Recursive Bi-Partitioning of Netlists for Large Number of Partitions

Rolf Drechsler
Wolfgang Günther
Thomas Eschbach
Lothar Linhard
Gerhard Angst

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 164, November 2001

Recursive Bi-Partitioning of Netlists for Large Number of Partitions

R. Drechsler[1] W. Günther[2] T. Eschbach[3] L. Linhard[4] G. Angst[4]

[1] University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

[2] Siemens AG
Corporate Technology
81730 Munich, Germany
wolfgang.guenther@mchp.siemens.de

[3] Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
eschbach@informatik.uni-freiburg.de

[4] Concept Engineering GmbH
Bötzinger Str. 29
79111 Freiburg im Breisgau, Germany
{gerhard,lothar}@concept.de

Abstract

In many application in VLSI CAD, a given netlist has to be partitioned into smaller sub-designs which can be handled much better. In this paper we present a new recursive bi-partitioning algorithm that is especially applicable, if a large number of final partitions, e.g. more than 1000, has to be computed. The algorithm consists of two steps. Based on recursive splits the problem is divided in several sub-problems, but with increasing recursion depth more run time is invested. By this an initial solution is determined very fast. The core of the method is a second step, where a very powerful greedy algorithm is applied to refine the partitions. Experimental results are given that compare the new approach to the state-of-the-art tools. The experiments show that the new approach outperforms the standard techniques with respect to run time and quality. Furthermore, the memory usage is very low and is reduced in comparison to other methods by more than a factor of four. The proposed method has been integrated in a commercially available tool.

1 Introduction

Netlist partitioning is very important in many applications in VLSI CAD. Since modern designs may contain several million gates, it is often necessary to partition the netlist into small parts which can be handled efficiently, for instance to optimize the netlist or to do the routing. The problem of partitioning also occurs is FPGA design, where designs have to be split into parts that fit into one FPGA, while the number of connections between these parts should be minimal.

Another application is visualization of netlists. The problem of drawing a netlist usually is divided into several subproblems [16, 8], like partitioning [9, 12], crossing minimization [10, 17, 18], and level assignment [5]. In most cases only a small fraction of the complete design is shown on the screen. E.g. of a multi-million gate ASIC only up to 100 gates are visible at a time.

In these cases much more than 1000 partitions of the netlist are necessary. At first glance, this number seems to be too extreme, however, if an error in a design has to be found which is either on the top level or at an unknown location of the design, it is necessary for the drawing algorithms to partition a multi-million gate netlist within a reasonable amount of time. Both memory usage and run time are important here, since the user neither wants to wait for hours before he can trace some signals in the design, nor does he want to buy a lot of main memory for this application. Last but not least, the quality of the partitioning is also important to obtain an easy-to-read drawing.

Table 1: Memory for k-way partitioning

k	k-way	rec. bipart.
2	16 MB	13 MB
50	44 MB	13 MB
100	106 MB	13 MB
200	174 MB	13 MB
300	222 MB	14 MB

1.1 Previous Work

Most partitioning algorithms use an iterative improvement method, which was already introduced in 1970 [14]. In the meantime it has been improved in several ways [7, 4, 6]. All these approaches split the netlist into two partitions and therefore they are called bi-partitioning.

To obtain more than two partitions, there are two major approaches:

1. recursive bi-partitioning. Partitions are split recursively until the desired number of partitions is obtained.
2. simultaneous computation of all partitions (also called k-way partitioning).

The latter approach was proposed in [19, 20]. It can be seen as a generalization of bi-partitioning to multiple partitions. In a recent study several move-based multi-way approaches have been compared [21]. All these approaches have in common that for k-way partitioning the amount of memory needed grows quadratic with k. Therefore, these techniques cannot be applied, if k becomes large. Instead of giving a detailed analysis, we give an impression of the memory explosion reporting some numbers derived by the k-way partitioning approach of METIS [12, 13], a state-of-the-art partitioning algorithm. In Table 1 the memory needed for k-way partitioning and for recursive bi-partitioning using benchmark *ibm06* from ISPD98 [1] is given for growing k. As can easily be seen, if k gets larger these approaches cannot be used any longer. But this choice of k is not unrealistic when netlists of one million gates have to be partitioned in parts of 100 gates each. This is the reason why we restrict ourselves to the approach based on recursive bi-partitioning.

The most successful approaches have been obtained so far using multi-level techniques [11, 2, 13] and these are also applicable for larger k. However, we only consider these techniques for comparison in the following, using the implementation of hMETIS.

For an excellent overview on different graph partitioning algorithms and their applications see [3].

1.2 New Algorithm

In this paper we present a new algorithm that is dedicated to large number of partitions. The algorithm consists of two main steps. Firstly, an initial solution is computed based on an improved move-based method. Instead of spending the same "effort" by using the same number of runs at each recursive level, our algorithm starts with a very simple solution and with increasing recursion depth, more and more starting solutions are considered. This has the effect that less effort is spent on the complete netlist, but more and more computation time is invested on smaller graphs. This results in a very fast algorithm. At each level a KLFM-like [14, 7, 15] approach is applied. In a second step a very powerful greedy algorithm is started that moves nodes across partitions, but in contrast to the "classical" k-way approaches only local gain values are computed and by this the memory consumption remains linear in the number of partitions. Another benefit of this technique is that it can easily be integrated in existing recursive bi-partitioning algorithms based on KLFM.

We give comparisons on benchmarks versus the state-of-the-art techniques that are based on multi-level structures [11, 2, 13]. It is shown that the new approach outperforms these algorithms for large numbers of partitions with respect to run time **and** quality.

The proposed method has been integrated in a commercially available tool.

The paper is structured as follows: In Section 2 the detailed problem formulation is given and a small example is presented. The new algorithm is described in Section 3. In Section 4 experiments are given and finally the main results are summarized.

2 Problem Formulation

Before we discuss the new algorithm, we formally describe the problem.

Min-Cut k-Way Partitioning: Let G be a graph that has to be partitioned in some clusters C_i. For the size of a cluster a lower bound L and an upper bound R is given. Then

$$cost(\{C_1, \ldots, C_k\}) = \sum_{i=1}^{k} |E(C_i)|$$

has to be minimized such that

$$L \leq w(C_i) \leq U \qquad \text{for all } i = 1, \ldots, k,$$

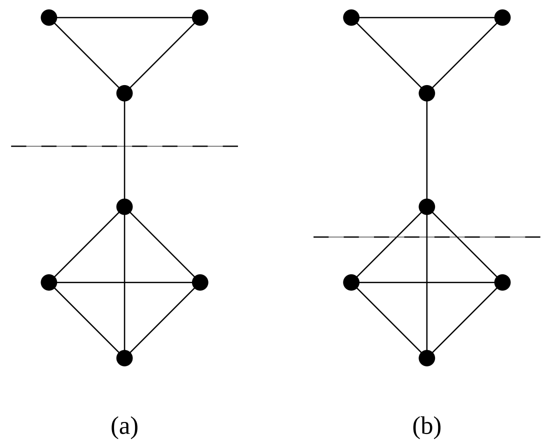

Figure 1: Graph with two possible partitions

where the cut $E(C)$ of a cluster C is given by the set of edges that have at least one, but not all pins in C, and $w(C)$ is the weight of a cluster C, given by the sum of the sizes of its elements.

Example 1 *Consider the two graphs given in Figure 1. They are split into 2 partitions. The first cut has size cost* $= 2$*, while the second one has cut size cost* $= 6$.

3 Partitioning Algorithm

We start with a description of the overall flow of the algorithm and then give the detailed parameters.

3.1 Basic Idea

The main components of the algorithm are briefly described before the details are given.

1. First the set of nodes is divided into two balanced partitions, using a KLFM-like algorithm ("bi-partitioning"). Then this algorithm is called recursively for both partitions, until the desired number of partitions is reached.

   A. Bi-partitioning is based on the following two steps. First, an initial balanced partition is generated using a BFS oriented heuristic: a node is selected at random, and some neighboring nodes are added until the partition is balanced. Then, a KLFM-like algorithm is used to iteratively improve this first partition.

B. It has already been observed by several authors (see e.g. [13]) that several runs have to be carried out to determine a good (and robust) solution. Thus Item A. is repeated and the best cut is taken. Here it should be noted that especially during the first few partitions computed in the recursion, this can become very expensive, since the whole graph has to be traversed several times. For this, we start with only one or two runs, but during the recursion this number is increased (see Section 3.3).

2. If k partitions have been obtained, a greedy algorithms starts to "move nodes around" to guarantee that the final solution is a (local) minima.

The first step is used to find a "reasonable good" starting point, while the quality of the overall algorithm mainly results from the second part[1]. In the following these main components are described in more detail.

3.2 KLFM-like Algorithm

The basic bi-partitioning algorithm is a KLFM-like [14, 7] iterative improvement algorithm that is based on a greedy strategy: iteratively nodes are selected and moved to the best (improving) partition.

To improve the quality, we use a look-ahead level [15] of 2, i.e. to evaluate the gain of a move, the best gain of any following move is also considered. It turned out in our experiments that this approach improves the result without being much slower.

3.3 Dynamic Parameter Selection

For one bi-partitioning step, several starting partitions can be chosen. Then KLFM is used to improve each of them, and the best cut is chosen. When using several starting partitions, the resulting cuts are usually much better and the algorithm becomes more robust. We will refer to the number of starting partitions as *#start* in the following.

The choice of the parameter *#start* has a direct impact on the run time, since KLFM has to be carried out *#start* times more often. Instead of fixing the parameter globally to a constant value (as it has been done in previous approaches), selecting this number is based on the recursion number in our algorithm. This is due to the following reasons:

[1] If the greedy algorithm is used as a post-processing to the solution determined by hMETIS, significant improvements can be observed, too.

- The first bi-partitioning step is the most expensive one, because the full set of nodes has to be considered. Further bi-partitioning steps have to deal with a rapidly decreasing number of nodes.
- The quality of the first cut does not have too much influence on the final result, while the latest cuts directly have an impact.

Therefore, in the first step, only one or two starting partitions are used. Whenever the recursion level is increased, also one or two additional starting partitions are used. (The choice whether one or two are considered influences the run time and the quality of the results. This will be further discussed in the description of the experimental results in the next section.)

3.4 Greedy Post-Processing

After an initial partitioning has been computed by the algorithm described above, we start the second step that can be described best as a greedy approach. The algorithm moves nodes across the k partitions based on a gain value that is computed analogously to the k-way approach in [19]. But to avoid the memory explosion described in Section 1.1 the gain value is only computed for each node separately. By this, only memory linear in the number of nodes (instead of quadratic) is needed.

A sketch of the algorithm is given in Figure 2. This algorithm is iterated several times using different choices for parameter *mingain*. This parameter controls the minimum improvement of each modification. It is chosen between 4 and -1.

4 Experimental Results

The algorithm described above has been implemented in C. All experiments have been carried out on a 550 MHz PentiumII PC running Linux. All run times are given in CPU seconds. The experiments are run on the same machine and in the same software environment. Some statistics about the ISPD98 benchmarks [1] are given in Table 2.

In a first series of experiments we compare the results of our algorithm to hMETIS [13, 12]. Each benchmark is partitioned in such a way that each final partition has 100 nodes. Thus up to 2000 partitions have to be computed. The final results and the corresponding run times are given in Table 3 and Table 4, respectively. In the first three rows the results for hMETIS using 4, 5, and 10 runs per recursive split are reported. In the last three rows the results obtained by

refine(starting partition P, number of partitions k, int $mingain$) {
    for each node n {
        let C_i be the partition to which n belongs
        if ($|C_i - \{n\}| > L$) { // L is lower bound on partition size
            for all partitions C_j ($j \neq i$) {
                if ($|C_j \cup \{n\}| \leq R$) { // R is upper bound on partition size
                    set P' to partition after moving n from C_i to C_j
                    set gain(C_j) to (cutset size(P)$-$cutset size(P'))
                } else {
                    set gain(C_j) to $-\infty$
                }
            } // end for
            if (max(gain(C_j)) $\geq$ $mingain$)
                move n from C_i to C_j
            } // end if
        } // end if
    } // end for
    return P
}

Figure 2: Sketch of GRE algorithm

the new algorithm are shown. In row N10 we give (for comparison reasons) the results obtained by our algorithm, if at each level 10 runs are carried out. In rows N+1 and N+2 starting with only 1 or 2 runs, the number is increased in each recursive split by 1 or 2, respectively. As can be seen all variants of our new algorithm constantly outperform hMETIS with respect to quality of the results (see Table 3). N+1 is the fastest algorithm of the ones proposed and even though it has better run time behavior than M4, it still outperforms M10 regarding quality. This mainly results from the powerful greedy algorithm as will be shown below. The KLFM-based first step is mainly used to avoid that the starting point is too bad, since this would increase run time significantly. But also the hMETIS results can be further improved, by our greedy algorithm resulting in cuts of similar quality as our approach.

Next, we show that the memory consumption can be further reduced, if we simplify the algorithm in the following way: We determine a starting point using a BFS algorithm and then

Table 2: Benchmark statistics

benchmark	nodes	edges
ibm01	12752	14111
ibm02	19601	19584
ibm03	23136	27401
ibm04	27507	31970
ibm06	32498	34826
ibm08	51309	50513
ibm10	69429	75196
ibm12	71076	77240
ibm14	147605	152772
ibm16	183484	190048
ibm18	210613	201920

Table 3: Min cut

	ibm01	ibm02	ibm03	ibm04	ibm06	ibm08	ibm10	ibm12	ibm14	ibm16	ibm18
M4	7947	23358	20490	26144	31157	46294	68476	80460	134977	177328	187538
M5	7982	23309	20429	25957	31040	46118	68630	81429	134237	177418	187947
M10	7980	23219	20417	26128	31157	46174	67791	81219	134144	176510	186883
N10	7855	22720	20479	25661	30548	45269	67395	80828	132726	175126	184134
N+1	7964	22557	20347	25792	30459	45406	67216	80369	133860	175810	184935
N+2	7945	22577	20476	25563	30732	44835	66771	80415	133082	173969	183905

directly apply the greedy refinement technique, i.e. we do not use the KLFM approach any more. The results for the final cut, the run times and the memory consumption (in MByte) for the greedy algorithm (denoted as GRE) in comparison to M4 and N+2 are given in Table 5. As can be seen the memory needed is reduced significantly. Furthermore, the quality can often be improved, but the run time is also larger, since the starting point is very bad and for this convergence takes longer. The run time can be further improved if the number of iterations is reduced, but then also the quality decreases. Here it is up to the designer to chose what is more important, run time or quality of the final result. But the choice of the number of runs allows

Table 4: Run times

	ibm01	ibm02	ibm03	ibm04	ibm06	ibm08	ibm10	ibm12	ibm14	ibm16	ibm18
M4	22	54	58	71	104	192	289	295	732	997	1135
M5	28	67	68	77	115	214	320	396	796	1138	1308
M10	54	124	127	151	209	399	570	633	1403	1974	2230
N10	40	93	107	119	170	346	460	471	1231	1529	2123
N+1	26	61	67	75	100	198	257	270	580	839	1054
N+2	34	83	87	99	138	291	364	380	896	1278	1580

Table 5: Comparison to greedy algorithm

	ibm01	ibm02	ibm03	ibm04	ibm06	ibm08	ibm10	ibm12	ibm14	ibm16	ibm18
M4											
cut	7947	23358	20490	26144	31157	46294	68476	80460	134977	177328	187538
time	22	54	58	71	104	192	289	295	732	997	1135
mem	4.9	8.6	9.7	11.1	12.5	20.1	29.3	31.7	56.7	76.6	81.4
N+2											
cut	7945	22577	20476	25563	30732	44835	66771	80415	133082	173969	183905
time	34	83	87	99	138	291	364	380	896	1278	1580
mem	4.6	6.7	8.5	9.6	11.3	17.4	23.8	25	48.8	62.1	68.4
GRE											
cut	7897	22479	20195	24889	30271	44681	65219	77496	130247	170968	181528
time	37	93	79	99	136	321	368	387	723	1185	1347
mem	1.6	2.2	2.6	3.0	3.3	4.9	6.8	7.1	13.1	17.0	18.5

for a “smooth” trade-off.

The experiments shown above give the impression that using GRE only was the best strategy, since it produces the best results using the smallest amount of memory. In the following we show that when using a small number of partitions only, it does make sense to use a KLFM-like algorithm first. In Figure 3, for benchmark ibm18 and a variety of partition numbers a comparison of M10, N+2, and GRE is given. Relative numbers compared to M10 are used. Thus, a value larger than 1.0 means that the method is worse than M10, while it is better for

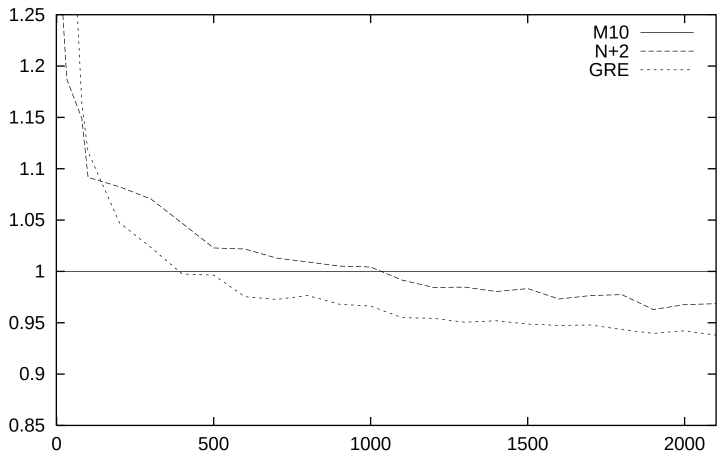

Figure 3: Comparison for different number of partitions (ibm18)

a value less than 1.0. For a small number of partitions, M10 produces much better results than GRE, while N+2 takes an intermediate place. However, for a growing number of partitions GRE also improves, and it is better than M10 for more than 500 partitions. To obtain a stable algorithm, i.e. a method that performs well independent of the number of partitions, it does make sense to use a combination of both approaches, i.e. to run the greedy algorithm after recursive bi-partitioning.

5 Conclusions

We presented a new method to improve recursive bi-partitioning of netlists for a very large number of partitions. The parameters for bi-partitioning are chosen dependent on the recursion level, since the quality of the cut is not very much of importance for the first cuts, while it has a large impact on the result for later cuts. Therefore, it is possible to speed up the algorithm for the first cuts where the number of nodes that have to be considered is still large. When the quality of the cuts becomes more important, much fewer nodes have to be considered and more iterations can be processed. The algorithm is further improved by using a post-processing step that allows to move nodes from one partition to another in a greedy fashion.

Using this post-processing step directly for the initial partitioning, i.e. without carrying out recursive bi-partitioning first, the memory can be further reduced, since no information for bi-partitioning has to be stored. Since more time can be spent for the greedy algorithm, also the quality of the results improves, if the number of partitions is large enough.

References

[1] C.J. Alpert. The ISPD-98 circuit benchmark suite. In *Int'l Symp. on Physical Design*, pages 80–85, 1998. See also http://vlsicad.cs.ucla.edu/∼cheese/ispd98.html.

[2] C.J. Alpert, J.-H. Huang, and A.B. Kahng. Multilevel circuit partitioning. In *Design Automation Conf.*, pages 530–533, 1997.

[3] C.J. Alpert and A.B. Kahng. Recent directions in netlist partitioning: A survey. *INTEGRATION, the VLSI Jour.*, 1-2(19):1–81, 1995.

[4] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. In *Design Automation Conf.*, pages 775–778, 1989.

[5] R. Drechsler, W. Günther, L. Linhard, and G. Angst. Level assignment for displaying combinational logic. In *EUROMICRO*, 2001.

[6] S. Dutt. New faster Kernighan-Lin-type graph-partitioning algorithms. In *Int'l Conf. on CAD*, pages 370–377, 1993.

[7] C.M. Fiduccia and R.M. Mattheyes. A linear-time heuristic for improving network partitions. In *Design Automation Conf.*, pages 175–181, 1982.

[8] Y.-S. Jehng, L.-G. Chen, and T.-M. Parng. ASG: Automatic schematic generator. *INTEGRATION, the VLSI Jour.*, 11:11–27, 1991.

[9] F.M. Johannes. Partitioning of VLSI circuits and systems. In *Design Automation Conf.*, pages 83–87, 1996.

[10] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. In *J. Graph Algorithms Appl.*, volume 1(1), pages 1–25, 1997.

[11] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Design Automation Conf.*, pages 526–529, 1997.

[12] G. Karypis and V. Kumar. *hMETIS: A Hypergraph Partitioning Package*. University of Minnesota, 1998. Also available at http://www.cs.umn.edu/∼karypis.

[13] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Design Automation Conf.*, pages 343–348, 1999.

[14] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[15] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. on Comp.*, 33(5):438–446, 1984.

[16] A. Kumar, A. Arya, V.V. Swaminathan, and A. Misra. Automatic generation of digital system schematic diagrams. *IEEE Design & Test of Comp.*, pages 58–65, 1986.

[17] M. Laguna, R. Martí, , and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers and Operations Research*, 24(12):1175–1186, 1997.

[18] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In *Graph Drawing Conference*, volume 1731 of *LNCS*, pages 217–224. Springer Verlag, 1999.

[19] L.A. Sanchis. Multiple-way network partitioning. *IEEE Trans. on Comp.*, 38(1):62–81, 1989.

[20] L.A. Sanchis. Multiple-way partitioning with different cost functions. *IEEE Trans. on Comp.*, 42(12):1500–1504, 1993.

[21] E. Yarack and J. Carletta. An evaluation of move-based multi-way partitioning algorithms. In *Int'l Conf. on Comp. Design*, pages 363–369, 2000.