Simple Yet Effective Techniques to Improve Flat Multiway Circuit Partitioning

S.M. Areibi
School of Engineering
University of Guelph
Guelph, Ontario, Canada
sareibi@uoguelph.ca

Abstract

Partitioning plays an important role in solving large, complicated design problems. For large benchmarks iterative improvement methods based on the Conventional Kernighan and Lin Heuristic [1] produce inferior results. These algorithms suffer from a propensity to freeze the movement of large cells that would immediately violate the balance constraint. To enhance the stability and quality of partitioning results a novel algorithm intensifies the capability of escaping from local optimal by first releasing the size constraint temporarily and enforcing it at the end of a run and controlling the migration direction and secondly by adding a dynamic hill climbing capability to avoid the possibility of getting trapped in a local minima. These simple heuristic techniques reduce the total cut on average by 30% and runtime by at least 70-90%.

1 Introduction

Circuit partitioning divides a given circuit into a collection of smaller sub-circuits to minimize the number of connections among the sub-circuits, subject to the area balance constraint. The circuit partitioning problem becomes more important as VLSI technology reaches sub-micron device dimensions. Traditionally, this problem was important for breaking up a complex system into several custom ASICs. Now, with the increasing use of FPGA-based emulators and prototyping systems, partitioning is becoming even more critical. While it is possible to solve the case of unbounded partition sizes exactly, the case of balanced partition sizes is NP complete [2]. As a result, numerous heuristic algorithms have been proposed [3]. A circuit netlist is usually modeled by a hypergraph **G = (V,E)**, where **V** is the set of cells in the circuit, and **E** is the set of nets (also called the hyperedges). A two-way partition of **G** is two disjoint subsets V_1 and V_2 such that each cell $v \in V$ belongs to either V_1 or V_2. A net is said to be cut if it has at least one cell in each subset and uncut otherwise. The objective is to partition the circuit into parts such that the sizes of the components are within prescribed ranges and the complexity of connections between the components is minimized. There are two primary approaches for generating multi-way partitioning solution; recursive or flat. The recursive approach applies bi-partitioning recursively until the desired number of partitions is obtained, whereas the flat approach partitions the circuit directly. In experiments on large benchmarks with the multi-way partitioning algorithm K-FM[4], it was observed in [5] that recursive bi-partitioning provided superior results. In practice, it appears that the flat multi-way iterative improvement approach encounters a large number of local minima which are difficult to escape [6]. Little progress has been made to improve iterative improvement-based flat multi-way partitioning algorithms such as K-FM despite the potential gain from the availability of more global information and larger solutions space [5]. In addition while recursive bi-partitioning can be quite effective in minimizing cut sizes, solutions developed in this manner are less useful in circuit placement applications (with the recursive formulation we lose the ability to accurately model inter-cluster routing costs)[6].

2 Previous Work

Kernighan and Lin (KL) [1] proposed a two-way graph partitioning algorithm which became the basis for most of the subsequent partitioning algorithms. The KL algorithm operates only on balanced partitions[1] and performs a number of passes over the

cells of the circuit where each pass comprises a repeated operation of pairwise cell swapping for all pairs of cells. Fiduccia and Mattheyses (FM) [7] obtained a faster implementation of KL with the help of a new data structure, called the bucket data structure. FM can operate on unbalanced partitions and employs a single cell move instead of a swap of a cell pair at each step in a pass. Krishnamurthy [3] added to FM a look-ahead ability (LA), which helps to break ties better in selecting a cell to move. Sanchis [3] generalized Krishnamurthy's algorithm to a multi-way circuit partitioning algorithm (MFM). Dutt and Deng [8] proposed a new iterative improvement method (CLIP) that selects cells to move with a view to moving clusters that straddle the two subsets of a partition into one of the subsets. Recently some new approaches that enhanced the performance of the original KL algorithm appeared [6] [5] and the reader is referred to the excellent survey in [3]. These iterative improvement approaches generally operate in a series of passes. Starting from an initial random partitioning, vertices are moved from cluster to cluster, with each vertex being moved once in a pass. After all vertices have been moved, a move-based iterative improvement approach rolls back to the best intermediate solution, and begins another pass. Iterative improvement techniques based on module interchange are the most robust, simple and successful heuristics in solving the partitioning and placement problems. The main disadvantage of these heuristics is that they mainly focus on the immediate area around the current initial solution, thus no attempt is made to explore all regions of the parameter space. More importantly, it has been shown that interchange methods fail to converge to "optimal" or "near optimal" solutions unless they initially begin from "good" initial starting points [9]. Sechen [10] showed that over 100 trials or different runs were required to guarantee that the best solution would be within twenty percent of the optimum solution.

3 Simple Effective Methods

FM and LA are the most commonly used two-way partitioning algorithms largely due to their excellent run times, simple implementation and flexibility. However, this class of iterative improvement algorithms have a common weakness (as explained in the previous section),viz., they only find solutions corresponding to local minima. In this section we will propose simple yet effective techniques that significantly improves the ability of FM and LA for finding good local minima. The new techniques first relax the constraint of size, giving the algorithm more degrees of freedom in exploring the solution space. In addition a dynamic hill climbing technique helps the algorithm to escape getting trapped in local optimum.

3.1 Relaxation of the Size Constraint

The partitioning problem for any given objective is inherently a constraint-driven one. The most common constraint is the balance ratio of total sizes that the two subsets of a partition must satisfy [11]. Other frequently encountered partitioning constraints include limits on the sizes, timing minimization, pins in multiple-FPGA partitioning.

```
While (Stopping Criteria is not met)
   Set All Modules Status to Free
   Relax Size Constraint to ffi %
        While (Modules can be Moved)
          Select node a_i with highest gain
          Move a_i to destination block
          Mark a_i as locked
          Update Gains of effected Modules
EndWhile
Calculate deviation of all blocks
Determine the gain of all modules
While (Block Sizes do not Meet their Target)
   Choose Block with Maximum Oversize
   Choose Block with Maximum Undersize
   Move Modules from Oversized to Undersized Block
     if  Balance Criteria is OK
        Choose Alternative Under/Over Sized Blocks
EndWhile
Record Best Solution
```

Figure 1: A Relaxed Size Constraint Driven Iterative Improvement Technique

This balance-constrained partitioning problem can be stated as follows. Given a netlist **G** which describes cell connectivities and cell sizes in a circuit, construct k sub-circuits of **G** with a balance ratio of $r_1 : r_2 : ... : r_k$ and an acceptable tolerance of $\pm t$ such that some objective functions are optimized. In an attempt to satisfy the balance-ratio, however, movement of larger cells and clusters can be restricted thus leading to sub-optimal results. One of the fundamental constraints in bi-partitioning is the required balance in the sizes of V_1 and V_2. Assuming $|V_1| \leq |V_2|$, the constraint can be stated as $r - t \leq \frac{|V_1|}{|V_1+V_2|} \leq r + t$.

It is well documented [11] that the qualitative figures of merit of most partitioning algorithms are very sensitive to size constraints. Constraint satisfaction during partitioning and placement of VLSI circuits is an important problem, and effective techniques to address it lead to high-quality physical design solutions.

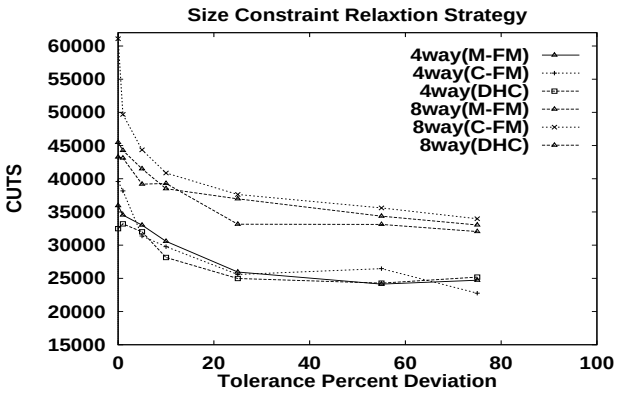

Figure 2: δ Percent Tolerance Parameter

The idea behind the relaxation hill climbing approach is illustrated in Figure 1. The main difference between our approach and the traditional MFM approach is that we allow a δ tolerance in the block size deviation. At the end of a run the system corrects the required deviation in a greedy manner. Our work presented here addresses the balance constraint in a different manner than that of [11]. The motivation behind this work is that during the iterative improvement process 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. Relaxed partitioning helps in removing natural clusters that straddle the cut-line by blocks of unidirectional moves that temporary violate the balance constraint; in a non-relaxed partitioner, these clusters can get locked in the cutset due to alternating moves in the two directions. Figure 2 illustrates the relationship between δ (tolerance percenet deviation) and quality of cut size for 4-way and 8-way partitioning schemes. The parameter δ works very well for MFM, CLIP and DHC between the range of 60%-80%.

3.2 Dynamic Hill Climbing

Dynamic hill climbing (DHC) is basically a modified implementation of the Sanchis iterative improvement heuristic (MFM). It is characterized by the ability of escaping local optimum, which usually cause simple descent algorithms to terminate, by dynamically taking a different direction of a steepest ascent. At each iteration during a pass, the best move is chosen. Passes are performed until no improvement in cut-net size is obtained. In SDHC, after the termination of the above heuristic, the heuristic considers all possible moves of each free cell from its home block to any of the other blocks, such that the value of the cut-size is increased. This is done such that the direction of the upward slope is different than the last pass performed. As seen in Figure 3 the heuristic continues to explore new regions until either cycling occurs or a certain number of passes have elapsed. Figure 4 compares the performance of the Sanchis heuristic to that of SDHC.

```
Pass = 0
While (Stopping Criteria is not met)
    Pass = Pass + 1
    START DESCEND ROUTINE
        Mark all nodes as not yet moved
        While (Modules can be Moved)
            Select node a_i with highest gain
            if Balance Criteria is OK
              Move a_i to destination block
              Mark a_i as locked
            End if
        end while
        Choose k nodes, which maximize G = sum_{i=1}^{k} g_i
        Perform Move on nodes a_1 to a_k
    END DESCEND ROUTINE
    START ASCENT ROUTINE
        Mark all nodes as not yet moved
        While (Modules can be Moved)
            Select node a_i with lowest gain
            if Balance Criteria is OK
              Move a_i to destination block
              Mark a_i as locked
            End if
        end while
        Choose k nodes, which minimize G = sum_{i=1}^{k} g_i
        Perform Move on nodes a_1 to a_k
        If (Cycling Occurs)
            Terminate Search
    END ASCENT ROUTINE
EndWhile
Record Best Solution
```

Figure 3: A dynamic hill climbing heuristic (SDHC)

The figure clearly indicates that once the Sanchis interchange technique stops at a local minima, SDHC focuses the search on other parts of the solution space to ensure that other regions are explored. It is worth noting that the complexity of the SDHC heuristic is similar to that of Sanchis[4].

4 Computational Results

We experimentally evaluated the quality of partitions produced by our new simple effective partitioning techniques on some hypergraphs that are part of widely used ACM/SIGDA circuit partitioning benchmarks suite. The characteristics of these hypergraphs

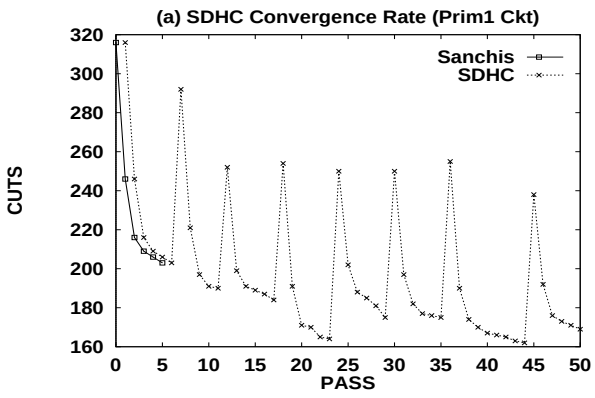

Figure 4: The convergence of SDHC

are shown in Table 1. The proposed method is implemented in C language on a Sun Ultra10 workstation. Tables 2 and 3 summarize the results obtained for 4-way and 8-way partitions. The first column presents the results based on the conventional multi-way FM technique (MFM), the second column presents the results with the size constraint relaxed MFM-RS (both based on 100 runs). Column 3 and 4 in the Tables present results based on CLIP [8] with and without relaxing the size constraint (i.e MCL and MCL-RS) using 20 runs. Finally the Dynamic Hill Climbing heuristic is run using 5 random initial partitions with and without relaxing the size constraint (i.e MDC and MDC-RS).

As can be seen in Tables 2 and 3, the quality of solutions obtained by SDHC are better than those obtained by the Sanchis heuristic in less computation time. The tables clearly indicate a tremendous improvement in cut-size on average by 30% and 80-90% in computation time when the size constraint is relaxed. It is interesting to note that CLIP [8] works well for bi-partitioning (i.e recursive partitioning) but when applied to multi-way flat partitioning would only perform well when the size constraint is relaxed and enforced at a later stage.

5 Conclusion & Ongoing Work

In this paper, we have presented simple effective techniques to enhance the performance of Sanchis multi-way partitioning heuristic which is an extension to FM bi-partitioning technique. The main contribution of our study is to present simple yet effective means to overcome the drawbacks of MFM. The result is an effective and efficient algorithm MDC-RS that improves conventional flat multi-way partitioning algorithm significantly. Our ongoing study includes adaption of Tabu Search and clustering to further reduce cut-size and runtime.

REFERENCES

[1] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

[2] M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, San Francisco CA, 1979.

[3] C.J. Alpert and A.B. Kahng. Netlist Partitioning: A Survey. *Integration, the VLSI Journal*, pages 64–80, 1995.

[4] L.A. Sanchis. Multiple-Way Network Partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.

[5] J. Cong and S. Lim. Multiway Partitioning with Pairwise Movement. In *IEEE International Conference on CAD*, pages 512–516, November 1998.

[6] P. Madden. Partitioning by Iterative Deletion. In *International Symposium on Physical Design*, pages 83–89, March 1999.

[7] C.M. Fiduccia and R.M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proceedings of 19th DAC*, pages 175–181, Las Vegas, Nevada, June 1982. ACM/IEEE.

[8] S. Dutt and W. Deng. VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques. In *IEEE International Conference on CAD*, pages 194–200. ACM/IEEE, 1996.

[9] S. Areibi and A. Vannelli. Circuit Partitioning Using a Tabu Search Approach. In *IEEE International Symposium on Circuits and Systems*, pages 1643–1646, Chicago, Illinois, 1993.

[10] C. Sechen and D. Chen. An improved Objective Function for Min-Cut Circuit Partitioning. *IEEE Transaction on CAD*, pages 502–505, 1988.

[11] S. Dutt and H. Theny. Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations. In *IEEE International Conference on CAD*, pages 350–355, November 1997.

Circuit	Nodes	Nets	Pins	Node Degree			Net Size		
				MAX	$\overline{x}$	œ	MAX	$\overline{x}$	œ
baluP	801	735	2626	9	3.27	1.38	117	3.5	5.19
Prim1	833	902	2908	9	3.49	1.28	18	3.22	2.59
Prim2	3014	3029	11219	9	3.72	1.55	37	3.70	3.81
Ind1	2271	2192	7743	10	3.41	1.14	318	3.53	9.00
Bio	6514	5742	21040	6	3.22	1.06	861	3.66	20.8
s13207	8772	8651	20606	5	2.34	0.68	38	3.38	1.81
s15850	10470	10383	24712	5	2.36	0.59	35	3.02	1.73
Ind2	12637	13419	48158	12	3.81	1.80	585	3.58	10.9
Ind3	15057	21808	65416	12	4.34	1.47	325	2.99	3.23
avq.small	21918	22124	76231	7	3.47	1.40	4042	3.44	53.3
avq.large	25178	25384	82751	7	3.28	1.29	4042	3.25	49.8
ibm01	12752	14111	50566	39	3.96	2.32	42	3.58	3.34
ibm02	19601	19584	81199	69	4.14	2.29	134	4.14	5.45
ibm03	23136	27401	93573	100	4.04	3.44	55	3.41	3.10
ibm04	27507	31970	105859	526	3.84	4.65	46	3.31	2.92

Table 1: Benchmarks used as test cases

Circuit	**MFM(100)**		**MFM-RS(100)**		**MCL(20)**		**MCL-RS(20)**		**MDC(5)**		**MDC-RS(5)**	
	Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time
baluP	163	0:14	100	0:21	141	0:3	139	0:3	151	0:4	126	:2
prim1	144	0:16	144	0:24	158	0:6	157	0:4	141	0:5	161	:2
prim2	614	1:34	599	1:39	779	0:38	548	0:19	609	0:22	647	:9
ind1	236	0:52	152	1:01	379	0:17	163	0:12	214	0:12	188	:6
biom	532	6:05	445	3:15	712	1:44	489	0:46	513	1:06	337	:22
s13207	540	6:41	251	6:34	492	2:03	351	1:19	445	1:10	252	:32
s15850	708	7:50	398	8:19	620	2:29	464	1:36	624	1:26	418	:39
ind2	1759	21:1	1748	8:49	2153	8:57	1050	2:12	1792	3:13	1644	:53
ind3	1663	13:0	1570	12:3	5006	16:4	2405	3:30	1679	2:46	1768	1:19
avq.s	2151	42:1	1813	12:2	2784	17:5	1384	7:16	2161	4:53	1778	1:23
avq.l	2545	53:1	2058	14:4	3187	21:3	1464	7:37	2349	6:33	1833	1:33
ibm01	3478	34:1	1941	15:3	3678	15:5	1675	2:46	2406	4:10	1858	1:29
ibm02	6356	123:	2367	18:1	6719	48:4	2611	3:30	5937	14:4	2612	2:15
ibm03	6771	113:	5104	36:5	6742	42:2	4943	5:44	6210	10:3	4639	3:10
ibm04	8291	181:	6033	39:2	8834	90:3	4923	9:36	7258	14:3	6022	3:20
Total	35951	609:	24723	183:	39600	292:	22766	40:0	32489	65:0	24283	15:0
%Imp	0%	0%	31%	69%	-9%	52%	36%	93%	10%	89%	32%	97.5%

Table 2: 4-Way partitioning

Circuit	**MFM(100)**		**MFM-RS(100)**		**MCL(20)**		**MCCL-RS(20)**		**MDC(5)**		**MDC-RS(5)**	
	Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time
baluP	224	0:22	157	0:28	235	0:5	163	:6	215	:14	158	0:4
prim1	189	0:22	186	0:34	232	0:6	203	:6	179	:13	208	0:5
prim2	804	3:32	731	2:36	920	0:58	747	:30	808	1:34	663	0:25
ind1	347	1:50	300	1:26	539	0:33	288	:22	372	:45	276	0:17
biom	789	13:50	390	5:27	1066	4:18	567	1:22	792	5:47	500	0:48
s13207	728	12:04	330	10:14	886	5:50	427	2:38	669	4:06	341	1:29
s15850	932	15:08	489	13:45	1186	5:55	547	3:05	840	4:55	498	1:44
ind2	2141	47:23	1948	13:49	2971	25:35	1739	4:31	2143	14:32	1815	2:07
ind3	2862	30:00	2620	20:51	7119	45:02	3861	14:26	2989	9:58	2874	2:48
avq.s	2537	89:46	2313	18:26	5297	62:23	1978	14:02	2778	22:46	1915	2:45
avq.l	2886	110:60	2517	21:45	5988	73:41	2065	15:14	2989	24:28	1835	3:15
ibm01	4629	111:18	2681	23:10	5267	46:11	2777	9:34	3845	29:10	2751	2:55
ibm02	7854	357:49	4086	58:37	8151	141:46	4154	11:26	7210	112:37	4131	6:57
ibm03	8318	394:14	6192	47:37	10029	116:34	6701	17:29	7839	78:16	6140	6:26
ibm04	10239	695:56	8099	68:07	11206	195:23	7766	27:08	9606	112:12	7938	7:59
Total	45479	1877:0	33039	305:0	61092	717:0	33983	123:0	43274	423:0	32043	32:0
%Imp	0%	0%	27%	83%	-25%	61%	25%	93%	4.8%	77%	29.5%	98.2%

Table 3: 8-Way partitioning