Performance-Driven Partitioning using a Replication Graph Approach

Lung-Tien Liu, Ming-Ter Kuo, Chung-Kuan Cheng, and Te C. Hu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Abstract— An efficient algorithm is proposed to tackle the performance-driven partitioning problem using retiming and replication. We devise a *replication graph* to model the composite effect of replication and retiming. With the replication graph, we formulate the problem as an integer linear programming problem. A heuristic algorithm is derived to solve the problem by exploring the dual program of its linear programming relaxation.

1 Introduction

Due to the physical geometric distance and interface technology limitations, intermodule delay is contributing the dominant portion of signal propagation delay. Consequently, we should take into account the intermodule delay for performance-driven partitioning.

For partitioning problem with timing and size constraints, Shih *et al.* [12, 13] propose a algorithm to guarantee that the delay between registers satisfies the timing constraint. In [9, 10], Liu *et al.* propose the partitioning algorithms using a *retiming* technique [6] to explore the ultimate clock period of the circuit.

As mentioned in [9], replication [5, 4] on parts of the circuit can improve the clock period of the partition. However, the combination of replication and retiming complicates the partitioning problem. No systematical solution has been proposed yet.

In this paper, an efficient algorithm is proposed to tackle the performance-driven partitioning problem using retiming and replication.

2 Statement of Problem

In this section, we first take an example to show that replication can improve the crossing edge count and the clock period simultaneously. Then we introduce the definitions and state the performance-driven partitioning problem.

2.1 Motivation of This Paper

Fig. 1 shows a circuit of six combinational elements (in circles) and five registers (in retangles). Each combinational element has its delay and size, while all registers have zero delays and sizes. Each combinational element is labeled with a delay (d). As in [9, 10], we assume that the combinational blocks are fine-grained and therefore can be split or merged. We also assume that each register in the circuit has a single input and a single output since a physical circuit can be transformed into this way as shown in section 8 of [6].

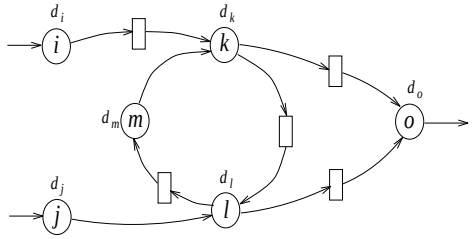

Figure 1: *A synchronous digital circuit*

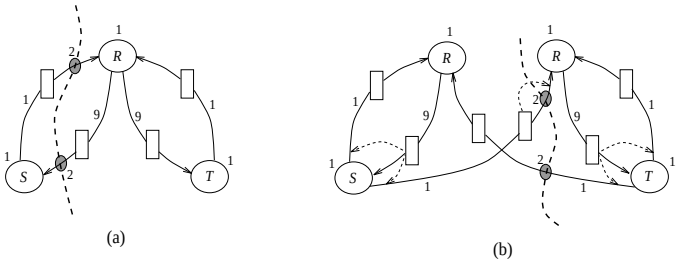

Figure 2: *Replication improves partitioning*

In Fig. 2, we show an example that replication can improve the crossing edge count and performance. Each edge in Fig. 2 is labeled by its capacity and associated with one register. In the partitioned circuits, the intermodule delays are depicted as shaded nodes and all are assumed to have the same delay of value 2. For a partition in Fig. 2(a), the clock period is equal to 3. Loop (S, R, S) has total delay time 6 and register count 2. Thus, the delay-to-register ratio of this loop is equal to 3. Based on this partition, we cannot get a clock period less than 3 even by retiming. Furthermore, the partition has a crossing edge count 10.

Fig. 2.(b) shows the circuit with node R in Fig. 2.(a) being duplicated. The replication results in a cut with crossing edge count equal to 2. Before retiming, the clock period of the partition in Fig. 2(b) equals to 3. By shifting the registers to the new locations as indicated by dash lines, we can achieve a clock period 2.

32nd ACM/IEEE Design Automation Conference ®

2.2 Definitions

Data Flow Graph: We use a directed data flow graph $G = (V, E)$ to represent a circuit, where V and E model the combinational elements and the interconnections, respectively. Each node i is associated with a size s_i. By distributing the combinational delay of node i into the output edges of i, each edges (i, j) have a delay time d_{ij}. For a node with no output edges, the delay is placed on its input edges. Each edge (i, j) has a register count r_{ij} which denotes the number of registers on the interconnections from combinational element i to j. Furthermore, each edge (i, j) is associated with a capacity c_{ij} representing the number of interconnections from node i to j.

In this paper, we focus on two-way partitioning. We use function $size(X)$ to denote the total size of a node set X. The upper size limits of two partitions are denoted by Z_1 and Z_2, respectively.

Partitioning with replication is strongly related to the directions of edges. For two disjoint node sets X and Y, we use $[X \rightarrow Y]$ to denote the directed cut set from X to Y. Therefore, $[X \rightarrow Y]$ contains all the cut edges (i, j) such that $i \in X$ and $j \in Y$. We use function $c([X \rightarrow Y])$ to denote the total capacities of the edges in $[X \rightarrow Y]$.

Retiming: Given a data flow graph $G = (V, E)$, a retiming [6] specifies a transformation of the original graph in which registers are added and removed. Lerserson *et al.* [6] propose an $O(nm \log n)$ algorithm to determine the minimum clock period achieved by adopting retiming, where n and m denote the numbers of nodes and edges, respectively.

Iteration Bound: While retiming can reduce the clock period of a circuit, there is a lower bound imposed by the feedback loops in the data flow [11]. Given a feedback loop l, let d_l and r_l be the sum of edge delays and the sum of registers on loop l, respectively. The delay-to-register ratio of l is equal to $\frac{d_l}{r_l}$. The *iteration bound* J is defined as the maximum delay-to-register ratio, i.e.,

$$J = \max\ (\frac{d_l}{r_l}) \qquad \forall \text{ loop } l\ .$$

Note that if loop ℓ contains cut edges in a partition, the total edge delay has to include the cut delays.

Cycle Mean Problem: For the special case that each edge has exactly one register on it, the iteration bound problem becomes a cycle mean problem [1]. In our test cases, each combinational block node connects between two register nodes. We can identify the iteration bound and the edges that contribute to the bound in $O(nm)$ time.

Timing Constraint: We use the iteration bound constraint as the timing constraint in this paper. We have two reasons to use the iteration bound. (i) It is faster to calculate the iteration bound. (ii) The iteration bound stands for the lower bound of the clock period can be achieved by retiming.

2.3 Problem Formulation

Given a data flow graph $G = (V, E)$ and three node sets S, R and T such that $S \cap T = \emptyset$ and $R = V - S - T$ as shown in Fig. 2(a), Fig. 2(b) presents a partitioning with R being the set of replicated nodes. Each copy of R needs to collect a complete set of input signals.

Given two disjoint sets S and T, let a *replication cut* $[S, T]$ denote the cut set of a partitioning with $R = V - S - T$ being duplicated. From Fig. 2(b), we can see that *replication cut* $[S, T]$ is the union of four directed cuts,

$$[S, T] = [S \rightarrow T] \cup [T \rightarrow S] \cup [S \rightarrow R] \cup [T \rightarrow R].$$

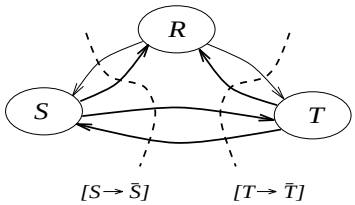

Figure 3: *Two directed cuts*

Interpretation of the Cut Set: Suppose we rewrite the replication cut in the format:

$$[S, T] = [S \rightarrow \bar{S}] \cup [T \rightarrow \bar{T}]$$

where $\bar{S}$ and $\bar{T}$ denote the complementary sets of S and T, respectively. We can interpret the cut set of the replication cut $[S, T]$ as two directed cuts on the original graph G as shown in Fig. 3.

Time-Constrained Replication Cut Problem: *Given a directed graph G and a ratio K, find a replication cut $[S, T]$ with an objective*

$$\min c([S, T]) = c([S \rightarrow \bar{S}] \cup [T \rightarrow \bar{T}])$$

subject to the size constraints that

$$size(S \cup R) \leq Z_1,\ and\ size(T \cup R) \leq Z_2,$$

and the feasible constraints

$$S \cap T = \emptyset, R = V - S - T$$

and the timing constraint

$$iteration\ bound\ of\ [S, T]\ \leq\ K.$$

3 Investigation of the Problem

In this section, we first show that the time-constrained replication cut problem without size constraint is $\mathcal{NP}$-Hard. Then, the edge dependency problem caused by replication in retiming is illustrated.

By reducing from **3SAT**, we have the following theorem [7]:

Theorem 1: *The time-constrained replication cut problem without size constraint is $\mathcal{NP}$-Hard.*

Edge Dependency of Replication in Retiming: Due to the replication, the loops which entirely locate in the set

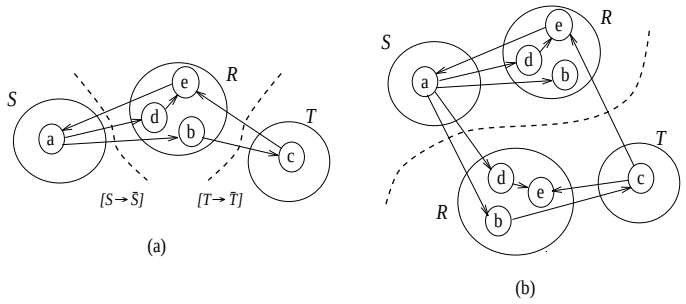

Figure 4: *(a) Replication cut $[S,T]$ (b) Duplicated graph G^d*

of nodes $S \cup R$ or $T \cup R$ will not be cut. Only those loops which intersect both node sets S and T will be cut. We use Fig. 4 to explain this concept. Given a graph G and a replication cut $[S,T]$ as shown in Fig. 4(a), graph G^d in Fig. 4(b) represents the graph with node set R duplicated. Due to the replication, directed edge (e,a) in G^d emanates from R to S on the same side of the partition and thus does not cross the cut. Similarly, directed edge (b,c) in G^d emanates from R to T on the same side of the partition and thus does not cross the cut.

The loop (a,d,e,a) in G entirely locate in the set of nodes $S \cup R$. So the corresponding loop (a,d,e,a) in G^d will not be cut. On the other hand, the corresponding loop $\ell = (a,b,c,e,a)$ in G^d of the loop (a,b,c,e,a) in G will be cut as shown in Fig. 4(b). Therefore, edges (a,b) and (c,e) contribute cut delay δ to loop ℓ in G^d with respect to replication cut $[S,T]$. These two cut delays result in loop ℓ with greater delay-to-register ratio for retiming. Thus, given an edge, the edge delay contributed by the replication cut depends on the configuration of the loop which contains the edge. This is called the *edge dependency problem*, which is caused by the composite effect of replication and retiming.

4 Construction of the Replication Graph

We devise a replication graph to tackle the edge dependency problem. We assume two nodes s and t are to be separated by the replication cut.

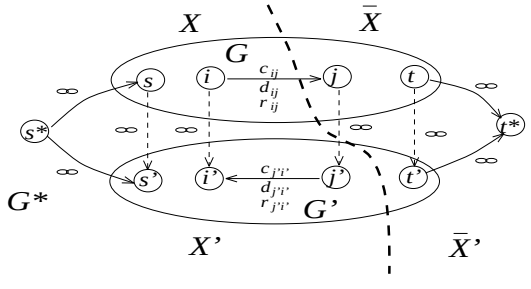

Figure 5: *The replication graph G^**

Construction of Replication Graph: Given the graph $G = (V,E)$ and a pair of nodes s and t, we create another graph $G' = (V',E')$ where each node i' corresponds to a node i in G and the directed edge (j',i') in G' is just the reverse of its corresponding edge (i,j) in G with equal capacity (i.e. $c_{j'i'} = c_{ij}$), equal delay time (i.e. $d_{j'i'} = d_{ij}$) and equal register count (i.e. $r_{j'i'} = r_{ij}$). From every node in G, we add a directed edge with infinite capacity, zero delay and zero register count to the corresponding node in G'. Then, we create a super source s^* and a super sink t^* which connect nodes s, s', t and t' with an infinite capacity, a delay time 0 and a register count 0. We refer to the combined graph as the *replication graph* $G^* = (V^*,E^*)$ shown in Fig. 5.

Given a directed cut $C = [\{s^*\} \cup X \cup X' \rightarrow \{t^*\} \cup \bar{X} \cup \bar{X}']$ of G^* with $V = X \cup \bar{X}$ and $V' = X' \cup \bar{X}'$, a replication cut $[S,T]$ of the original graph with $S = X$, $T = \{i \mid i' \in \bar{X}'\}$ and $R = V - S - T$ is derived. Note that T is derived from the cut in node set V'.

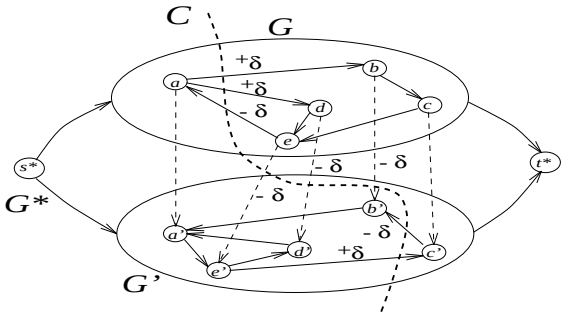

Figure 6: *Replication graph*

Solution of the Edge Dependency Problem: Given a graph G, the dependency of the edge delay can be solved on the replication graph G^* in the following way. Given a directed cut on the replication graph, we assign cut edges (forward edges) and backward edges in G^* with cut delay δ and $-\delta$, respectively. Fig. 6 shows the replication graph of the graph in Fig. 4.(a). In Fig. 6, (a,b), (a,d) and (e',c') are cut edges with cut delay δ; (e,a), (c',b'), (b,b'), (d,d') and (e,e') are backward edges with cut delay $-\delta$. Thus, the total cut delay contributing to loop (a,d,e,a) is zero (Fig. 6). On the other hand, path (a,b,c) and path (a',e',c') are cut on edges (a,b) and (e',c'). The two cuts contribute two cut delay 2δ to loop (a,b,c,e,a) in the duplicated graph (Fig. 4.(b)). Note that G' is the reverse graph of G. Thus, path (a,b,c) and path (a',e',c') correspond a loop (a,b,c,e,a) in the original graph.

5 Heuristic Algorithm

In this section, we incorporate the size constraint and release the requirement that two specified nodes s and t need to be separated.

Since the replication graph can solve the edge dependency problem, we can formulate the time-constrained replication cut as an integer linear programming formulation by utilizing the proposed replication graph. The relaxation of integer constraint derives a lower bound solution. The duality of the relaxed programming problem motivates the concept of penalty function of edge capacities which is the base of the proposed heuristic algorithm of this section. The detail derivation appears in [7].

TPRG

Initially, $k = 1$; *Iteration* =CONSTANT;
Step1. Build replication graph G^*.
Step2. Find directed cut C^k of G^* by directed FM.
Step3. If (C^k violates the timing constraint)
then reflect penalty on edge capacities
else stop
Step4. Increase k by 1
If ($k >$ *Iteration*)
then stop
else goto Step2

Figure 7: *Outline of algorithm*

Fig. 7 show that the outline of the heuristic algorithm Time-Constrained Partitioning using Replication Graph (TPRG). In each iteration, the algorithm identifies the loop which violates the timing constraint and adds penalty to the edges of the loop. The penalty function of edge capacities enforces the following directed cut to avoid cutting through the violating loops. The iteration stops if a feasible solution is found or the number of iterations is beyond the limit.

5.1 Revision of the Replication Graph

We release the constraint of two special nodes s and t in G. Therefore, there is no need to create s^* and t^* and their connecting edges in G^*.

5.2 Directed Cut using FM Method

We use a directed FM [4] to get a directed cut of G^*. However, instead of applying a directed FM method to the original graph as in [4], our approach applies the directed FM to the proposed replication graph to minimize the replication cut cost.

5.3 Penalty Function of Edge Capacities

Let $C^k = [X \cup X' \rightarrow \bar{X} \cup \bar{X}']$ denote the directed cut we found. Based on cut C^k, the cut edges and backward edges are assigned cut delay δ and $-\delta$, respectively. Then, we use the mean cycle algorithm [1] to calculate the iteration bound h^k of G^*. In the mean cycle algorithm, we can also compute a value h^k_{uv} for each edge (u, v) in G^*, where h^k_{uv} represents the maximum delay-to-register ratio of all loops passing edge (u, v) in G^* with respect to cut C^k. If h^k is not greater than K, the algorithm stops. Otherwise, the capacity of each edges (u, v) in G^* with $h^k_{uv} > K$ is updated as follows:

$$c^{k+1}_{uv} = max\ \{0,\ c^k_{uv} + \frac{h^k_{uv} - K}{\sum_{(i,j)\in E^*\ and\ h^k_{ij}>K}(h^k_{ij} - K)}\Delta\},$$

where Δ is a given constant. By updating the capacities, edges located on the most critical loops (i.e., their delay-to-register ratio violate the timing constraint most) will get largest penalties. In our program, we set Δ to be 100.

6 Experimental Results

We use the same seven industrial circuits from [13] as our test cases. In this experiment, each module has size constraint equal to 60% of the circuit size. Therefore, the size constraint restricts replication to 20% of the total size.

We compare our algorithm TPRG to the Fiduccia-Mattheyses (FM) [3] algorithm, and LAMP [10]. All programs are run on a single-processor SUN SPARC 10 workstation. The results of FM are chosen from the best of 20 runs each.

Table 1 shows the characteristics of the test cases. The fourth and fifth columns stands for the *iteration bound* J and the *path delay bound* B which is defined later with the *external-loop constraint*. Since $s2$ and $s6$ do not have feedback loops, the iteration bounds are equal to zero. The sixth and seventh columns list the crossing edge count of FM and LAMP, respectively. As indicated in [2], the intermodule delay δ can increase to nearly 100% of the clock cycle period. Therefore, we set δ to be of 60% of $\max(J, B)$, which is calculated before partitioning in this paper.

cir.	#reg.	#comb.	J	B	FM cut	LAMP cut
s1	342	8280	6373	5447	2860	3134
s2	472	3378	0	4421	875	847
s3	521	6325	2527	3238	1422	1629
s4	380	3850	4922	5545	1045	1032
s5	545	12172	4241	4876	3465	3478
s6	357	3026	0	3724	848	817
s7	607	4990	996	3563	1103	1141

Table 1: Characteristics of test cases.

External-Loop Constraint: A system can interact with external systems. Hence, macroscopically, there possibly exist external feedback loops from the primary outputs to the primary inputs. We call this assumption the *external-loop constraint*. According to the external-loop constraint, we have to take into account the path delay. Given a path p from the primary input pad to the primary output pad, let d_p and r_p be the sum of delay time and the sum of registers on path p. The *path delay bound* of a circuit is defined by:

$$B = \max \frac{d_p}{r_p} \qquad \forall \text{ IO-path } p$$

In the following, we present the experimental results with the external-loop constraint. The experimental results without the external-loop constraint appears in [7].

6.1 Experiment with External-Loop Constraint

According to the external-loop constraint, we have to take into account the path delay. Then the dominant delay of a given partitioned circuit is: $A = \max(J, B)$.

Table 2 gives our experiments. The timing constraint is the the same as that in [10]. The data in the first subcolumn are the value of A derived from above equation after partitioning. The T in the second subcolumn is the clock cycle period of the partitioned circuit after retiming.

The reductions on the cut edge counts are as follows. When compared to the FM, TPRG achieved $1.92 \sim 72.89\%$ with an average of 22.15%. TPRG achieved $0.61 \sim 73.79\%$ with an average of 23.79%, compared with LAMP. Especially, TPRG improve the cut edge count by 73.79% for $s7$, compared with LAMP. Due to $s7$ with many nodes having large fan-out and small fan-in, the cut edge count is dramatically reduced by replication. When compared to the FM and LAMP, the clock cycle period reductions are as follows. TPRG achieved $22.80 \sim 34.43\%$ with an average of 27.60% for A and $13.04 \sim 34.42\%$ with an average of 26.25% for T, compared with FM. Even given the same timing constraint, TPRG still get improvement, compared with LAMP. TPRG achieved $0 \sim 12.91\%$ with an average of 3.43% for A and $0 \sim 13.78\%$ with an average of 3.97% for T. The number of iterations of TPRG for these seven test cases are 6, 2, 1, 9, 1, 7 and 1, respectively. The size overheads are 19.96%, 0.95%, 13.82%, 19.66%, 20.00%, 19.93% and 12.48%, respectively.

cir.	FM		LAMP		TPRG		
	A	T	A	T	A	T	cut
s1	8371	9238	6373	6653	6373	6587	2805
s2	7074	7215	5206	5310	5080	5130	723
s3	5338	5444	4019	4099	3500	3534	1018
s4	8239	8631	6360	7505	6360	7505	905
s5	7666	8432	6366	6493	5824	5882	2881
s6	6544	6544	4502	5042	4502	5042	812
s7	5103	5227	3644	3935	3637	3891	299

Table 2: The experiment with external-loop constraint

6.2 Experiment with Tighter Constraint

By using binary search, we can determine the minimum dominate delay A achieved by our algorithm. Given a ratio K, if our algorithm cannot find a partition with dominate delay A no greater than K within 20 iterations, it fails; otherwise it success and we can go ahead to tighter timing constraints.

Table 3 show the experimental results. *#iter.* denotes how many iterations TPRG takes to terminate and *over.* denotes the percentage of the size overhead of duplicated nodes. 10.08% improvement on the clock period and 20.65% reduction on the cut edge count are achieved, compared with LAMP approach. Compared with FM approach, TPRG can achieved 33.33% improvement on the clock period and 18.8% reduction on the cut edge counts.

7 Conclusive Remarks

In this paper, a replication graph is adopted to to tackle the performance-driven partitioning problem using retiming and replication.

cir.	K	cut	A	T	#iter.	over.
s1	6373	2805	6373	6587	6	19.96
s2	4700	785	4569	4614	8	2.74
s3	3300	1172	3300	3325	13	19.75
s4	6360	905	6360	7505	9	19.66
s5	5500	2879	5476	5530	16	19.97
s6	4300	857	4038	4318	14	19.95
s7	3565	296	3563	3603	7	16.78

Table 3: Results with tighter constraint

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] D. A. Doane and P. D. Franzon *ed.*, *Multichip Module Technologies and Alternatives –The Basics"*, Van Nostrand Reinhold, New York, 1993, pp. 666 - 667

[3] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proc. 19th ACM/IEEE Design Automation Conference*, 1982, pp. 175 - 181.

[4] J. Hwang and A. E. Gamal, "Optimal Replication for Min-Cut Partitioning", *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, Nov., 1992, pp. 432 - 435.

[5] C. Kring and A. R. Newton, "A Cell-Replicating Approach to Mincut Based Circuit Partitioning," *Proc. IEEE Int. Conf. on Computer-Aided Design*, 1991, pp 2 - 5.

[6] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica,* Vol. 6, No. 1, 1991, pp. 5 - 35.

[7] L.T. Liu, "Performance-Driven Partitioning Using Retiming and Replication," Ph.D dissertation, CS94-379, Dept. of Computer Science and Engineering, UC San Diego, 26 June, 1994

[8] L.T. Liu, M. T. Kuo, C.K. Cheng, and T. C. Hu, "A Replication Cut for Two-Way Partitioning," *to appear in IEEE Trans. on Computer Aided Design*

[9] L.T. Liu, M. Shih, N.C. Chou, C.K. Cheng, and W. Ku, "Performance-Driven Partitioning Using Retiming and Replication," *Proc. IEEE International Conference on Computer-Aided Design,* Santa Clara, Nov. 1993, pp. 296-299

[10] L. T. Liu, M. Shih, and C. K. Cheng, "Data Flow Partitioning for Clock Period and Latency Minimization," *Proc. 31th ACM/IEEE Design Automation Conference*, San Diego, Jun. 1994, pp. 658-663.

[11] K. K. Parhi and D. G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding," *IEEE Trans. on Computers,* Vol. 40, No. 2, 1991, pp. 178-195.

[12] M. Shih, E. S. Kuh, and R.-S. Tsay, "Performance-Driven System Partitioning on Multi-Chip Modules," *Proc. 29th ACM/IEEE Design Automation Conf.,* 1992, pp. 53 - 56.

[13] M. Shih and E. S. Kuh "Quadratic Boolean Programming for Performance-Driven System Partitioning," *Proc. 30th ACM/IEEE Design Automation Conf.,* 1993, pp. 761-765.