

A Stable Partitioning Algorithm for VLSI Circuits*

Jong-Sheng Cherng and Sao-Jie Chen

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
E-mail: csj@cc.ee.ntu.edu.tw

Abstract

In this paper, a novel module migration based two-way partitioning algorithm is proposed to enhance the stability and quality of partitioning result. The proposed algorithm intensifies the capability of escaping from local optimal by releasing the size constraint temporarily and controlling the migration direction. And a circuit clustering procedure is incorporated into the algorithm to further improve the partitioning quality. Compared with the Fiduccia and Mattheyses [3] and Cheng and Wei [1] algorithms, the experimental results of our proposed algorithm show a significant improvement in most cases and outstanding performance in particular with large size circuits.

1. Introduction

With its increasing complexity, a VLSI circuit may contain millions of transistors. As a result, the size of problem has become so large that hierarchical approaches become essential to reduce the design complexity. Partitioning plays an important role in the hierarchical design of many physical design problems, such as circuit packaging, circuit layout, etc.. In partitioning electrical circuits, since the interconnection of nets among modules is the most essential, the main objective of many circuit partitioning algorithms is to keep the number of nets connecting different partitions minimal.

Since the problem of partitioning with size constraints is *NP-complete*, optimal solution is hard to obtain when the problem size is large and various heuristics have been developed. Kernighan and Lin [5] proposed a well-known module interchanging strategy for two-way partitioning which has become the basis of most iterative improvement partitioning algorithms. Fiduccia and Mattheyses [3] improved this algorithm by reducing time complexity to be in linear proportion to the pin number by employing a bucket list data structure. However, the Kernighan and Lin based algorithms have been observed to be highly sensitive to the choices of initial partition [6] [9]. As a consequence, many trials are needed to guarantee the quality of solutions. Also, the size of each partition must be predefined in the Kernighan and Lin based algorithms. A module migration process will be interrupted when the size constraints are broken, so the freedom of exploring the solution space is limited by this rule. The concept of ratio cut proposed by Wei and Cheng [8] [10] has received much attention recently. Since this ratio cut approach removes the

constraint on subset size and tends to identify natural clusters in the circuit, it generates better result.

On the other hand, clustering techniques have been found to be useful in reducing the problem complexity and producing stable partitioning result [2] [4] [7] [9]. A cluster is a group of highly connected components in a circuit. The goal of clustering algorithms is to identify the natural clusters in a circuit. Generally speaking, the clustering algorithms are used as a preprocessing step for partitioning. That is, clustering is first applied on the original network, and then partitioning is applied on the clustered network. There are approaches to achieve partitioning with this two-level scheme. For example, Cheng and Wei [1] [9] combined the ratio cut scheme [8] with the Fiduccia and Mattheyses [3] algorithm to obtain a stable performance two-way partitioning algorithm. At the first level of their approach, some highly connected components from the original circuit are clustered into groups by a ratio cut based top-down clustering technique to reduce the problem complexity. Then the Fiduccia and Mattheyses algorithm is applied to rearrange the groups into two subsets with prespecified sizes at the second level. Similarly, Shin and Kim adopted a bottom-up clustering technique to cluster highly connected components at the first level, then a gradual constraint-enforcing partitioning algorithm was used to complete the second level partition. The algorithm starts with a relaxed size constraint on subsets and then gradually tightens the size constraint until the required constraint is met. Cong and Smith [2] also presented a parallel bottom-up clustering algorithm to generate stable partitioning results. Additionally, Hagen and Kahng [4] used a random-walk method on the network to extract the global relations among modules and to discover natural clusters, then the Fiduccia and Mattheyses algorithm was incorporated to complete the second level partitioning.

2. Ratio Cut for Clustering

Since clustering-based two-level partitioning algorithms can produce partitioning results stabler than direct partitioning, we used the ratio cut technique as a preprocessing step for clustering the original network. The ratio cut method removes the size constraints on partitions and identifies natural clusters in a circuit, so it is suitable for ratio cut to achieve a good clustering result. According to Wei and Cheng [8] [10], the ratio of a cut is defined as $RC = \frac{C_{AB}}{|A|*|B|}$. The set of nodes V of a network $H = (V, E)$ is divided into two disjoint subsets $A \subset V$ and $B = V - A$ with $A \neq \emptyset$. The cut-size C_{AB} is the number of nets connecting the partitions A and B . The ratio cut is the cut that generates the minimum ratio among all cuts in the network. However, finding the ratio cut belongs to the class of *NP-complete*

*This work was supported by the National Science Council, R.O.C., under Grants NSC85-2221-E002-050 and NSC85-2215-E002-019

problems. The algorithm in [8] is adopted as our basic top-down clustering tool to recursively divide the circuit into small, highly connected groups. Next, a module migration based algorithm is applied to the contracted network, which nodes are groups of modules generated during the clustering procedure.

3. Module Migration Partitioning Algorithm

After clusters are formed by using a ratio cut based top-down clustering technique, a partitioning algorithm must be applied to rearrange the clusters into two subsets with prespecified sizes. For that purpose, we will develop a module migration partitioning algorithm. In a given circuit, if the connections among the modules are localized, some highly connected groups exist such that the connections among these modules are heavier than others, and dividing these groups into different partitions will increase the number of nets cut. Therefore, if we can move some modules to make these groups not to be cut, the net cut number will be fewer. This concept will be used as the main guideline in our proposed approach. An overview of our algorithm is shown as follows:

Algorithm : Module Migration Partitioning

- 1) Obtain an initial partitioning.
- 2) Set migration direction as from the left partition to the right partition.
- 3) Randomly select a module from the left partition.
- 4) Move the selected module and update associated information.
- 5) If the total accumulated size TAS of moved modules exceeds $GSLB$ and the reversing point is found, then go to Step 7; otherwise, go to Step 6.
- 6) Select from the left partition a module having a maximal gain and the strongest connection to previously moved modules, then go to Step 4.
- 7) Set migration direction as from the right partition to the left partition.
- 8) Randomly select a module from the right partition.
- 9) Move the selected module and update associated information.
- 10) If current partitioning result satisfies the size ratio and generates a minimal net cut, then output this best result and go to Step 12; otherwise, go to Step 11.
- 11) Select from the right partition a module having a maximal gain and the strongest connection to previously moved modules, then go to Step 9.
- 12) Update the $GSLB$ value.
- 13) Assign the current best result as a new initial partitioning and repeat Steps 2 to 12 until the iteration count is reached.
- 14) Output the current best result and use a self-adjusted probabilistic function set to update the values of starting $GSLB$ and net weighting. Repeat Steps 13 and 14 until the number of trials is reached.
- 15) Report the best partitioning result.

First, an initial partitioning solution is generated as a starting point, then we attempt to move the modules in highly connected groups of one partition into the other one to improve the existing solution. This is achieved by fixing the migration direction and choosing modules to be moved with great care. This "forward" migration process is continued until the assumed group is swept into the other side. Next, the migration direction is reversed and the modules are moved to keep the size ratio fall into an acceptable range. We call a *pass* is done when the size ratio falls below the acceptable range during the "backward" migration. The best feasible solution will be recorded any time it occurs and used to form a new starting point for the next *pass*. The above *pass* operation will be repeated

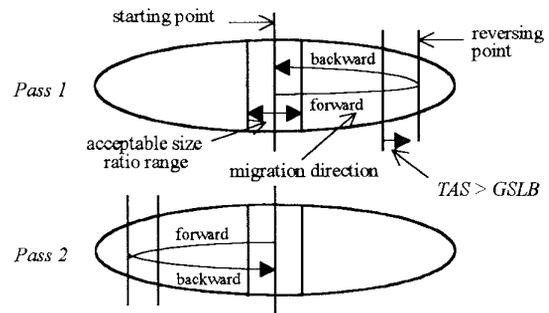


Fig. 1: *Pass* operation of the module migration algorithm.

iteratively to improve the result and the iteration count can be determined in advance. Note that we have relaxed the size constraints temporarily during a *pass* process. The basic operation of a *pass* (Steps 2 to 12 in the above Algorithm) is shown in Fig. 1, where the migration direction of each *pass* is changed alternatively.

We hope only modules in one highly connected group are moving during forward or backward migration. Thus it is undesirable to choose a module which has no connection to the previously moved modules as the next target to move. Therefore, a module m_i chosen for migration must have a maximal gain (here gain means the decrement of the net cut number if m_i is moved into the opposite partition) and have the strongest connection to previously moved modules. In Step 5 of the above Algorithm, $GSLB$, which is the *guessed size lower bound* of the maximal module set in the highly connected group that we want to move, is devised to help moving modules efficiently. Heuristically, we hope modules in larger groups be first moved into one side, and then modules in smaller groups. Therefore, the value of $GSLB$ is set to be high at the first *pass*, and will be gradually decreased at the following *passes*.

Backward migration is essential to search acceptable size ratio results and to find better solution for each *pass*. Naturally, the change of migration direction occurs when modules in a highly connected group have just been moved into one side. By direct observation, when such a group is moved into the opposite side, the net cut number will fall into a minimal point. If we still continue the moving operation unaware, the net cut number will certainly increase. Under this consideration, when the accumulated size of moved modules is greater than the given $GSLB$, we begin to keep record of the decreasing net cut number and it is time to reverse the migration direction when this value begins to increase. Since the moving gain of candidate modules in a module migration based algorithm can reflect this value changing, we need only to check the "moving gain" at the implementation, and this is one of the factors that makes our module migration based algorithm efficient.

Since weighting nets (especially those multi-pin nets) in a partitioning graph and setting the $GSLB$'s starting value are two important parameters in our algorithm, all these parameter settings must be managed efficiently. To reduce the sensitivity to the variety of parameters, we hope the algorithm has the ability to search for an optimal parameter range automatically. Therefore, we need a set of probabilistic functions to aid the algorithm in finding a

9.1.2

better value range for each of these parameters iteration by iteration. The function of such a parameter management scheme is briefly described as follows: if the current value of a certain parameter used in the algorithm produces most of times good (bad) results, the probability that setting the next iteration value of this parameter close to its current value must be kept high (low).

Since the result from just a single run of the proposed module migration partitioning algorithm is very stable, we have only to apply the algorithm just five times to obtain a good result on the contracted network. Finally, the last run of the proposed algorithm will be applied to the original flattened network to fine tune the partitioning result. In next section, experiments on partitioning benchmarks are performed to verify the superiority of our proposed algorithm.

4. Experimental Results

Our clustering-based module migration (CMM) algorithm was coded in C language and implemented on a Sun SPARC workstation. For comparison, we implemented the Fiduccia and Mattheyses (FM) algorithm [3] and the Cheng and Wei (CW) algorithm was from [1]. We tested our algorithm using 18 circuits as listed in Table 1. The runs of the CW and our CMM algorithms are set to be 20. The g value of the CW algorithm is set to be 50 based on [1]. We set the run of the FM algorithm to be 100 for meaningful comparison. By experimental results, we demonstrate that our partitioning algorithm outperforms the FM and CW algorithms in terms of the minimal net cut number and the average net cut value.

In Table 1, we compare our result to the FM and CW results with the size of each partition being allowed to have 1% deviation from exact bipartition. For most circuits, the CW algorithm performs better than the FM algorithm in terms of the average net cut value, but the situation is opposite in terms of the best minimal net cut number. However, the proposed CMM algorithm generates better results than those of the FM and CW algorithms for both terms. For instance, in the example of *industry2*, although much improvement to the FM algorithm is achieved by the CW algorithm (1098 \rightarrow 938), the minimal net cut number is still high compared with our result (197). The average net cut value also drops from 2028.53 to 1139.90 to 241.65. These are 82% and 79% improvements for the minimal net cut number, and 88% and 79% improvements for the average net cut value, over the FM algorithm and the CW algorithm, respectively. On average, our module migration partitioning algorithm shows a 27% and 26% improvements over the best results and a 51% and 45% improvements over the average results achieved by the FM algorithm and the CW algorithm, respectively, for all the 18 test circuits. For huge circuits like *avg.small* and *avg.large*, although our algorithm spends approximately three times of run time than the CW algorithm, it is worthy for us to achieve 58% and 64% improvements for the minimal net cut number, and 50% and 53% improvements for the average net cut value over the CW algorithm, for the circuits *avg.small* and *avg.large*, respectively. From Table 1, we also find that the CMM and CW algorithms have the same time magnitude, and the average time used in the FM algorithm is much less than others. Although the FM algorithm works very fast for each run, more than 100 runs are required to derive the same solution quality as our CMM algorithm. Therefore, our

approach is superior to the FM method with the whole performance consideration.

In next experiment, we relax the deviation such that each partition is allowed to have a up to 10% deviation from bipartition. Table 2 presents the minimal net cut number and the average net cut value in this interval. On average, the CMM approach yields 23% and 47% improvements in terms of the minimal net cut number and the average value compared to the FM approach. For the average net cut value, the improvement of the CMM approach over the CW approach ranges from 1% (*primGAI*) to 43% (*industry2*), with an average of 9%. For the minimal net cut number, the CMM method outperforms the CW method on most circuits, and about an average of 8% improvement is obtained.

Since we have put a rather large tolerance on the size constraints (1 : 1.222) in Table 2, all of the three algorithms have more chances to search for better partitionings, therefore, they generate similar solutions for some circuits. But for large circuits like *industry2*, *industry3*, *avg.small*, and *avg.large*, the CMM algorithm obtains outstanding performance over other algorithms not only in Table 2 but also in Table 1. Additionally, the stability of the FM method is worse than the others when comparing the differences between average values and best values for circuits. According to the experimental results of both tables, we show that in the case where a strictly balanced partition is required, both the FM and CW algorithms have a poor performance. On the contrary, the behavior of the proposed CMM algorithm is very stable all the way. This shows that for two-way partitioning, our module migration algorithm is a better choice than the FM and CW algorithms.

5. Conclusion

A new two-way partitioning algorithm has been developed, where we adopted an efficient module migration scheme to release the size constraints temporarily and to control the migration direction. Experimental results obtained indicate that our proposed module migration partitioning algorithm improves the unstable property of conventional module migration based algorithms and outperforms the Fiduccia and Mattheyses algorithm [3] and the Cheng and Wei algorithm [1] in terms of the minimal net cut number and the average net cut value with the prespecified size constraints. Furthermore, our heuristic algorithm generates even better performance on large circuits than the others with reasonable runtime. Especially, our algorithm is very stable even when the deviation is quite small, that is, the solution quality of our proposed algorithm is less sensitive to the choice of the deviation.

Acknowledgements

The authors are thankful to Professors C. K. Cheng and C. W. Yeh for their supplying us both the testing benchmarks and the ratio cut program.

References

- [1] C. K. Cheng and Y. C. Wei, "An improved two-way partitioning algorithm with stable performance," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 12, pp. 1502-1511, Dec. 1991.
- [2] J. Cong and M. Smith, "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design," in *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 755-760.

- [3] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partition," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175-181.
- [4] L. Hagen and A. Kahng, "A new approach to effective circuit clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1992, pp. 422-427.
- [5] B. W. Kemighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291-307, Feb. 1970.
- [6] T. K. Ng, J. Oldfield, and V. Pitchumani, "Improvements of a mincut partition algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 470-473.
- [7] H. Shin and C. Kim, "A simple yet effective technique for partitioning," *IEEE Trans. VLSI System*, vol. 1, no. 3, pp. 380-386, Sept. 1993.
- [8] Y. C. Wei and C. K. Cheng, "Towards efficient hierarchical designs by ratio cut partitioning," in *Proc. IEEE Int. Conf. Computer Aided Design*, 1989, pp. 298-301.
- [9] Y. C. Wei and C. K. Cheng, "A two-level two-way partitioning algorithm," in *Proc. IEEE Int. Conf. Computer Aided Design*, 1990, pp. 516-519.
- [10] Y. C. Wei and C. K. Cheng, "Ratio cut partitioning for hierarchical designs," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 7, pp. 911-921, Jul. 1991.

Table 1
Partitioning results allowing 1% deviation from bipartition.

Example	# nets	# modules	# pins	FM			CW			CMM			Improvement (%)			
				min	avg	sec/run	min	avg	sec/run	min	avg	sec/run	I ₀	I ₁	I ₂	I ₃
19ks	3343	2684	10387	137	181.32	11.14	154	184.45	65.35	117	125.85	74.58	15	31	24	32
19kstw	3731	3079	11238	98	161.89	13.53	103	151.60	86.23	90	101.35	119.11	8	37	13	33
primGA1	904	752	2707	48	77.62	1.60	51	70.90	13.59	47	48.75	15.99	2	37	8	31
primGA2	3029	2907	10959	161	281.88	16.30	169	266.85	73.72	134	139.80	72.58	17	50	21	48
primSC1	904	752	2707	50	80.45	1.59	50	71.75	13.23	47	49.15	16.67	6	39	6	31
primSC2	3029	2907	10959	172	286.55	16.00	172	249.50	72.79	133	135.10	69.88	23	53	23	46
test02	1866	1663	6220	139	238.50	2.65	108	163.25	28.24	90	99.80	82.12	35	58	17	39
test03	1699	1607	5866	90	132.21	3.70	71	112.20	27.66	56	58.50	39.99	38	56	21	48
test04	1738	1515	5998	78	92.82	1.35	45	73.15	25.56	44	44.00	88.22	44	53	2	40
test05	2910	2595	10121	71	113.53	2.46	64	92.25	52.36	51	51.60	106.09	28	55	20	44
test06	1745	1752	6728	68	88.80	4.88	70	85.20	34.15	60	67.55	39.20	12	24	14	21
8870	349	286	1138	16	41.12	0.35	18	34.95	4.22	14	14.90	9.01	13	64	22	57
5655	843	801	2755	54	84.03	1.20	58	76.65	13.45	49	51.45	19.61	9	39	16	33
industry1	2594	2271	7721	34	78.33	7.73	34	77.95	54.51	24	27.10	52.69	29	65	29	65
industry2	13915	12142	47232	1098	2028.53	10.58	938	1139.90	191.24	197	241.65	377.37	82	88	79	79
industry3	21966	15059	65356	360	688.98	56.97	409	804.50	726.43	272	290.85	530.90	24	58	33	64
avg.small	22124	21918	76231	336	638.33	509.13	501	615.65	2246.57	211	307.90	6456.26	37	52	58	50
avg.large	25384	25178	82751	432	825.57	634.45	536	711.05	2544.80	193	336.90	7775.00	55	59	64	53
Average													27	51	26	45

Note: $I_0 = (\min_{fm} - \min_{cmm}) / \min_{fm}$ $I_1 = (\text{avg}_{fm} - \text{avg}_{cmm}) / \text{avg}_{fm}$
 $I_2 = (\min_{cw} - \min_{cmm}) / \min_{cw}$ $I_3 = (\text{avg}_{cw} - \text{avg}_{cmm}) / \text{avg}_{cw}$

Table 2
Partitioning results allowing up to 10% deviation from bipartition.

Example	FM		CW		CMM		Improvement (%)			
	min	avg	min	avg	min	avg	I ₀	I ₁	I ₂	I ₃
19ks	133	178.41	118	123.00	112	121.80	16	32	5	1
19kstw	97	167.32	90	96.50	79	88.00	19	47	12	9
primGA1	45	67.80	42	42.65	42	42.30	7	38	0	1
primGA2	169	280.22	120	122.60	120	121.95	29	56	0	1
primSC1	44	71.28	42	42.95	42	42.50	5	40	0	1
primSC2	158	282.02	119	124.65	119	123.05	25	56	0	1
test02	128	192.59	81	98.50	75	85.35	41	56	7	13
test03	69	131.98	56	61.10	55	56.15	20	57	2	8
test04	44	48.34	44	44.95	42	42.80	5	11	5	5
test05	42	60.70	42	43.20	42	42.00	0	31	0	3
test06	60	85.02	60	69.60	60	62.50	0	26	0	10
8870	14	22.47	14	15.10	14	14.35	0	36	0	5
5655	52	69.23	49	53.10	47	48.25	10	30	4	9
industry1	25	70.52	20	24.80	20	22.35	20	68	0	10
industry2	1429	1991.48	346	422.90	178	239.85	88	88	49	43
industry3	261	653.89	211	252.45	190	220.20	27	66	10	13
avg.small	341	638.75	270	373.20	191	286.85	44	55	29	23
avg.large	449	815.09	212	353.40	160	319.20	64	61	25	10
Average							23	47	8	9

9.1.4