Circuit Theory Laboratory Report Series CT-49

Using METIS and hMETIS Algorithms in Circuit Partitioning

Pekka Miettinen, Mikko Honkala, and Janne Roos

Helsinki University of Technology
Department of Electrical and Communications Engineering
Circuit Theory Laboratory Espoo 2006

Distribution:
Helsinki University of Technology
Circuit Theory Laboratory
P.O. Box 3000
FI-02015 TKK
Tel. +358-9-451 2295
Fax. +358-9-451 4818
E-mail: pekka@ct.hut.fi

ISBN 951-22-8362-X
ISSN 1455-9757

Picaset Oy
Helsinki 2006

P. Miettinen, M. Honkala, and J. Roos,
Using METIS and hMETIS Algorithms in Circuit Partitioning,

Circuit Theory Laboratory Report Series, No. CT-49, Espoo 2006, 17 pp.,
ISBN 951-22-8362-X,
ISSN 1455-9757.

Abstract

In order to facilitate the analysis of large circuits, it has been found useful to partition a circuit into smaller subcircuits. METIS and hMETIS are algorithms used primarily to partition graphs and hypergraphs, respectively. In this document, several methods of formulating graphs from circuit netlists for METIS and hMETIS are considered and tested with circuit examples. The simulation results show that although METIS is considerably faster, hMETIS is more flexible and often manages to find better solutions, especially regarding contiguous partitioning.

Keywords: METIS, hMETIS, Circuit Partitioning, Graph

Contents

List of Figures

List of Tables

1 Introduction

Simulating a large circuit creates various computational problems, the foremost being the processing time. Thus, it is often useful to partition the circuit into smaller subcircuits and process the subcircuits separately, for example, in parallel processing. Another application that benefits from circuit partitioning is Model-Order Reduction (MOR), and that of linear MOR of RLC circuits particularly, which is the orientation in this study.

A typical example of linear MOR is the moment-matching technique [1]–[3], which approximates the higher order networks with waveforms generated with lower order moments. However, since the reduced circuits are generally described by full admittance matrices [4]–[6] (Y-matrices), if the number of external ports is large, these methods are ineffective due to the increasing size of the admittance matrix.

A method of overcoming the large Y-matrix, when dealing with a large number of external ports, is to partition the circuit into smaller subcircuits. Thus, instead of modeling the whole circuit with one large high-order Y-matrix, the circuit is divided into smaller sections, which can each be represented with lower order equivalent subcircuits. This will greatly reduce the simulation time and generally increase accuracy.

This paper discusses the methods of partitioning a given RLC circuit into smaller subcircuits using METIS and hMETIS algorithms. METIS and hMETIS are algorithms designed mainly for graph and hypergraph partitioning and thus the first problem is to map a circuit netlist as a graph. Later on, the two algorithms are tested with four benchmark circuits and the test results are discussed.

2 METIS and hMETIS algorithms

METIS is an algorithm package for partitioning large irregular graphs, partitioning large meshes, and computing fill-reducing orderings of sparse matrices. METIS provides two stand-alone programs, `pmetis` and `kmetis`, to partition graphs into partitions of equal size. The METIS algorithms are based on multilevel graph partitioning: `pmetis` is based on multilevel recursive bisectioning described in [7] and `kmetis` is based on multilevel k-way partitioning described in [8]. Multilevel partitioning algorithms first reduce the size of the graph by coarsening the graph's details. This takes form as collapsing adjacent vertices and edges (see Section 3). The smaller graph is then partitioned and refined into the original graph.

As the partitioning algorithms operate with the reduced-size graph, they are extremely fast compared to traditional partitioning algorithms that compute a partition directly on the original graph. Extensive testing has also shown that the partitions provided by METIS are consistently better than those produced by spectral partitioning algorithms [9].

The hMETIS is an extension of METIS, which uses hypergraphs instead of graphs [10]. A hypergraph is a generalization of a graph that allows an edge — i.e., a

hyperedge — to connect more than two vertices. A stand-alone program `shmetis` was used to partition the hypergraphs.

METIS and hMETIS are designed to operate on large graphs with generally more than 1000 vertices. Experiments with applying the algorithms to much smaller graphs (20–40 vertices) have shown poor quality partitionings.

The METIS algorithm package is available as a C-based source code. hMETIS is available as a binary library file. In this work, both have been verified to function identically to the stand-alone programs with standard C function calls. A METIS algorithm package intended for parallel computing, ParMETIS, is also available [11].

3 Graphs

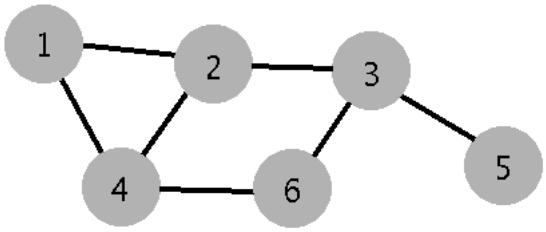

Figure 1: An example graph. Vertices are numbered 1–6.

A graph (see Fig. 1) is a set of objects called vertices and links between the objects called edges that connect two vertices. Information can be stored in graphs by assigning values to the vertices and/or edges. A classic definition can be found in [12]: "A graph G consists of a finite nonempty set $V = V(G)$ of p points together with a prescribed set X of q unordered pairs of distinct points of V. Each pair $x = u, v$ of points in X is a line of G, and x is said to join u and v. We write $x = uv$ and say that u and v are adjacent points (sometimes denoted u adj v); point u and line x are incident with each other, as are v and x. If two distinct lines x and y are incident with a common point, then they are adjacent lines. A graph with p points and q lines is called a (p, q) graph. The $(1, 0)$ graph is trivial."

A graph can be either undirected or directed. If a line between vertices A and B is considered the same as line between B and A, the graph is undirected. In a directed graph, each line between the vertices, i.e. edges, has a direction and the edges are called directed edges. All edges in METIS and hMETIS are undirected.

Generally, an edge connects exactly two vertices, but in a hypergraph (see Fig. 2), hyperedges can connect multiple edges. This is particularly useful for describing typical circuit netlists, for example.

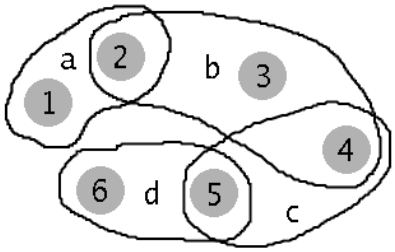

Figure 2: An example hypergraph. Vertices are numbered 1–6 and hyperedges connecting the vertices are labeled a–d.

4 Mapping circuit netlists in graphs

Three different approaches to mapping circuits into graphs were studied: 1) a case with circuit components (R,C,L) as edges, 2) components as vertices, and 3) hyperedges as circuit nodes with components as vertices.

Ground branches — a circuit branch with the other node as the ground node — of circuit netlists are left out from the mapping. This is done for several reasons. First, in graph theoretic network formulations, the ground branches are not considered in the graph incidence matrices. Second, the ground branches seem to confuse METIS's non-weighted graph partitionings, especially when using the first mapping method, where components are mapped as edges. Finally, if weights are used, the circuit ground node and the ground branches are virtually always grouped together due to the number of components connecting to the node, which is seldom desirable.

It should be noted here that the graph partitioning is purely symbolic; although weights can be applied to both vertices and edges, there is no simple method of determining the importance of a single component and therefore all components are treated as equally important. Likewise, no difference between the types of components is noted: resistors, capasitors, and inductances are treated as identical elements.

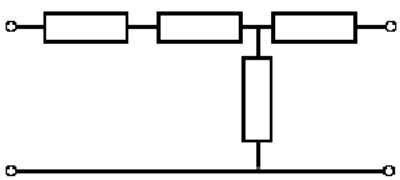

Figure 3: An example circuit.

Figure 3 presents an example circuit that is used to demonstrate the different mapping methods in the next sections. The component types are arbitrary. In the following examples, vertices are shown as component boxes where they are analoguous — although it is important to note that once the circuits are mapped as graphs, a reverse mapping must be performed to obtain a subcircuit based on the

graph partitioning. For example if the method described in Section 4.2 is used, the location of particular circuit nodes may be no longer unambiguous. In Fig. 5 the circuit node between analoguous vertices 2, 3, and 4 is replaced by 3 edges. In this case the mapping is analoguous to star→delta transformation.

In a traditional graph, edges can link only two vertices together. In a circuit netlist, the appropriate edges would be the circuit nodes, since the primary information of a circuit is stored in the components. In all but the most simple circuit netlists, circuit nodes often connect more than two components. This generates a problem for graph mapping.

4.1 Components as edges

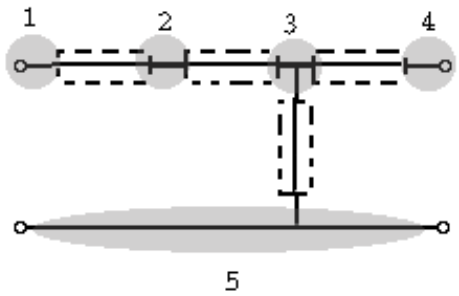

Figure 4: Using components as edges in a graph. Vertices are shown in grey circles numbered 1–5.

In the approach used here, circuit components were mapped as graph edges, and circuit nodes as graph vertices, since a vertex can connect multiple edges. No hypergraphs were used, so METIS algorithms were used to partition these graphs. The upside of this approach is that it should be easy to implement with SPICE-like netlists, since only one edge is needed to map a circuit component. In Table 1, the SPICE netlist of Fig. 4 and the corresponding METIS graph format is shown. Each line i represents the i:th vertex. For each vertex, the adjacent vertices are listed.

Table 1: SPICE netlist compared to METIS graph format.

SPICE netlist format	METIS graph format
Ra 1 2 10k	1 : 2
Rb 2 3 10k	2 : 1 3
Rc 3 4 10k	3 : 2 4 5
Rd 3 5 10k	4 : 3
	5 : 3

The problem, however, is that METIS partitions the graphs divided into groups of vertices. Sections are made across the edges and thus information (i.e., the

location of the component) contained in the divided edge is temporarily lost — the edge can be later retrieved by comparing the graph with the original netlist. Although this is clearly unfortunate, the importance of one lost information of a component placing per edgecut is probably in most cases not critical, especially when large graphs with several thousands of components are partitioned. Furthermore, the METIS algorithm strives, first, to divide the graph into parts of equal size, and thus the placing of a component is not always the optimum regarding the grouping. For example, this will sometimes lead to non-contiguos partitioning (See Fig. 10).

In this mapping method, no weights were added to the edges or the vertices.

4.2 Components as vertices

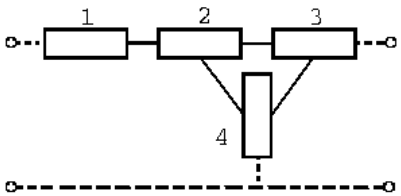

Figure 5: Using components as vertices numbered 1–4. Edges are drawn with continuous lines

An edge can connect only two vertices, whereas in typical circuit netlists often more than two components are connected by nodes. This is solved in the approach here by drawing edges between all connected components, thus a circuit node is often represented by multiple edges — namely $N = \sum_{i=1}^{n-1} i$, where n is the number of vertices connected, and N is the total number of edges generated.

Again, METIS is used, since no hyperedges are needed. As is easily seen, the number of edges used to map the circuit into a graph is much higher than in the previous method, and depending on the number of components connected by single nodes, the number of edges can increase rapidly.

No weights were applied to the edges, but since a circuit node with multiple connected components creates a proportional number of edges as a graph, the method has in fact a built-in weight factoring.

4.3 Hyperedges as circuit nodes

Hyperedges can connect multiple vertices and represent a typical circuit node the best. hMETIS is used to partition the hypergraph thus formed. Components are mapped as vertices between the hyperedges.

Weights can be added to the hyperedges; a simple method is to give a hyperedge a weight equal to the number of connected vertices minus one. Thus a hyperedge connecting two vertices has a weight of 1, hyperedge connecting three vertices has

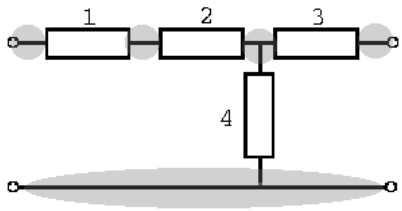

Figure 6: Hyperedges in graphs. Vertices are numbered 1–4, hyperedges are marked with grey circles. Note that a single hyperedge can connect multiple vertices.

a weight of 2, and so forth. hMETIS and METIS allow positive, non-zero weights only.

5 Benchmark circuits

Four different circuit layouts were created for testing graph partitioning, and three of those are presented in the following. The fourth circuit layout was a simplified version of the ladder circuit in Fig. 7 with only 20 nodes. As mentioned earlier, no ground branches are visible in the netlists and they are considered to be grouped with the nearest component in the partitioning.

5.1 Ladder circuit

The ladder circuit is presented in Fig. 7. The circuit contains 202 circuit nodes and 209 components and represents two coupled transmission lines.

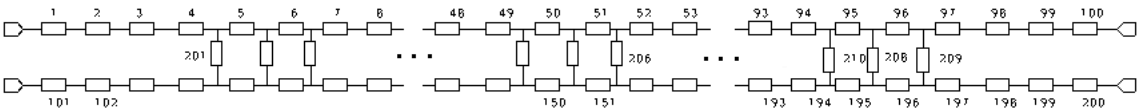

Figure 7: Ladder circuit.

5.2 Tree circuit

The tree circuit is presented in Fig. 8. The layout mimics a large interconnect circuit with seven transmission lines. 140 components and 141 circuit nodes were used.

5.3 Multiladder circuit

A circuit containing 212 components and 197 circuit nodes is shown in Fig. 9. The system represents multiple transmission lines with crosstalk disturbances.

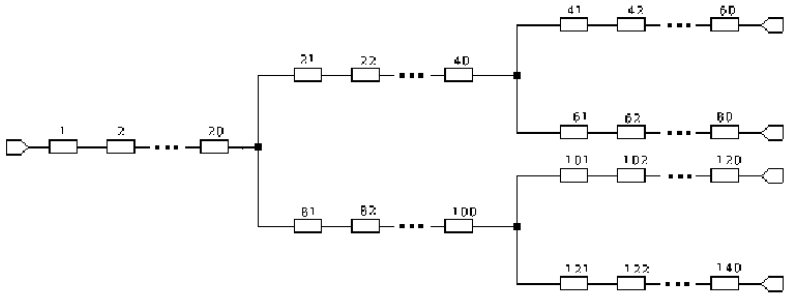

Figure 8: Tree circuit

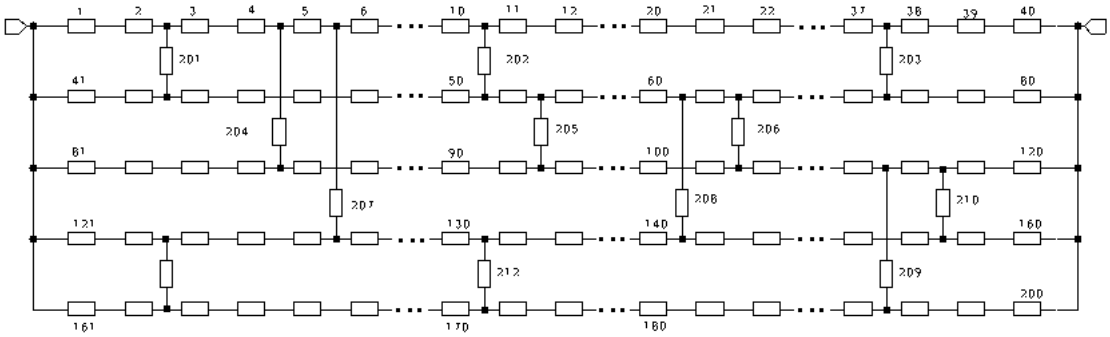

Figure 9: Multiladder circuit.

6 Partitioning results

The three graph-mapping methods described in Section 4 were tested with the four benchmark circuits from Section 5. METIS/hMETIS algorithms are non-determinant, meaning each partitioning is based on random-number algorithms. Thus, successive partitioning runs may produce different results, especially in the case of non-symmetric circuits. To monitor the possible different results generated due to this randomness, partitioning tests were processed several times with the same initial parameters.

The three larger circuits described in Section 5 were all divided into 5 and 10 partitions and, in addition, other divisions were tested on the more symmetrical ladder and tree circuits.

METIS algorithms attempt to partition the circuit into equal partitions. If the circuit is not symmetrical and the number of vertices is not divisible by the number of partitions (as is usually the case), the algorithm sometimes finds only non-contiguous partitioning results (Fig. 10). Although not all non-contiguous partitions are undesirable (such as partitioning adjacent parts of coupled transmission lines together), generally they will increase the number of resulting ports and generate disconnected components and/or component clusters that can not be reduced by MOR algorithms. Thus an attempt was made to minimize non-contiguous par-

titioning in the course of the testing. This meant increasing the balancing factor for hMETIS — for METIS nothing could be done if only non-contiguous partitions were obtained.

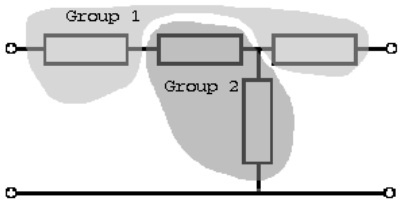

Figure 10: Example of a non-contiguous partitioning: vertex group 1.

The partitionings of a graph are given in a result file with one line for each vertice in the given graph. On each line is a number that tells the partition the vertice belongs. Thus the partitioning does not give ready subgraphs or subcircuits, and further processing is needed to identify, e.g., which edges are cut, and no longer exist. Of course, if other than the hyperedge method for graph mapping is used, an algorithm for reverse mapping, i.e. a graph into a circuit layout, is needed.

6.1 METIS and components as edges

Results for graph partitioning with the METIS algorithms are presented in Table 2. The ladder, tree and multiladder circuits are shown in detail in Section 5. For each circuit, several partitions are listed and for each generated subcircuit the numbers of external and internal nodes are shown: n_e and n_i, respectively. Also the ratio $n_\mathrm{i}/n_\mathrm{e}$ is listed. If the partitioning resulted repeatedly in a non-contiguous groupings of vertices — i.e., it was evident that METIS could not produce a contiguous partitioning — this is stated in the Contiguous-column. The particular non-contiguous group in the partition is marked in lower-case. In cases where both contiguous and non-contiguous partitionings were obtained, it was noticeable that contiguous solutions had fewer external nodes than non-contiguous ones. Finally, the time to process the program is listed.

METIS partitions the graphs described in previous sections generally well. For graph partitions with less than eight partitions, `pmetis` program was used and for more partitions `kmetis` program was used due to its better performance with greater number of partitions [9].

For symmetrical graphs with an optimum number of partitions such as the ladder circuit and tree circuit, METIS finds excellent partitions. Such is the case when the tree circuit in Fig. 8 is divided into seven or fourteen parts. In these cases, the algorithms found contiguous groups of vertices that were of nearly identical size. For example, the tree graph was partitioned such that each of the seven partitions took one transmission line of the circuit, which is intuitively clearly the best approach.

Table 2: METIS partitioning results, components as edges.

Circuit	Partitions	n_e	n_i	$n_\mathrm{i}/n_\mathrm{e}$	Contiguous	CPU/s
Ladder	1 :	4	198	49.5	–	–
	5 :	16	192	12.0	YES	0.001
	no. 1	4	38	9.8		
	no. 2	2	40	19.5		
	no. 3	4	37	9.0		
	no. 4	2	40	19.5		
	no. 5	4	37	9.8		
	10 :	28	186	6.6	NO	0.003
	no. 1	4	17	4.3		
	no. 2	4	17	4.3	no	
	no. 3	2	21	10.5		
	no. 4	2	21	10.5		
	no. 5	4	16	4.0		
	no. 6	2	20	10.0		
	no. 7	2	19	9.5		
	no. 8	4	16	4.0		
	no. 9	2	19	9.5		
	no. 10	2	19	9.5		
Tree	1 :	5	136	27.2	–	–
	5 :	17	130	7.6	NO	0.002
	no. 1	3	26	8.7		
	no. 2	4	26	6.5	no	
	no. 3	3	26	8.7		
	no. 4	3	26	8.7		
	no. 5	4	26	6.5		
	7 :	14	137	9.9	YES	0.001
	no. 1	2	20	10.0		
	no. 2	2	20	10.0		
	no. 3	2	20	10.0		
	no. 4	2	19	9.5		
	no. 5	2	19	9.5		
	no. 6	2	20	10.0		
	no. 7	2	19	9.5		
	10 :	24	127	5.3	YES	0.003
	no. 1	2	13	6.5		
	no. 2	3	13	4.3		
	no. 3	2	13	6.5		
	no. 4	3	13	4.3		
	no. 5	2	12	6.0		
	no. 6	2	12	6.0		

Table 2: (continued)

Circuit	Partitions	n_e	n_i	n_i/n_e	Contiguous	CPU/s
	no. 7	2	13	6.5		
	no. 8	3	13	6.5		
	no. 9	2	12	6.0		
	no. 10	2	12	6.0		
Multi	1 :	2	195	97.5	–	–
	5 :	26	184	7.1	YES	0.002
	no. 1	5	37	7.4		
	no. 2	6	35	5.8		
	no. 3	4	40	10.0		
	no. 4	6	35	5.8		
	no. 5	5	37	7.4		
	10 :	41	177	4.3	YES	0.003
	no. 1	3	20	6.7		
	no. 2	5	13	2.6		
	no. 3	2	20	10.0		
	no. 4	4	20	5.0		
	no. 5	5	15	3.0		
	no. 6	4	18	4.5		
	no. 7	5	11	2.2		
	no. 8	4	20	5.0		
	no. 9	7	11	1.6		
	no. 10	2	25	12.5		

Problems arise easily, however, when the number of desired partitions is different from the optimum. For example, partitioning the tree graph into five partitions resulted in non-contiguous partitioning for two groups of vertices. This happens evidently because the algorithm attempts to equalize the group sizes first and foremost. The partitions can be of different sizes, if the relative sizes are given in advance as parameters. In this application, giving the partition sizes in advance is rarely an option, though.

Since METIS produces partitions of near-equal size, the number of internal nodes in each partition is roughly inversely proportional to the number of external nodes and, thus, also proportional to the n_i/n_e ratio as well.

The multiladder circuit in Fig. 9 was examined as an example of a non-symmetrical graph. Partitioning the graph into five partitions resulted in non-contiguous groups using the `pmetis` program. However, `kmetis` that uses k-way partitioning, managed for some reason to find contiguous partitioning solutions for five and ten partitions most of the time, although it is not recommended to be used for partitions with less than eight partitions [9].

METIS algorithms are extremely fast. A processing time for the pmetis and kmetis programs of 0.001–0.003 seconds could be measured, but the actual algorithm processed the graph partitioning in less than 0.001 seconds, which was too short for the measuring precision.

6.2 METIS and components as vertices

The METIS algorithms were implemented with graph-mapping methods described in Sections 4.1 and 4.2. As noted earlier, using components as edges has the drawback that one component placing information per edgecut is lost. Using components as vertices overcomes this problem, but produces slightly more external nodes for the subcircuits as shown in Table 3.

Table 3: Results of using components as vertices.

Circuit	Partitions	n_e	n_i	$n_\mathrm{i}/n_\mathrm{e}$	Contiguous	CPU/s
Multi	5	28	183	6.5	YES	0.002
	no. 1	8	23	2.9		
	no. 2	5	41	8.2		
	no. 3	4	45	11.3		
	no. 4	5	39	7.8		
	no. 5	6	35	5.8		
	10 :	45	175	3.9	NO	0.001
	no. 1	6	19	3.2		
	no. 2	4	16	4.0		
	no. 3	2	23	11.5		
	no. 4	5	15	3.0		
	no. 5	7	10	1.4		
	no. 6	4	18	4.5		
	no. 7	6	16	2.7		
	no. 8	4	22	5.5		
	no. 9	3	22	7.3		
	no. 10	4	14	3.5	no	

A more important benefit of using components as vertices was found, however, in the built-in weighting of the circuit nodes. If the circuit nodes are not weighted in relation with the number of connected components, unfortunate partitionings may occur regarding future use of the partitioning, which in this paper is considered to be MOR. The difference of unweighted and weighted partitioning is demonstrated in Figs. 11 and 12, respectively. In Fig. 11, components 2 and 4 cannot be reduced, since they are between nodes of another group, although the partition is contiguous.

In Fig. 12, this is not a problem. Note that the partition groups are of different sizes in the example and that the partitioning is shown on the circuit netlist, not on a graph.

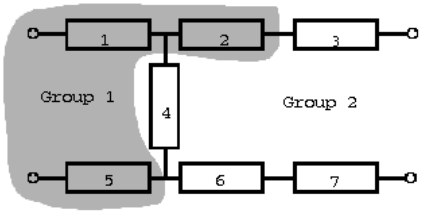

Figure 11: Unweighted partitioning.

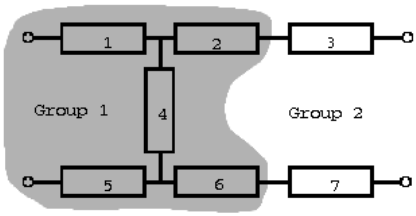

Figure 12: Weighted partitioning.

Of course, weights can also be used with the components-as-edges method. In that case, results similar to the components-as-vertices method should be expected.

Using the components-as-vertices method creates a graph with notably more edges than the components-as-edges method, $N = \sum_{i=1}^{n-1} i$ compared to n. How much this affects the processing time with very large circuits is unknown, because even with the increased number of edges, the processing times for the graphs studied were too short for the measuring precision (0.001 seconds).

6.3 hMETIS and hyperedges

Table 4 shows the results of graph partitioning for hMETIS algorithms. Only results with contiguous partitionings are listed; the partitionings with non-contiguous groups had 2–14 more external nodes with roughly the same number of internal nodes. By comparing the results with the METIS results in Table 2 it can be seen that hMETIS generally produces partitions with fewer external nodes, which is a desirable quality in a subcircuit for MOR. Unlike METIS, hMETIS partitions vary in size according to the `UBfactor` and, therefore, the $n_\mathrm{i}/n_\mathrm{e}$ ratio is not proportional to the number of internal or external nodes.

Implementing hMETIS and hyperedges in graphs is clearly the most appropriate method of graph mapping, since circuit nodes can be mapped as single hyperedges and components as analoguous vertices. The stand-alone program `shmetis` was used to partition the graphs.

Table 4: hMETIS partitioning results.

Circuit	Partitions	n_e	n_i	n_i/n_e	`UBfactor`	CPU/s
Ladder	5	16	192	12.0	5	0.120
	no. 1	4	48	12.0		
	no. 2	2	40	20.0		
	no. 3	2	38	19.0		
	no. 4	4	31	7.8		
	no. 5	4	35	8.8		
	10	28	186	6.6	20	0.339
	no. 1	2	19	9.5		
	no. 2	4	20	5.0		
	no. 3	2	21	10.5		
	no. 4	2	18	9.0		
	no. 5	4	16	4.0		
	no. 6	2	19	9.5		
	no. 7	4	14	3.5		
	no. 8	2	21	10.5		
	no. 9	2	13	6.5		
	no. 10	4	24	12.0		
Tree	5	12	132	11.0	20	0.187
	no. 1	3	37	12.3		
	no. 2	3	38	12.7		
	no. 3	2	19	9.5		
	no. 4	2	19	9.5		
	no. 5	2	19	9.5		
	7	14	133	9.5	5	0.247
	no. 1	2	19	9.5		
	no. 2	2	19	9.5		
	no. 3	2	19	9.5		
	no. 4	2	19	9.5		
	no. 5	2	19	9.5		
	no. 6	2	19	9.5		
	no. 7	2	19	9.5		
	10	20	130	6.5	20	0.333
	no. 1	2	9	4.5		
	no. 2	2	9	4.5		
	no. 3	2	19	9.5		
	no. 4	2	19	9.5		
	no. 5	2	19	9.5		
	no. 6	2	9	4.5		
	no. 7	2	9	4.5		

Table 4: (continued)

Circuit	Partitions	n_e	n_i	n_i/n_e	UBfactor	CPU/s
	no. 8	2	9	4.5		
	no. 9	2	9	4.5		
	no. 10	2	19	9.5		
Multi	5	21	189	9.0	20	0.241
	no. 1	2	27	13.5		
	no. 2	3	19	6.3		
	no. 3	3	23	7.7		
	no. 4	6	57	9.5		
	no. 5	6	63	10.5		
	10	33	178	5.4	30	0.362
	no. 1	4	19	4.8		
	no. 2	4	22	5.5		
	no. 3	4	17	4.3		
	no. 4	4	18	4.5		
	no. 5	3	19	6.3		
	no. 6	4	17	4.3		
	no. 7	2	19	9.5		
	no. 8	4	12	3.0		
	no. 9	4	19	4.8		
	no. 10	3	16	5.3		
Multi, with weights on the hyperedges	5	18	187	3.6	20	0.283
	no. 1	2	27	13.5		
	no. 2	2	26	13.0		
	no. 3	5	45	9.0		
	no. 4	5	46	9.2		
	no. 5	6	43	7.2		
	10	37	179	4.8	20	0.376
	no. 1	2	15	7.5		
	no. 2	2	14	7.0		
	no. 3	2	23	11.5		
	no. 4	5	17	3.4		
	no. 5	4	20	5.0		
	no. 6	6	16	2.7		
	no. 7	3	19	6.3		
	no. 8	4	22	5.5		
	no. 9	4	16	4.0		
	no. 10	5	17	3.4		

hMETIS is notably slower to run than METIS. Processing times for tested graphs were between 0.1 and 0.4 seconds. Although absolutely speaking this is still quite fast, the relative difference between hMETIS and METIS is considerable.

The greatest difference in the usage of hMETIS and METIS is that hMETIS does not aim to find partitions of exactly equal size; indeed, with hMETIS, partitions of exactly equal size cannot be found except by accident. Instead, a relative imbalance factor `UBfactor` is given as an initial parameter, which tells the algorithm how much the partitions can differ in size. This is a very serviceable feature if the graph is such that contiguous partitions of equal size are hard to find — if the algorithm does not find an acceptable solution at first, the imbalance factor can be increased and eventually the algorithm finds a satisfactory partitioning. It should be noted that the algorithm arguably always strives to find contiguous partitionings and non-contiguous partitionings are found only in the case of difficulties with the graph layout and/or desired number of partitions.

hMETIS performed well with the graph-partition testings. In the case of symmetrical graphs and a convenient number of partitions with the tree and ladder graph (in Figs. 8 and 7), the solutions were almost identical to those found with the METIS algorithms — as should be expected. However, when non-contiguous partitionings were encountered with not as fitting initial parameters, contiguous solutions could be found by permitting increased imbalance between the partitions (the `UBfactor` was increased from 5 to 20 in most cases).

hMETIS was able to find contiguous solutions with higher imbalance-factor values also with the multiladder circuit shown in Fig. 9. The algorithm operates on some level based on random numbers, so different results could be obtained with consecutive runs of the program. This can be both an advantage — a fortunate partitioning can be found by chance in the hardest of graphs — or a drawback, in which case a known seed number can be given to the algorithm to ensure uniform results.

Weights can be added to the hyperedges to avoid the problem of unreduceable components described in Section 6.2. This does not seem to hinder the partitioning noticeably otherwise.

If some information about the graph is known beforehand, a special `fixfile` can also be given to hMETIS as a parameter that defines the partitioning for some or all of the vertices.

7 Discussion

METIS and hMETIS work in many ways in a similar fashion. METIS is considerably faster, but hMETIS finds contiguous solutions when METIS fails to do so. This comes, however, at the price of increasing difference in the size of the resulting partitions.

It is not obvious if the contiguity of the partitions is important and how important, if so. In the best-case scenario, a non-contiguous partition may be even better than a complex contiguous one, but implementing an algorithm to identify such cases is anything but trivial and more importantly, may take more time than the difference between METIS and hMETIS.

Furthermore, the imbalance between the contiguous partitions that hMETIS produces with high values of `UBfactor` may affect the quality of following MOR. If severe imbalance is needed to reach contiguous groupings of vertices, one partition may be too small to benefit fully from MOR, while another may be too large to be represented with a low order model.

If, however, contiguous partitioning is considered the optimum — as is done in this paper — hMETIS is probably a better choice for circuit partitioning. It is clearly slower than METIS, but seems to find solutions with better external–internal port ratio and when the hMETIS processing time is considered a part of the whole MOR process, this should be neglible or at least minor compared to the whole running time.

If an optimal choice regarding the CPU time is required, the best approach might be to first apply the METIS algorithms to the graph, then test the partitionings for contiguity with a separate algorithm, and, if non-contiguous results are obtained, use hMETIS on the graph with increasing values of `UBfactor` instead.

Acknowledgments

This work was funded by NEC Europe Ltd., C&C Research Laboratories, via the NETlist-based Model-Order Reduction (NETMOR) project.

References

[1] L. T. Pillage and R. A. Rohrer, "Asymptotic Waveform Evaluation for Timing Analysis," *IEEE Trans. on CAD,* vol. CAD-9, pp. 352–366, April 1990.

[2] S. Lin, and E. S. Kuh, "Transient Simulation of Lossy Interconnects Based on the Recursive Convolution Formulation," *IEEE Trans. on Circuits and Systems I: Fundamental theory and Applications,* vol. 39, pp. 879–892, Nov. 1992.

[3] H. Liao, W. Dai, R. Wang, and F. Y. Chang, "S-Parameter Based Macro Model of Distributed-Lumped Networks Using Exponentially Decayed Polynomial Function," *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 726–731, June 1993.

[4] H. Liao and W. W.-M. Dai, "Partitioning and reduction of RC interconnect networks based on scattering parameter macromodels," in *Digest of Technical Papers of IEEE/ACM International Conference on Computer Aided Design*, pp. 704–709, 1995.

[5] J. V. Raghavan, J. E. Bracken, and R. A. Rohrer, "AWESpice: A general Tool for the Accurate and Efficient Simulation of Interconnect Problems," *Proceedings of the 29th ACM/IEEE Design Automation Conference,* pp. 87–92, June 1992.

[6] S. Y. Kim, N. Gopal, and L. T. Pillage, "Time-Domain Macromodels for VLSI Interconnect Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 13, pp. 1257–1270, Oct. 1994.

[7] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing,* 1998.

[8] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," *Journal of Parallel and Distributed Computing,* vol. 48, no. 1, pp. 96–129, 1998.

[9] G. Karypis and V. Kumar, "METIS, A software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0," `http://glaros.dtc.umn.edu/gkhome/metis/metis/download`

[10] G. Karypis and V. Kumar, "hMETIS, A Hypergraph Partitioning Package Version 1.5.3," `http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download`

[11] G. Karypis and V. Kumar, "ParMETIS — Parallel Graph Partitioning and Fill-reducing Matrix Ordering," `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`

[12] H. Frank, "Graph Theory", Addison–Wesley, Reading, MA, 1969.