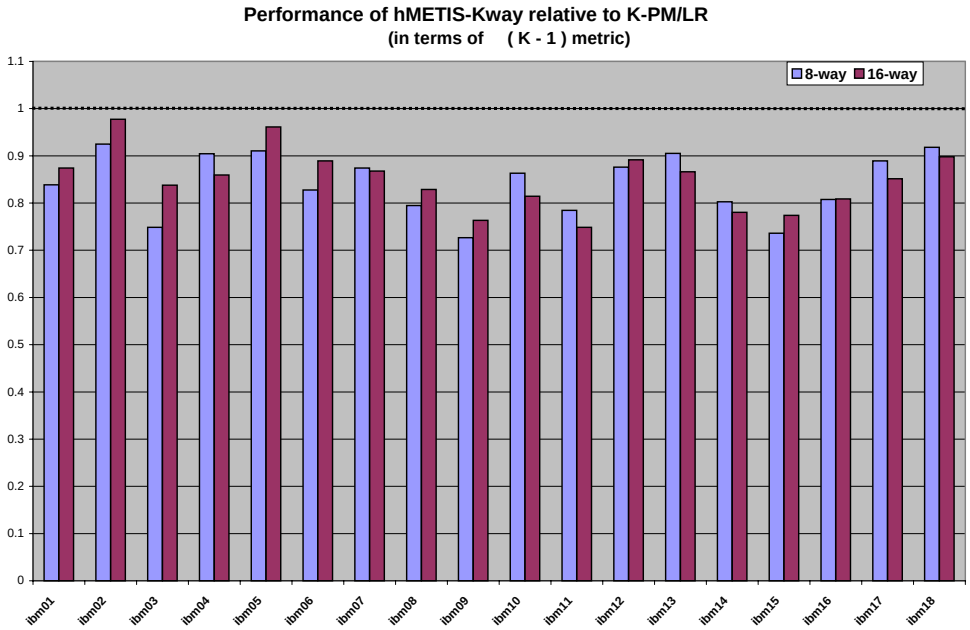


Figure 6: The quality of the partitionings in terms of the $(K-1)$ metric produced by hMETIS-Kway relative to those produced by K-PM/LR for an 8- and 16-way partitioning. Bars bellow the 1.0 line indicate that hMETIS-Kway performs better than K-PM/LR.

Circuit	hMETiS-Kway		K-PM/LR	
	8-way	16-way	8-way	16-way
ibm01	930	1592	1109	1821
ibm02	1750	4058	1892	4152
ibm03	3083	4745	4119	5662
ibm04	3320	4956	3671	5766
ibm05	5958	8982	6543	9344
ibm06	3300	5248	3988	5900
ibm07	4115	5948	4707	6854
ibm08	4312	6102	5426	7364
ibm09	3043	4564	4187	5978
ibm10	4763	6944	5518	8525
ibm11	4174	6303	5321	8420
ibm12	6598	9358	7530	10495
ibm13	3319	6394	3667	7382
ibm14	5962	9734	7427	12476
ibm15	8104	11182	11008	14448
ibm16	7529	12052	9322	14901
ibm17	10510	17740	11818	20830
ibm18	6410	10498	6982	11692
ARQ	0.840	0.849	1.191	1.178

Table 5: The $(K-1)$ metric of the partitionings obtained by hMETiS-Kway and the K-PM/LR partitioning algorithms for 8- and 16-way partitionings. The row labeled 'ARQ' shows the Average Relative Quality of one scheme versus the other. For example, the ARQ value of 1.191 for the 8-way partitioning of K-PM/LR means that the $(K-1)$ metric solutions produced by K-PM/LR are on the average 19.1% higher (worse) than the corresponding solution produced by hMETiS-Kway. An ARQ value that is less than 1.0 indicates that the particular scheme on the average performs better.

4 Conclusions

The multilevel k-way partitioning scheme presented in this paper substantially outperforms the state-of-the-art K-PM/LR algorithm for multi-way partitioning [19] both for minimizing the hyperedge cut as well as minimizing the $(K-1)$ metric. The power of hMETiS-Kway is primarily derived from the robustness of the multilevel paradigm that allows the use of a simple k-way partitioning refinement heuristic instead of the $O(k^2)$ complexity k-way FM refinement [4] or a sequence of pair-wise FM refinements [19]. The simple k-way refinement heuristic is able to perform an excellent job in optimizing the objective function, as it is applied to successively finer hypergraphs. Furthermore, as our experiments indicate, the multilevel k-way paradigm offers the additional benefit of producing high quality partitionings while enforcing tight balancing constraints.

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] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, pages 62–81, 1989.

[5] C. W. Yeh, C. K. Cheng, and T. T. Lin. A general purpose multiple-way partitioning algorithm. In *Proc. of the Design Automation Conference*, pages 421–426, 1991.

[6] P. Chan, M. Schlag, and J. Zien. Spectral k-way ratio-cut partitioning and clustering. In *Proc. of the Design Automation Conference*, pages 749–754, 1993.

[7] L. A. Sanchis. Multiple-way network partitioning with different cost functions. *IEEE Transactions on Computers*, pages 1500–1504, 1993.

[8] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? Technical Report RNR-93-012, NAS Systems Division, NASA, Moffet Field, CA, 1993.

[9] C. J. Alpert and A. B. Kahng. Multi-way partitioning via space-filling curves and dynamic programming. In *Proc. of the Design Automation Conference*, pages 652–657, 1994.

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

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

[12] J. Cong, W. Labio, and N. Shivakumar. Multi-way VLSI circuit partitioning based on dual net representation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 396–409, 1996.

[13] 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.

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

[15] C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *Proc. of the 34th ACM/IEEE Design Automation Conference*, 1997.

[16] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *Proceedings of the Design and Automation Conference*, 1997.

[17] George Karypis and Vipin Kumar. A coarse-grain parallel multilevel k-way partitioning algorithm. In *Proceedings of the eighth SIAM conference on Parallel Processing for Scientific Computing*, 1997.

[18] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. of the Intl. Symposium of Physical Design*, pages 80–85, 1998.

[19] Jason Cong and Sung Kyu Lim. Multiway Partitioning with Pairwise Movement. In *Intl. Conference on Computer Aided Design*, 1998.

[20] G. Karypis and V. Kumar. hMeTiS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

[21] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Supercomputing*, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[22] Sverre Wichlund and Einar J. Aas. On Multilevel Circuit Partitioning. In *Intl. Conference on Computer Aided Design*, 1998.

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

[24] 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.

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

[26] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. *IEEE Transactions on VLSI Systems*, 1998 (to appear). A short version appears in the proceedings of DAC 1997.