hMETIS*

A Hypergraph Partitioning Package

Version 1.5.3

George Karypis and Vipin Kumar

University of Minnesota, Department of Computer Science & Engineering
Army HPC Research Center
Minneapolis, MN 55455

{karypis, kumar}@cs.umn.edu

November 22, 1998

Metis [MEE tis]: *Metis was a titaness in Greek mythology. She was the consort of Zeus and the mother of Athena. She presided over all wisdom and knowledge.*

*hMETIS is copyrighted by the regents of the University of Minnesota. This work was supported by IST/BMDO through Army Research Office contract DA/DAAH04-93-G-0080, and by 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 were provided by Minnesota Supercomputer Institute, Cray Research Inc. Related papers are available via WWW at URL: **http://www.cs.umn.edu/˜karypis**

Contents

1 Introduction

Hypergraph partitioning is an important problem and has extensive applications in many areas, including VLSI design [2], efficient storage of large databases on disks [13], transportation management, and data-mining [5]. 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.

2 What is hMETIS

hMETIS is a software package for partitioning large hypergraphs, especially those arising in circuit design. The algorithms in hMETIS are based on multilevel hypergraph partitioning described in [10, 11, 7], and they are an extension of the widely used METIS graph partitioning package described in [9, 8]. Traditional graph partitioning algorithms compute a partition of a graph by operating directly on the original graph as illustrated in Figure 1(a). These algorithms are often too slow and/or produce poor quality partitions. Multilevel partitioning algorithms, on the other hand, take a completely different approach[6, 9, 8, 10]. These algorithms, as illustrated in Figure 1(b), reduce the size of the graph (or hypergraph) by collapsing vertices and edges (during the coarsening phase), partition the smaller graph (initial partitioning phase), and then uncoarsen it to construct a partition for the original graph (uncoarsening and refinement phase). hMETIS uses novel approaches to successively reduce the size of the hypergraph as well as to further refine the partition during the uncoarsening phase. During coarsening, hMETIS employs algorithms that make it easier to find a high-quality partition at the coarsest graph. During refinement, hMETIS focuses primarily on the portion of the graph that is close to the partition boundary. These highly tuned algorithms allow hMETIS to quickly produce high-quality partitions for a large variety of hypergraphs.

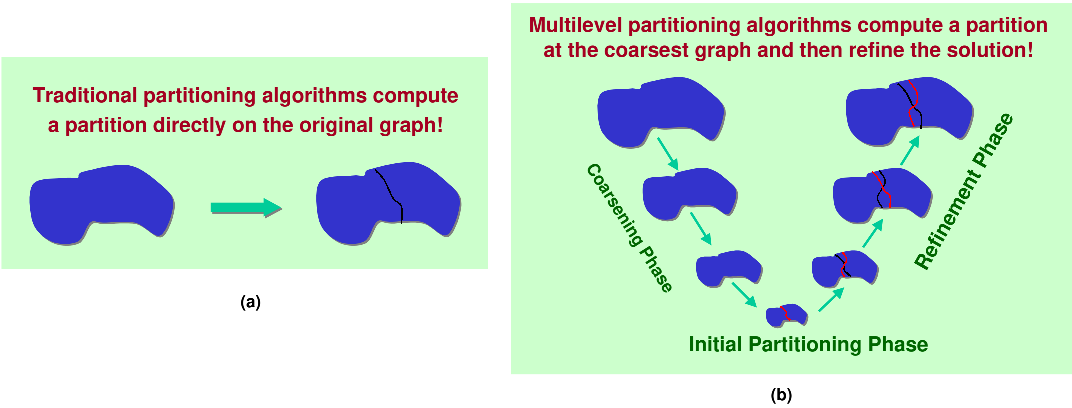

Figure 1: (a) Traditional partitioning algorithms. (b) Multilevel partitioning algorithms.

The advantages of hMETIS compared to other similar algorithms are the following:

☞ **Provides high quality partitions!**
Experiments on a large number of hypergraphs arising in various domains including VLSI, databases, and data mining show that hMETIS produces partitions that are consistently better than those produced by other widely used algorithms, such as KL, FM, LA, PROP, CLIP, *etc.*.

☞ **It is extremely fast!**
Experiments on a wide range of hypergraphs has shown that hMETIS is one to two orders of magnitude faster than other widely used partitioning algorithms. hMETIS can produce extremely high quality bisections of hypergraphs with 100,000 vertices in well under 3 minutes on an R10000-based SGI workstation and a Pentium Pro-based personal computer.

2.1 Overview of the Algorithms used in hMETIS

In the rest of this section, we briefly describe the various phases of the multilevel algorithm. The reader should refer to [10] for further details.

Coarsening Phase During the hypergraph coarsening phase, a sequence of successively smaller hypergraphs is constructed. 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 size 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 Kernighan-Lin algorithm [12, 4] are very effective in refining small hyperedges but are quite ineffective in refining hyperedges with a large number of vertices belonging to different partitions. The group of vertices that are contracted together to form single vertices in the next level coarse hypergraph can be selected in different ways. hMETIS implements various such grouping schemes (also called *matching schemes*) some of which are described in [10].

Initial Partitioning phase During the initial partitioning phase, a bisection of the coarsened hypergraph is computed. Since this hypergraph has a very small number of vertices (usually less than 100 vertices) many different algorithms can be used without significantly affecting the overall runtime and quality of the algorithm. hMETIS uses multiple random bisections followed by the Fiduccia-Mattheyses(FM) refinement algorithm.

Uncoarsening and refinement phase During the uncoarsening phase, the partitioning of the coarsest hypergraph is used to obtain a partitioning for the finer hypergraph. This is done by successively projecting the partitioning to the next level finer hypergraph and using a partitioning refinement algorithm to reduce the cut and thus improve the quality of the partitioning. Since the next level finer hypergraph has more degrees of freedom, such refinement algorithms tend to improve the quality. hMETIS implements a variety of algorithms that are based on the FM algorithm [4]. The details of some of these schemes can be found in [10].

V-Cycle Refinement The idea behind this refinement algorithm is to use the power of the multilevel paradigm to further improve the quality of a bisection. The V-cycle refinement algorithm consists of two phases, namely a coarsening and an uncoarsening phase. The coarsening phase preserves the initial partitioning that is input to the algorithm. We will refer to this as *restricted coarsening* scheme. In this restricted coarsening scheme, the groups of vertices that are combined to form the vertices of the coarse graphs correspond to vertices that belong only to one of the two partitions. As a result, the original bisection is preserved through out the coarsening process, and becomes the initial partition from which we start performing refinement during the uncoarsening phase. The uncoarsening phase of the V-cycle refinement algorithm is identical to the uncoarsening phase of the multilevel hypergraph partitioning algorithm described earlier. It moves vertices between partitions as long as such moves improve the quality of the bisection. Note that the various coarse representations of the original hypergraph, allow refinement to further improve the quality as it helps it climb out of local minima.

3 hMETIS's Stand-Alone Programs

hMETIS provides the shmetis, hmetis, and khmetis programs that can be used to partition a hypergraph into k parts. The first two programs (shmetis and hmetis) compute a k-way partitioning using multilevel recursive bisection [10]. The shmetis program is suited for those users who want to use hMETIS without getting into the details of the underlying algorithms, while hmetis is suited for those users that want to experiment with the various algorithms used by hMETIS. Both shmetis and hmetis can also compute a k-way partitioning when certain vertices of the hypergraph have pre-assigned partitions (*i.e.*, there are at most k sets of vertices each fixed to a particular partition).

The third program (khmetis) computes a k-way partitioning using multilevel k-way partitioning [8]. This is a new feature of hMETIS 1.5, and the underlying algorithms are still under development.

3.1 shmetis

The shmetis program is invoked by providing three or four arguments at the command line as follows:

shmetis *HGraphFile* *Nparts* *UBfactor*

or

shmetis *HGraphFile* *FixFile* *Nparts* *UBfactor*

The meaning of the various parameters is as follows:

HGraphFile
This is the name of the file that stores the hypergraph (the format is described in Section 3.4).

FixFile This is the name of the file that stores information about the pre-assignment of vertices to partitions (the format is described in Section 3.5).

Nparts This is the number of desired partitions. shmetis can partition a hypergraph into an arbitrary number of partitions, using recursive bisection. That is, for a 4-way partition, shmetis first computes a 2-way partition of the original hypergraph, then constructs two smaller hypergraphs, each corresponding to one of the two partitions, and then computes 2-way partitions of these smaller hypergraphs to obtain the desired 4-way partition[1]. Note that shmetis, while constructing the smaller hypergraphs, completely removes the hyperedges that were cut during the bisection[2].

UBfactor This parameter is used to specify the allowed imbalance between the partitions during recursive bisection. This is an integer number between 1 and 49, and specifies the allowed load imbalance in the following way. Consider a hypergraph with n vertices, each having a unit weight, and let b be the *UBfactor*. Then, if the number of desired partitions is two (*i.e.*, we perform a bisection), then the number of vertices assigned to each one of the two partitions will be between $(50-b)n/100$ and $(50+b)n/100$. For example, for $b=5$, then we will be allowing a 45-55 bisection, that is, the number of vertices in each partition will be between $0.45n$ and $0.55n$. Note that this allowed imbalance is applied at each bisection step, so if instead of a 2-way partition we are interested in a 4-way partition, then a *UBfactor* of 5 will result in partitions that can contain between $0.45^2n = 0.20n$ and $0.55^2n = 0.30n$ vertices. Also note that shmetis does not allow you to produce perfectly balanced partitions. This is a limitation that will be lifted in future releases.

Upon successful execution, shmetis displays statistics regarding the quality of the computed partitioning and the amount of time taken to perform the partitioning (the times are shown in seconds). The actual partitioning is stored in a file named ***HGraphFile.part.Nparts***, whose format is described in Section 3.6.

Figure 2 shows the output of shmetis for partitioning a hypergraph into four parts. From this figure we see that shmetis initially prints information about the hypergraph, such as its name, the number of vertices (*#Vtxs*), the number of hyperedges (*#Hedges*), and also the number of desired partitions (*#Parts*) and allowed imbalance (*UBfactor*). Next, prints information about the different bisections that were computed. In this example, since we asked for four partitions, the algorithm computes a total of three bisections, and for each one prints information regarding the size of the hypergraph that is bisected and the quality of the computed bisections. In particular, with respect to quality, it prints the minimum and average number of cuts, and also the balance corresponding to the minimum cut.

The overall quality of the obtained partitioning is summarized by computing the following quality measures (in the case of hypergraphs with weighted hyperedges, these definitions are extended in a straight-forward manner):

1. *Hyperedge Cut* This is the number of the hyperedges that span multiple partitions. The partitioning routines in hMETIS try to directly minimize this quantity.

[1] shmetis can handle non-power of 2 partitions, by performing unbalanced bisections. That is, for a 3-way partition it computes a 2-way partition such that the first part has 2/3 of the total number of vertices, and the other part has 1/3. It then it bisects the first part into two equal-size parts, each containing 1/3 of the original number of vertices.

[2] The hmetis program allows you to change this behavior.

2. *Sum of External Degrees* The external degree $|E(P_i)|$ of a partition P_i, is defined as the number of hyperedges, that are incident but not fully inside this partition. The sum of the external degrees of a k-way partitioning, is then $\sum_{i=1}^{k} |E(P_i)|$.

3. *Scaled Cost* This is defined as
$$\frac{1}{n(k-1)} \sum_{i=1}^{k} \frac{|E(P_i)|}{w(P_i)},$$
where $w(P_i)$ is the sum of the vertex weights of partition P_i (note that if the vertices do not have weights, then $w(P_i) = |P_i|$).

4. *Absorption* This is defined as
$$\sum_{i=1}^{k} \sum_{e \in E | e \cap P_i \neq \emptyset} \frac{|e \cap P_i| - 1}{|e| - 1}$$
where E is the set of hyperedges, $|e \cap P_i|$ is the number of vertices of hyperedge e that are also in partition P_i, and $|e|$ is the number of vertices in the hyperedge e.

Following these quality measures, shmetis prints the size of the various partitions as well as the external degrees of each partition. Finally, it shows the time taken by the various phases of the algorithm. All times are in seconds.

```
prompt% shmetis ibm02.hgr 4 5

*******************************************************************************
 HMETIS 1.5.3  Copyright 1998, Regents of the University of Minnesota

HyperGraph Information -----------------------------------------------------
 Name: ibm02.hgr, #Vtxs: 19601, #Hedges: 19584, #Parts: 4, UBfactor: 0.05
 Options: HFC, FM, Reconst-False, V-cycles @ End, No Fixed Vertices

Recursive Partitioning... --------------------------------------------------

  Bisecting a hgraph of size [vertices=19601, hedges=19584, balance=0.50]
    The mincut for this bisection = 262, (average = 277.8) (balance = 0.46)

  Bisecting a hgraph of size [vertices=9028, hedges=8501, balance=0.50]
    The mincut for this bisection = 186, (average = 241.4) (balance = 0.49)

  Bisecting a hgraph of size [vertices=10573, hedges=10821, balance=0.50]
    The mincut for this bisection = 192, (average = 193.5) (balance = 0.47)

 ---------------------------------------------------------------------------
  Summary for the 4-way partition:
                Hyperedge Cut:       619               (minimize)
      Sum of External Degrees:      1305               (minimize)
                  Scaled Cost:  4.56e-06               (minimize)
                   Absorption:  19336.20               (maximize)

      Partition Sizes & External Degrees:
          4669[ 382]   4303[ 276]   5048[ 338]   5581[ 309]

Timing Information ---------------------------------------------------------
  Partitioning Time:             73.340sec
           I/O Time:              0.230sec
*******************************************************************************
```

Figure 2: Output of shmetis for *ibm02.hgr* and a 4-way partition

3.2 hmetis

The program hmetis is invoked by providing 9 or 10 command line arguments as follows:

hmetis *HGraphFile Nparts UBfactor Nruns CType RType Vcycle Reconst dbglvl*

or

hmetis *HGraphFile FixFile Nparts UBfactor Nruns CType RType Vcycle Reconst dbglvl*

The meaning of the various parameters is as follows:

HGraphFile, FixFile, Nparts, UBfactor
The meaning of these parameters is identical to those of shmetis.

Nruns This is the number of the different bisections that are performed by hmetis. It is a number greater or equal to one, and instructs hmetis to compute *Nruns* different bisections, and select the best as the final solution. A default value of 10 is used by shmetis.

Section 5.2.1 provides an experimental evaluation of the effect of Nruns in the quality of k-way partitionings.

CType This is the type of vertex grouping scheme (*i.e.*, matching scheme) to use during the coarsening phase. It is an integer parameter and the possible values are:

1 Selects the hybrid first-choice scheme (HFC). This scheme is a combination of the first-choice and greedy first-choice scheme described later. This is the scheme used by shmetis.

2 Selects the first-choice scheme (FC). In this scheme vertices are grouped together if they are present in multiple hyperedges. Groups of vertices of arbitrary size are allowed to be collapsed together.

3 Selects the greedy first-choice scheme (GFC). In this scheme vertices are grouped based on the first-choice scheme, but the grouping is biased in favor of faster reduction in the number of the hyperedges that remain in the coarse hypergraphs.

4 Selects the hyperedge scheme. In this scheme vertices are grouped together that correspond to entire hyperedges. Preference is given to hyperedges that have large weight.

5 Selects the edge scheme. In this scheme pairs of vertices are grouped together if they are connected by multiple hyperedges.

You may have to experiment with this parameter to see which scheme works better for the classes of hypergraphs that you are using. Section 5.1.1 provides an experimental evaluation of the various values of CType for a range of hypergraphs.

RType This is the type of refinement policy to use during the uncoarsening phase. It is an integer parameter and the possible values are:

1 Selects the Fiduccia-Mattheyses (FM) refinement scheme. This is the scheme used by shmetis.

2 Selects the one-way Fiduccia-Mattheyses refinement scheme. In this scheme, during each iteration of the FM algorithm, vertices are allowed to move only in a single direction.

3 Selects the early-exit FM refinement scheme. In this scheme, the FM iteration is aborted if the quality of the solution does not improve after a relatively small number of vertex moves.

Experiments have shown that FM and one-way FM produce better results than early-exit FM. However, early-exit FM is considerably faster, and the overall quality is not significantly worse. Section 5.1.2 provides an experimental evaluation of the various values of RType for a range of hypergraphs.

Vcycle This parameter selects the type of V-cycle refinement to be used by the algorithm. It is an integer parameter and the possible values are:

0 Does not perform any form of V-cycle refinement.

1 Performs V-cycle refinement on the final solution of each bisection step. That is, only the best of the *Nruns* bisections are refined using V-cycles. This is the options used by shmetis.

2 Performs V-cycle refinement on each intermediate solution whose quality is equally good or better than the best found so far. That is, as hmetis computes *Nruns* bisections, for each bisection that matches or improves the best one, it is also further refined using V-cycles.

3 Performs V-cycle refinement on each intermediate solution. That is, each one of the *Nruns* bisections is also refined using V-cycles.

Experiments have shown that the second and third choices offer the best time/quality tradeoffs. If time is not an issue, the fourth choice (*i.e.*, Vcycle = 3) should be used.

Reconst This parameter is used to select the scheme to be used in dealing with hyperedges that are being cut during the recursive bisection. It is an integer parameter and the possible values are:

0 This scheme removes any hyperedges that were cut while constructing the two smaller hypergraphs in the recursive bisection step. In other words, once a hyperedge is being cut, it is removed from further consideration. Essentially this scheme focuses on minimizing the number of hyperedges that are being cut. This is the scheme that is used by shmetis.

1 This scheme reconstructs the hyperedges that are being cut, so that each of the two partitions retain the portion of the hyperedge that corresponds to its set of vertices.

Section 5.2.2 provides an experimental evaluation of the effect of Reconst in the quality of k-way partitionings.

dbglvl This is used to request hMETIS to print debugging information. The value of *dbglvl* is computed as the sum of codes associated with each option of hmetis. The various options and their values are as follows:

0 Show no additional information.

1 Show information about the coarsening phase.

2 Show information about the initial partitioning phase.

4 Show information about the refinement phase.

8 Show information about the multiple runs.

16 Show additional information about the multiple runs.

For example, if we want to see all information about the multiple runs the value of *dbglvl* should be $8 + 16 = 24$. Note that some of the options may generate a lot of output. Use them with caution.

Upon successful execution, hmetis displays statistics regarding the quality of the computed partitioning and the amount of time taken to perform the partitioning. The actual partitioning is stored in a file named ***HGraphFile.part.Nparts***, whose format is described in Section 3.6. Figure 3 shows the output of hmetis for a 2-way partition.

3.3 khmetis

The khmetis program is invoked by providing 7 command line arguments as follows:

khmetis *HGraphFile Nparts UBfactor Nruns CType OType Vcycle dbglvl*

The meaning of the various parameters is as follows:

HGraphFile, Nparts, Nruns, CType, Vcycle, dbglvl
The meaning of these parameters is identical to those of hmetis.

```
prompt% hmetis ibm03.hgr 2 5 10 1 1 3 0 24

*******************************************************************************
 HMETIS 1.5.3  Copyright 1998, Regents of the University of Minnesota

HyperGraph Information -----------------------------------------------------
 Name: ibm03.hgr, #Vtxs: 23136, #Hedges: 27401, #Parts: 2, UBfactor: 0.05
 Options: HFC, FM, Reconst-False, Always V-cycle, No Fixed Vertices

Recursive Partitioning... --------------------------------------------------

  Bisecting a hgraph of size [vertices=23136, hedges=27401, balance=0.50]
    Cut of trial    0:      979 [0.50]
    Cut of trial    1:      957 [0.46]
    Cut of trial    2:      979 [0.50]
    Cut of trial    3:      982 [0.48]
    Cut of trial    4:     1010 [0.47]
    Cut of trial    5:      956 [0.46]
    Cut of trial    6:      990 [0.50]
    Cut of trial    7:      957 [0.46]
    Cut of trial    8:     1142 [0.48]
    Cut of trial    9:      956 [0.46]
    The mincut for this bisection = 956, (average = 990.8) (balance = 0.46)

 --------------------------------------------------------------------------
  Summary for the 2-way partition:
                Hyperedge Cut:       956                (minimize)
      Sum of External Degrees:      1912                (minimize)
                  Scaled Cost:  7.18e-06                (minimize)
                   Absorption:  27029.76                (maximize)

      Partition Sizes & External Degrees:
         12419[ 956]  10717[ 956]

Timing Information -----------------------------------------------------
  Partitioning Time:             85.190sec
         I/O Time:                0.280sec
*******************************************************************************
```

Figure 3: Output of hmetis for *ibm03.hgr* and a 2-way partition

UBfactor This parameter is used to specify the allowed imbalance between the k partitions. This is an integer greater than 5 and specifies the allowed load imbalance as follows. A value of b for *UBfactor* indicates that the weight of the heaviest partition should not be more than $b\%$ greater than the average weight. For example, for $b = 8$, $k = 5$, and a hypergraph with n vertices (each having unit vertex weight), the weight of the heaviest partition will be bounded from above by $1.08 * n/5$. Note that this specification of the allowed imbalance between the k partitions is **different** from the specification used by either shmetis or hmetis.

OType This determines which objective function the refinement algorithm tries to minimize. It is an integer parameter and the possible values are:

1 Minimizes the hyperedge cut.

2 Minimizes the sum of external degrees (SOED).

This feature was introduced with version 1.5.3.

Upon successful execution, khmetis displays statistics regarding the quality of the computed partitioning and the amount of time taken to perform the partitioning. The actual partitioning is stored in a file named ***HGraph-File.part.Nparts***, whose format is described in Section 3.6. Figure 4 shows the output of khmetis for a 10-way partitioning.

```
prompt% khmetis ibm04.hgr 10 10 10 1 1 2 24

*******************************************************************************
 HMETIS 1.5.3  Copyright 1998, Regents of the University of Minnesota

HyperGraph Information -----------------------------------------------------
 Name: ibm04.hgr, #Vtxs: 27507, #Hedges: 31970, #Parts: 10, UBfactor: 1.10
 Options: HFC, Cut-minimization, V-cycle for Min

K-way Partitioning... -----------------------------------------------------

  Partitioning a hgraph of size [vertices=27507, hedges=31970, balance=1.10]
    Cut/SOED of trial    0:        3259   7333 [1.10]
    Cut/SOED of trial    1:        3498   7946 [1.09]
    Cut/SOED of trial    2:        3397   7728 [1.10]
    Cut/SOED of trial    3:        3192   7242 [1.10]
    Cut/SOED of trial    4:        3277   7283 [1.10]
    Cut/SOED of trial    5:        3314   7555 [1.07]
    Cut/SOED of trial    6:        3390   7554 [1.10]
    Cut/SOED of trial    7:        3414   7723 [1.06]
    Cut/SOED of trial    8:        3307   7357 [1.10]
    Cut/SOED of trial    9:        3322   7433 [1.10]
    The mincut for this partitioning = 3192, (average = 3337.0) (balance = 1.10)

 -------------------------------------------------------------------------
  Summary for the 10-way partition:
                Hyperedge Cut:      3192                (minimize)
      Sum of External Degrees:      7242                (minimize)
                  Scaled Cost:  1.06e-05                (minimize)
                   Absorption:  30250.46                (maximize)

      Partition Sizes & External Degrees:
          2504[ 701]   2796[ 515]   2728[ 634]   2836[1092]   3020[1007]
          2686[ 794]   2662[ 549]   2706[ 740]   2906[ 508]   2663[ 702]


Timing Information -----------------------------------------------------
  Partitioning Time:            136.720sec
          I/O Time:               0.310sec
*******************************************************************************
```

Figure 4: Output of khmetis for *ibm04.hgr* and a 10-way partition

Note that khmetis should never be used to compute a bisection (*i.e.*, 2-way partitioning) as it produces worse results than hmetis. Furthermore, the quality of the partitionings produced by khmetis for small values of k will be worse, in general, than the corresponding partitionings computed by hmetis. However, khmetis is particularly useful for computing k-way partitionings for relatively large values of k, as it often produces better partitionings and it can also enforce tighter balancing constraints.

3.4 Format of Hypergraph Input File

The primary input of hMETIS is the hypergraph to be partitioned. This hypergraph is stored in a file and is supplied to hMETIS as one of the command line parameters. A hypergraph $H = (V, E^h)$ with V vertices and E^h hyperedges is stored in a plain text file that contains $|E^h| + 1$ lines, if there are no weights on the vertices and $|E^h| + |V| + 1$ lines if there are weights on the vertices. Any line that starts with '%' is a comment line and is skipped.

The first line contains either two or three integers. The first integer is the number of hyperedges ($|E^h|$), the second is the number of vertices ($|V|$), and the third integer (*fmt*) contains information about the type of the hypergraph. In particular, depending on the value of *fmt*, the hypergraph H can have weights on either the hyperedges, the vertices, or both. In the case that H is unweighted (*i.e.*, all the hyperedges and vertices have the same weight), *fmt* is omitted.

After this first line, the remaining $|E^h|$ lines store the vertices contained in each hyperedge–one line per hyperedge. In particular, the ith line (excluding comment lines) contains the vertices that are included in the $(i-1)$th hyperedge. This format is illustrated in Figure 5(a). Weighted hyperedges are specified as shown in Figure 5(b). The first integer in each line contains the weight of the respective hyperedge. Note, hyperedge weights are integer quantities. Furthermore, note that the *fmt* parameter is equal to 1, indicating the fact that H has weights on the hyperedges. Finally, weights on the vertices are also allowed, as illustrated in Figure 5(c). In this case, $|V|$ lines are appended to the input file containing the weight of the $|V|$ vertices. Note that the value of *fmt* is equal to 10. As was the case with hyperedge weights, vertex weights are integer quantities. Figure 5(d) shows the case when both the hyperedges and the vertices are weighted. *fmt* in this case is equal to 11.

Figure 5 shows the *HGraphFile* expected by hMETIS for the example hypergraphs shown in the figure. It shows the four cases in which the hypergraph is unweighted, has weighted hyperedges, has weighted vertices and has both hyperedges and vertices weighted. The hypergraph shown in Figure 5(a) has four unweighted hyperedges a, b, c, and d. Number of vertices in the hypergraph is 7. When the hypergraph is unweighted, first line of the *HGraphFile* contains two integers denoting the number of hyperedges and the number of the vertices in the hypergraph. After that, each line corresponds to a hyperedge containing an entry for each vertex in the hyperedge. Hypergraph shown in Figure 5(b) has hyperedge weights equal to 2, 3, 7, and 8 on each of the hyperedge a, b, c, and d respectively. For this weighted hyperedges first line of the *HGraphFile* consists of three integers. Third integer specify that the hyperedges are weighted and is equal to 1. Each line corresponding to each hyperedge, has first entry equal to its weight. The following entries corresponds to the vertices in the respective hyperedge. The case when both the vertices are weighted *fmt* is equal to 10, and 7 lines corresponding to the 7 vertices are appended to the input file each containing weight of the respective vertex. This is shown in Figure 5(c). Figure 5(d) shows the case when both the hyperedges and the vertices are weighted.

3.5 Format of the Fix File

The *FixFile* is used to specify the vertices that are pre-assigned to certain partitions. In general, when computing a k-way partitioning, up to k sets of vertices can be specified, such that each set is pre-assigned to one of the k partitions. For a hypergraph with $|V|$ vertices, the *FixFile* consists of $|V|$ lines with a single number per line. The ith line of the file contains either the partition number to which the ith vertex is pre-assigned to, or -1 if that vertex can be assigned to any partition (*i.e.*, free to move). Note that the partition numbers start from 0.

3.6 Format of Output File

The output of hMETIS is a partition file. The partition file of a hypergraph with $|V|$ vertices, consists of $|V|$ lines with a single number per line. The ith line of the file contains the partition number that the ith vertex belongs to. Partition numbers start from 0. If `foo.graph` is the name of the file storing the hypergraph, the partition for a 2-way partition is stored in a file named `foo.graph.part.2`.

4 hMETIS's Library Interface

The hypergraph partitioning algorithms in hMETIS can also be accessed directly using the stand-alone library `libhmetis.a`. This library provides the HMETIS_PartRecursive() and HMETIS_PartKway() routines. The first routine corresponds to the hmetis whereas the second routine corresponds to the khmetis program. The calling sequences and the description of the various parameters of these routines are as follows:

4.1 HMETIS_PartRecursive

HMETIS_PartRecursive (int nvtxs, int nhedges, int *vwgts, int *eptr, int *eind, int *hewgts, int nparts, int ubfactor, int *options, int *part, int *edgecut)

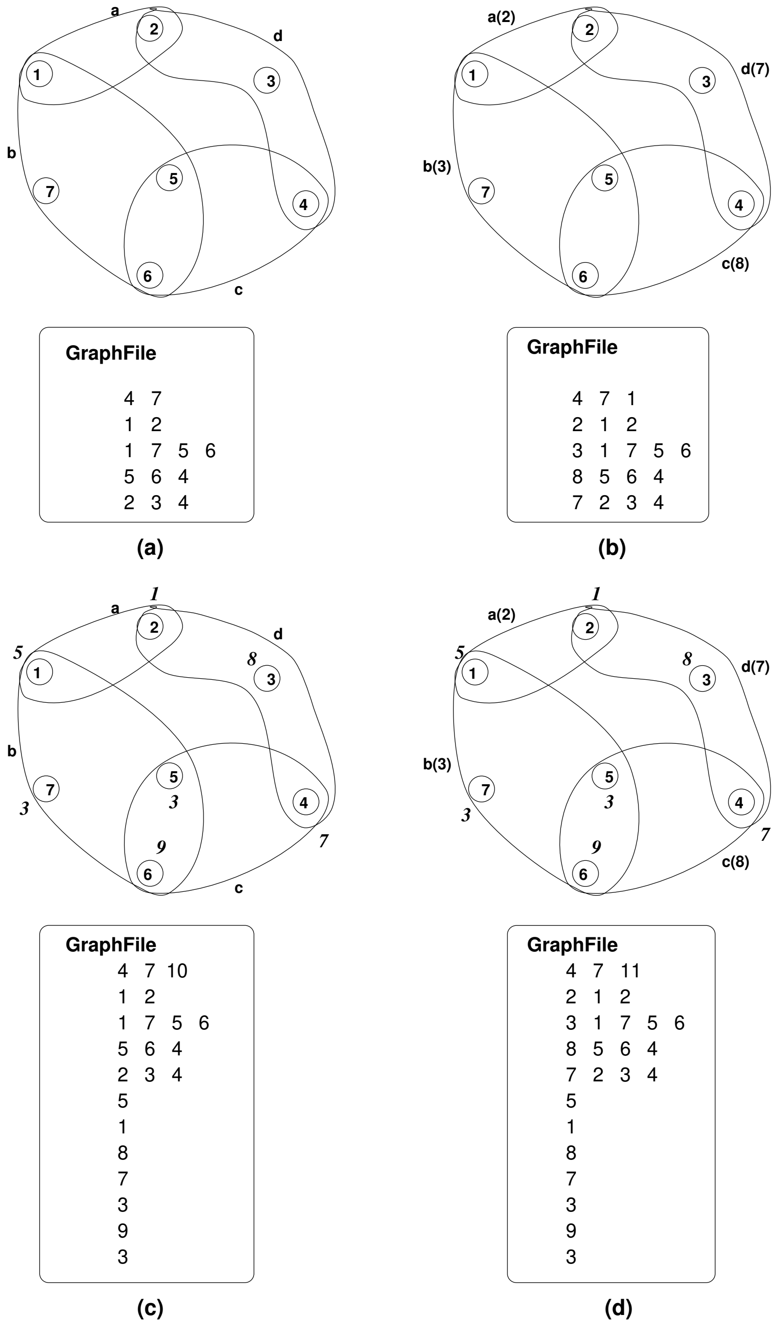

Figure 5: (a) *HGraphFile* for unweighted hypergraph, (b) *HGraphFile* for weighted hyperedges, (c) *HGraphFile* for weighted vertices, and (d) *HGraphFile* for weighted hyperedges and vertices

nvtxs, nhedges

The number of vertices and the number of hyperedges in the hypergraph, respectively.

vwgts An array of size *nvtxs* that stores the weight of the vertices. Specifically, the weight of vertex i is stored at *vwgts[i]*. If the vertices in the hypergraph are unweighted, then *vwgts* can be NULL.

eptr, eind

Two arrays that are used to describe the hyperedges in the graph. The first array, *eptr*, is of size *nhedges+1*, and it is used to index the second array *eind* that stores the actual hyperedges. Each hyperedge is stored as a sequence of the vertices that it spans, in consecutive locations in *eind*. Specifically, the ith hyperedge is stored starting at location *eind[eptr[i]]* up to (but not including) *eind[eptr[*$i+1$*]]*. Figure 6 illustrates this format for a simple hypergraph. The size of the array *eind* depends on the number and type of hyperedges. Also note that the numbering of vertices starts from 0.

hewgts An array of size *nhedges* that stores the weight of the hyperedges. The weight of the i hyperedge is stored at location *hewgts[i]*. If the hyperedges in the hypergraph are unweighted, then *hewgts* can be NULL.

nparts The number of desired partitions.

ubfactor This is the relative imbalance factor to be used at each bisection step. Its meaning is identical to the *UBfactor* parameter of shmetis, and hmetis described in Section 3.

options This is an array of 9 integers that is used to pass parameters for the various phases of the algorithm. If *options[0]=0* then default values are used. If *options[0]=1*, then the remaining elements of *options* are interpreted as follows:

options[1] Determines the number of different bisections that is computed at each bisection step of the algorithm. Its meaning is identical to the *Nruns* parameter of hmetis (described in Section 3.2).

options[2] Determines the scheme to be used for grouping vertices during the coarsening phase. Its meaning is identical to the *CType* parameter of hmetis (described in Section 3.2).

options[3] Determines the scheme to be used for refinement during the uncoarsening phase. Its meaning is identical to the *RType* parameter of hmetis (described in Section 3.2).

options[4] Determines the scheme to be used for V-cycle refinement. Its meaning is identical to the *Vcycle* parameter of hmetis (described in Section 3.2).

options[5] Determines the scheme to be used for reconstructing hyperedges during recursive bisections. Its meaning is identical to the *Reconst* parameter of hmetis (described in Section 3.2).

options[6] Determines whether or not there are sets of vertices that need to be pre-assigned to certain partitions. A value of 0 indicates that no pre-assignment is desired, whereas a value of 1 indicates that there are sets of vertices that need to be pre-assigned. In this later case, the parameter *part* is used to specify the partitions to which vertices are pre-assigned. In particular, *part[i]* will store the partition number that vertex i is pre-assigned to , and -1 if it is free to move.

options[7] Determines the random seed to be used to initialize the random number generator of hMETIS. A negative value indicates that a randomly generated seed should be used (default behavior).

options[8] Determines the level of debugging information to be printed by hMETIS. Its meaning is identical to the *dbglvl* parameter of hmetis (described in Section 3.2). The default value is 0.

part This is an array of size *nvtxs* that returns the computed partition. Specifically, *part[i]* contains the partition number in which vertex i belongs to. Note that partition numbers start from 0.

Note that if *options[6] = 1*, then the initial values of *part* are used to specify the vertex pre-assignment requirements.

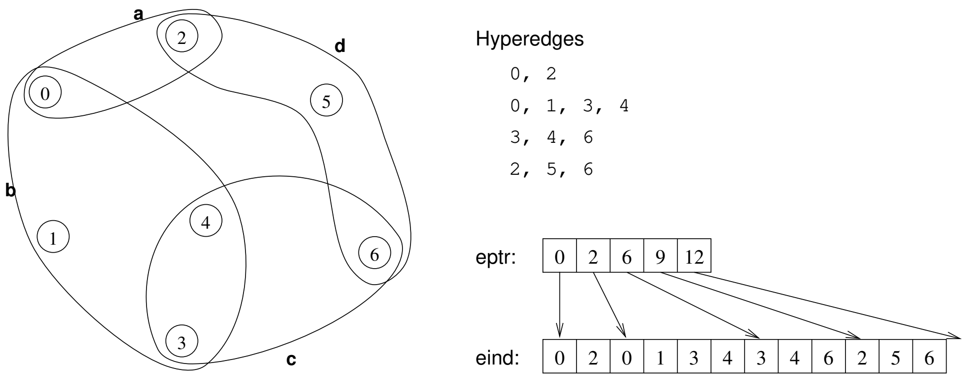

Figure 6: The *eptr* and *eind* arrays that are used to describe the hyperedges of the hypergraph.

edgecut This is an integer that returns the number of hyperedges that are being cut by the partitioning algorithm.

4.2 HMETIS_PartKway

HMETIS_PartKway (int nvtxs, int nhedges, int *vwgts, int *eptr, int *eind, int *hewgts, int nparts, int ubfactor, int *options, int *part, int *edgecut)

nvtxs, nhedges, vwgt, eptr, eind, hewgts, nparts
The meaning of these parameters is identical to meaning of the corresponding parameters of HMETIS_PartRecursive.

ubfactor This is the maximum load imbalance allowed in the k-way partitioning. Its meaning is identical to the *UBfactor* parameter of khmetis, Section 3.3.

options This is an array of 9 integers that is used to pass parameters for the various phases of the algorithm. If *options[0]=0* then default values are used. If *options[0]=1*, then the remaining elements of *options* are interpreted as follows:

- options[1] Determines the number of different k-way partitionings that is computed. Its meaning is identical to the *Nruns* parameter of khmetis (described in Section 3.3).
- options[2] Determines the scheme to be used for grouping vertices during the coarsening phase. Its meaning is identical to the *CType* parameter of khmetis (described in Section 3.3).
- options[3] Determines which objective function the partitioning algorithm tries to minimize. Its meaning is identical to the *OType* parameter of khmetis (described in Section 3.3). The default value is 1 (*i.e.*, minimize the hyperedge cut).
- options[4] Determines the scheme to be used for V-cycle refinement. Its meaning is identical to the *Vcycle* parameter of khmetis (described in Section 3.3).
- options[5] Not used.
- options[6] Not used.
- options[7] Determines the random seed to be used to initialize the random number generator of hMETIS. A negative value indicates that a randomly generated seed should be used (default behavior).
- options[8] Determines the level of debugging information to be printed by hMETIS. Its meaning is identical to the *dbglvl* parameter of khmetis (described in Section 3.3). The default value is 0.

part This is an array of size *nvtxs* that returns the computed partition. Specifically, *part[i]* contains the partition number in which vertex i belongs to. Note that partition numbers start from 0.

edgecut This is an integer that depending on the value of *options[3]* returns either the number of hyperedges that are being cut by the partitioning algorithm or the sum of the external degrees of the partitioning.

5 General Guidelines on How to Use hMETIS

5.1 Selecting the Proper Parameters

The hmetis program allows you to control the multilevel hypergraph bisection paradigm by providing a variety of algorithms for performing the various phases. In particular, it allows you to control:

1. How the vertices are grouped together during the coarsening phase. This is done by using the *CType* parameter.
2. How the quality of the bisection is refinement during the uncoarsening phase. This is done by using the *RType* parameter.

In designing the shmetis program, we had to make some choices for the above parameters. However, depending on the classes of the hypergraphs that are partitioned, these default settings may not necessarily be optimal. You should experiment with these parameters to see which schemes work better for your classes of problems.

In this section, we present an experimental evaluation of the various choices for *CType* and *RType* for various hypergraphs taken from the circuits of the ACM/SIGDA [3] and ISPD98 [1] benchmarks. The characteristics of these circuits are shown in Table 1. We hope that these experiments will help in illustrating the various quality and/or runtime trade-offs that are present in the various choices.

Circuit	No. of Vertices (*i.e.*, cells + pins)	No. of Hyperedges (*i.e.*, nets)
avqsmall	21918	22124
avqlarge	25178	25384
industry2	12637	13419
industry3	15406	21923
s35932	18148	17828
s38417	23949	23843
s38584	20995	20717
golem3	103048	144949
ibm01	12752	14111
ibm03	23136	27401
ibm05	29347	28446
ibm07	45926	48117
ibm09	53395	60902
ibm11	70558	81454
ibm13	84199	99666
ibm15	161570	186608
ibm17	185495	189581

Table 1: The characteristics of the various circuits used in the study of the various parameters of hMETIS.

5.1.1 Effect of the CType Parameter

Table 2 shows the quality of the bisections produced by hmetis for different vertex grouping schemes. The experiments in this table were performed by setting the remaining parameters of hmetis as follows: *Nruns = 20*, *UBfactor = 5*, *RType = 1*, *Vcycle = 1*, and *Reconst = 0*. For each different vertex grouping scheme, the column labeled "Min" shows the minimum cut out of the 20 trials, the column labeled "Avg" shows the average cut over all 20 trials, and the column labeled "Time" shows the overall amount of time required by hmetis (*i.e.*, the time to perform the 20 trials and the final V-cycle refinement).

As we can see from this table, different vertex grouping schemes perform better for different circuits. In general, the *HFC* scheme (that is used by default in shmetis) performs reasonably well for all the circuits (*i.e.*, it is within a few percentage points of the best scheme), but it is not necessarily the best. As this table suggests, one should experiment with the different vertex grouping schemes, to determine which one is suited for the classes of problems that she/he may have.

5.1.2 Effect of the RType Parameter

Table 3 shows the quality of the bisections produced by hmetis for different refinement schemes. The experiments in this table were performed by setting the remaining parameters of hmetis as follows: *Nruns = 20*, *UBfactor = 5*, *CType = 1*, *Vcycle = 1*, and *Reconst = 0*. For each different refinement scheme, the column labeled "Min" shows the minimum cut out of the 20 trials, the column labeled "Avg" shows the average cut over all 20 trials, and the column labeled "Time" shows the overall amount of time required by hmetis (*i.e.*, the time to perform the 20 trials and the final V-cycle refinement).

As we can see from this table, the three refinement schemes offer different quality/time trade-offs. In general, the EEFM scheme requires half the time required by either the FM or the 1WayFM schemes. Moreover, the quality of the bisections produced by EEFM, are in general only slightly worse (if any) than those produced by FM or 1WayFM. For example, in the 17 circuits of Table 3, EEFM performed significantly worse than the other two schemes only for *ibm15*. From the remaining two refinement schemes, the results of Table 3 suggest that they perform similarly with 1wayFM producing slightly better results and requiring somewhat less time.

5.2 Computing a k-way Partitioning

hMETIS can compute a k-way partitioning (for $k > 2$) using either the multilevel recursive bisection paradigm (implemented by hmetis) or the multilevel k-way partitioning paradigm (implemented by khmetis). In our discussion of khmetis (Section 3.3), we already provided some general guidelines as to when someone should use hmetis or khmetis. In general, when k is large (*e.g.*, $k > 16$) khmetis should be preferred over hmetis, as it is faster and enforces load imbalance constraints that are more natural than the bisection imbalance constraints enforced by hmetis.

In this section we focus our discussion on using hmetis to compute a k-way partitioning. In particular, besides the *CType* and *RType* parameters discussed in Section 5.1, the quality of the resulting k-way partitioning also depends on the choice of the *Nruns* and *Reconst* parameters.

5.2.1 Effect of the Nruns Parameter

Recall from Section 3.2, that *Nruns* is the number of different bisections that are computed by hmetis during each recursive bisection level. Out of these *Nruns* bisections, the one with the smallest cut is selected and used to bisect the hypergraph. For example, if *Nruns = 20*, then in the case of a 4-way partitioning, hmetis will first compute 20 bisections of the original hypergraph, and split it into two sub-hypergraphs based on the best bisection. Then, it will compute 20 bisections of each one of the two sub-hypergraphs, and again select the best solution for each one of the two sub-hypergraphs. However, an alternate approach of computing the 4-way partitioning (using the same overall number of different bisections), is to set *Nruns = 5*, run hmetis four times, and select the best 4-way partition out of these four solutions. That is, instead of running

```
hmetis xxx.hgr 4 5 20 1 1 1 0 0
```

we can run

```
hmetis xxx.hgr 4 5 5 1 1 1 0 0
hmetis xxx.hgr 4 5 5 1 1 1 0 0
hmetis xxx.hgr 4 5 5 1 1 1 0 0
hmetis xxx.hgr 4 5 5 1 1 1 0 0
```

and select the best solution. The overall amount of time for both approaches should be comparable (even though the second approach will be somewhat slower as the amount of time it spends in V-cycle refinement is four times higher). However, the quality of the solution obtained from the second approach may be better.

	HFC, CType=1			FC, CType=2			GFC, CType=3			HEDGE, CType=4			EDGE, CType=5		
Circuit	Min	Avg	Time	Min	Avg	Time	Min	Avg	Time	Min	Avg	Time	Min	Avg	Time
avqsmall	127	145.2	70.48	131	157.4	81.27	127	145.6	67.93	127	163.8	111.52	127	174.3	96.99
avqlarge	127	152.2	90.69	127	159.8	93.41	127	133.9	78.24	127	163.5	134.92	127	181.5	105.65
industry2	163	217.2	67.8	183	224.1	68.66	162	212.5	60.38	172	226.4	75.91	170	228.7	70.85
industry3	255	267.1	97.46	249	265.9	106.74	254	273.8	85.78	255	274.9	121.38	255	289.2	109.94
s35932	43	43.6	46.18	43	43.4	46.81	73	73.0	41.69	43	51.4	61.50	41	47.2	48.43
s38417	49	51.8	59.83	50	51.5	61.96	49	51.4	55.88	50	69.8	90.20	50	74.5	79.03
s38584	48	49.0	66.45	48	48.6	66.68	48	50.0	63.69	48	56.6	91.68	48	59.4	75.26
golem3	1333	1357.2	749.55	1336	1354.5	805.76	1339	1420.2	900.33	1485	1846.5	1518.37	1642	2159.7	1582.20
ibm01	181	214.2	74.66	180	193.0	78.58	181	241.7	74.28	181	252.5	93.26	181	194.7	69.47
ibm03	956	1017.3	216.21	952	1022.4	223.65	978	1153.4	209.34	962	1045.0	295.90	961	1051.3	283.78
ibm05	1715	1809.7	321.89	1723	1792.0	300.06	1738	1808.0	355.94	1747	1856.6	474.70	1784	1892.5	434.22
ibm07	851	948.1	626.45	876	996.6	569.90	853	948.1	603.37	896	980.2	634.90	852	914.0	629.22
ibm09	638	704.9	484.33	637	694.1	474.96	629	675.5	412.91	636	770.0	597.89	648	718.2	554.23
ibm11	960	1159.2	848.06	960	1051.5	741.36	965	1184.8	744.90	1007	1197.2	1168.02	982	1221.1	925.57
ibm13	869	930.2	939.98	861	897.9	1002.63	833	935.1	1073.89	836	1063.3	1304.32	834	987.4	1115.54
ibm15	2624	3058.1	2459.89	2625	2932.7	2059.90	2753	3488.4	1903.85	2676	3190.6	2732.28	2732	3241.8	2609.96
ibm17	2248	2371.5	2643.27	2220	2317.1	2536.81	2324	2507.7	2358.99	2254	2457.4	2848.69	2295	2487.6	2985.52

Table 2: The performance achieved by different vertex grouping schemes (*i.e.*, different values of *CType*). All the results correspond to bisections computed by hmetis with *Nruns = 20*, *UBfactor = 5*, *RType = 1*, *Vcycle = 1*, and *Reconst = 0*. All times are in seconds on a Pentium Pro @ 200 Mhz

	FM, RType=1			1wayFM, RType=2			EEFM, RType=3		
Circuit	Min	Avg	Time	Min	Avg	Time	Min	Avg	Time
avqsmall	127	154.2	69.97	127	148.1	71.55	127	143.8	55.51
avqlarge	127	149.5	88.99	127	150.1	82.69	127	147.9	61.67
industry2	163	212.3	62.04	165	219.4	64.13	162	214.8	50.41
industry3	258	274.6	97.14	257	277.2	94.30	241	271.2	76.88
s35932	43	43.4	47.53	43	43.5	51.00	43	43.5	38.19
s38417	49	51.4	62.14	49	51.1	64.50	49	52.2	44.54
s38584	48	49.2	65.39	48	48.6	70.95	47	48.1	51.84
golem3	1334	1352.1	704.96	1333	1350.0	683.02	1336	1359.8	519.59
ibm01	181	215.8	70.91	180	226.9	64.84	181	220.4	48.48
ibm03	955	1015.5	206.15	956	1010.8	173.65	956	1034.8	143.51
ibm05	1723	1804.2	337.72	1699	1765.8	276.10	1710	1791.9	195.41
ibm07	840	935.9	547.09	842	933.9	506.66	855	966.9	299.25
ibm09	637	729.6	488.04	629	699.8	477.79	629	691.7	289.51
ibm11	960	1122.9	778.80	960	1096.7	690.31	962	1103.0	435.30
ibm13	859	944.3	1080.43	851	963.4	755.76	832	1029.8	633.73
ibm15	2625	2975.0	2737.74	2625	3044.8	2258.74	2856	3082.4	1593.12
ibm17	2218	2406.9	3585.65	2239	2380.7	3239.31	2218	2383.4	2181.86

Table 3: The performance achieved by different refinement schemes (*i.e.*, different values of *RType*). All the results correspond to bisections computed by hmetis with *Nruns = 20*, *UBfactor = 5*, *CType = 1*, *Vcycle = 1*, and *Reconst = 0*. All times are in seconds on a Pentium Pro @ 200 Mhz

Table 4 shows the quality of the 4- and 8-way partitionings produced by the above two approaches. As we can see from this table, the second approach performs better in 16 cases, worse in 10 cases, and similarly for the remaining 8 cases.

	4-way		8-way	
Circuit	Nruns=20	4×Nruns=5	Nruns=20	4×Nruns=5
avqsmall	228	228	370	370
avqlarge	228	228	372	372
industry2	372	355	636	644
industry3	775	744	1546	1502
s35932	111	111	163	163
s38417	99	95	162	151
s38584	131	129	203	205
golem3	2217	2224	2872	2856
ibm01	496	501	758	742
ibm03	1686	1687	2392	2410
ibm05	3081	3062	4468	4449
ibm07	2234	2183	3280	3255
ibm09	1709	1708	2606	2638
ibm11	2331	2368	3503	3445
ibm13	1663	1740	2858	2727
ibm15	5167	5190	6833	6324
ibm17	5442	5385	8723	8870

Table 4: The performance achieved for a k-way partitionings using a single k-way partitioning with *Nruns = 20*, and four k-way partitionings with *Nruns = 5*.

5.2.2 Effect of the Reconst Parameter

Recall from Section 3.2, that the *Reconst* parameter controls how a hyperedge that is part of the cut is reconstructed in the sub-hypergraphs during recursive bisection. In particular, if *Reconst = 0*, then a hyperedge that is part of the cut is removed entirely from the sub-hypergraphs, and if *Reconst = 1*, then the hyperedge is reconstructed in each sub-hypergraph. This is done by creating two hyperedges (one for each partition), that span the vertices of the original hyperedge that are assigned to each partition.

The choice for the *Reconst* parameter can affect the quality and runtime of the k-way partitioning. In particular, if *Reconst = 0*, then the partitioning algorithm will run faster (as successive hypergraphs will have fewer hyperedges), and if *Reconst = 1*, then the partitioning algorithm can potentially do a better job in reducing the sum of external degrees (SOED) of the k-way partitioning.

This is illustrated in Table 5 that shows the effect of the *Reconst* parameter on the cut, SOED, and runtime, for a 4-way partitioning. From this table we can see that *Reconst = 0*, indeed results in a somewhat faster code, and that *Reconst = 1*, results in partitionings whose SOED is, in general, smaller. However, what is interesting with the results of Table 5, is that *Reconst = 0* results in partitionings that have smaller cut, compared to those obtained by setting *Reconst = 1*. So, if the objective is to obtain a k-way partitioning that has the smaller cut, one should use *Reconst = 0*. However, if minimizing the SOED is the primary focus, one may want to use *Reconst = 1*.

	No Reconstruction Reconst = 0			With Reconstruction Reconst = 1		
Circuit	Cut	SOED	Time	Cut	SOED	Time
avqsmall	228	568	111.92	246	567	118.28
avqlarge	253	605	126.82	257	569	137.67
industry2	381	841	107.47	429	884	110.56
industry3	791	1704	173.45	821	1647	179.89
s35932	111	232	72.05	111	226	72.07
s38417	100	224	96.78	109	228	99.70
s38584	130	294	106.35	138	291	111.90
golem3	2222	4613	1162.19	2239	4519	1226.58
ibm01	496	1003	124.53	498	998	128.04
ibm03	1691	3685	285.15	1717	3573	301.55
ibm05	3023	6701	459.12	3119	6611	532.71
ibm07	2212	4670	786.26	2253	4579	850.59
ibm09	1691	3485	790.61	1768	3579	774.53
ibm11	2339	4778	1155.37	2412	4862	1198.74
ibm13	1738	3770	1365.23	1755	3604	1398.37
ibm15	5103	10815	3339.88	5299	10844	3069.92
ibm17	5398	11041	4420.78	5421	10984	4854.22

Table 5: The performance achieved for a 4-way partitionings using different settings for the *Reconst* parameter. All the results correspond to 4-way partitioning computed by hmetis with *Nruns = 20*, *UBfactor = 5*, *CType = 1*, *RType = 1*, and *Vcycle = 1*. All times are in seconds on a Pentium Pro @ 200 Mhz

6 System Requirements and Contact Information

hMETIS has been written in C and it has been extensively tested on Sun, SGI, Linux, and IBM. Even though, hMETIS contains no known bugs, it does not mean that it is bug free. If you find any problems, please send email to ***metis@cs.umn.edu*** with a brief description of the problem. Also, any future updates to hMETIS will be made available on WWW at ***http://www.cs.umn.edu/˜metis***.

References

[1] Chalres Alpert. The ISPD98 circuit benchmark suite. In *Proc. Intl. Symposium of Physical Design*, 1998.

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

[3] 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*.

[4] 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.

[5] Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. Clustering based on association rule hypergraphs. In *Proc. of Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

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

[7] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. Technical Report TR 98-036, Department of Computer Science, University of Minnesota, 1998.

[8] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. Also available on WWW at URL http://www.cs.umn.edu/˜karypis.

[9] 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.

[10] 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.

[11] 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.

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

[13] S. Shekhar and D. R. Liu. Partitioning similarity graphs: A framework for declustering problmes. Technical Report TR 94–18, University of Minnesota, Department of Computer Science, Minneapolis, MN, 1994. Accepted in Information Systems Journal.