

# Data Flow Partitioning with Clock Period and Latency Constraints

Lung-Tien Liu, Minshine Shih, John Lillis, and Chung-Kuan Cheng, *Senior Member, IEEE*

**Abstract**— We propose an efficient performance-driven two-way partitioning algorithm to take into account clock period and latency with retiming. We model the problem with a Quadratic Programming formulation to minimize the crossing edge count with nonlinear timing constraints. By using a *Lagrangian Approach on Modular Partitioning (LAMP)*, we merge nonlinear constraints into the objective function. We then decompose the problem into primal and dual subprograms. The primal program is solved by a heuristic Quadratic Boolean Programming approach and the dual program is solved by a subgradient method using a cycle mean method. Experimental results on seven industrial circuits have demonstrated our algorithm is able to achieve an average of 23.25% clock period and latency reductions compared to the best results produced by 20 runs on each test case using a Fiduccia-Mattheyses algorithm. In terms of the average number of crossing edges, our results are only 1.85% more than those of the Fiduccia-Mattheyses algorithm without timing constraints. Compared with previous network flow based approach, our algorithm reduces the average crossing edge count by 14.59%. Furthermore, an average of 7.70% clock period and latency reductions are achieved.

## I. INTRODUCTION

**D**UE TO PHYSICAL GEOMETRIC distance and interface technology limitations, intermodule delay is contributing a dominant portion of signal propagation delay. Consequently, instead of minimizing the number of the crossing edges [4], [8], [11], [14], [23], [27] as the only objective during partitioning, we should take into account the intermodule delay for performance-driven partitioning.

Clock period is a major measurement for circuit performance. For a partitioning problem with timing and size constraints, Shih *et al.* [25] first proposed a  $K$ -way Kernighan-Lin (KL) [14] based algorithm. Later, Shih and Kuh [26] formulated the partitioning problem with timing and size constraints as the Quadratic Boolean Programming problem (QBP) with linear constraints. Their approaches can generate a partition such that the delay between registers satisfies the timing constraint. In terms of the crossing edge count, their algorithms consistently produce results comparable with Kernighan-Lin and Fiduccia-Mattheyses (FM) [8] based algorithms without timing constraints.

Manuscript received January 4, 1995; revised July 17, 1996. This paper was recommended by Associate Editor G. Gielen.

L.-T. Liu is with AT&T Bell Laboratories, Murray Hill, NJ 07974 USA.

M. Shih is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA.

J. Lillis and C.-K. Cheng are with the Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 USA.

Publisher Item Identifier S 1057-7122(97)02072-2.

In [20], Liu *et al.* proposed a flow-based partitioning algorithm using a *retiming* technique [18], [19] to explore the ultimate clock period of the circuit. The algorithm ensures that any critical path is cut at most once. They achieve clock period reduction at the expense of crossing edge count. The partitioning results in 16.44% more crossing edges [20] than the FM algorithm [8] without timing constraints.

In addition to clock period, the primary input to primary output latency is another important measurement for circuit performance. In this paper, our performance-driven partitioning method extends the problem of [20] to incorporate system latency. In other words, we minimize the crossing edge count of the partition subject to the performance constraints that the clock period and latency achieved by using retiming are within the given limits.

Given the size and performance (timing) constraints, the partitioning problem using retiming is formulated as QBP with linear constraints on the size limit and nonlinear constraints on the clock period and latency limits. By using a *Lagrangian relaxation*, the nonlinear constraints are merged into the objective function as a Lagrangian problem. The Lagrangian problem is then decomposed into primal and dual subprograms.

The performance-driven partitioning problem is solved through the primal and dual iterations on the Lagrangian problem. The primal program is a traditional partitioning problem without timing constraints and is solved by a heuristic Quadratic Boolean Programming approach [2], [26]. The dual program is solved by a subgradient method [24] using the cycle mean method [13]. According to the experimental results, the proposed partitioning algorithm can simultaneously enhance the clock period and system latency considerably. Moreover, the crossing edge counts of our results are comparable with those of the FM algorithm without timing constraints.

The remainder of this paper is organized as follows. The concepts of retiming and latency are described and the performance driven partitioning problem is formulated in Section II. We present a quadratic programming formulation of the problem in Section III. Section IV presents the algorithm. Experimental results and concluding remarks are given in Sections V and VI.

## II. PROBLEM FORMULATION

In this section, we first give some basic definitions. Then, the performance-driven partitioning problem is formally defined.

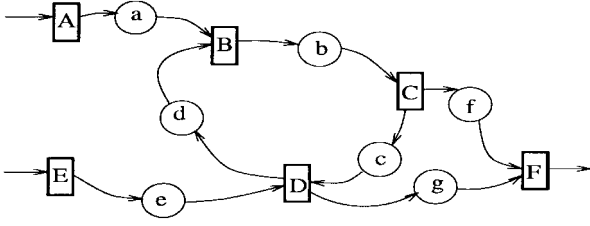


Fig. 1. An example of data flow graph, where  $A$  and  $E$  are primary inputs and  $F$  is a primary output.

### A. Data Flow Graph

A synchronous digital circuit may consist of combinational elements and globally clocked registers. Each combinational element has positive delay and size associated with it, while all registers have delay and size values equal to zero.

A *directed data flow graph*  $G = (V, E)$ , where  $V = R \cup C$ , is adopted to represent the synchronous digital circuit, where  $R$  and  $C$  are sets of register nodes and combinational block nodes, respectively, and  $E$  is a set of directed edges which correspond to signal flow in the system. Each node  $i$  has an associated size  $s_i$  and delay  $d_i$ . As in [20], we assume that the combinational blocks are fine-grained. A node is called fine-grained, if it can be split into several smaller nodes. On the other hand, if a node can not be split, it is called coarse-grained. An attribute  $c_{ij}$  denoting the number of interconnections between nodes  $i$  and  $j$  is associated with each edge  $(i, j)$ . Furthermore, each edge is of *zero delay*. Fig. 1 is an example of such a graph. In Fig. 1, registers are represented by rectangles and combinational blocks are represented by circles.  $A$  and  $E$  are primary inputs and  $F$  is a primary output.

A path of length  $k$  from a node  $i$  to a node  $j$  in a graph  $G = (V, E)$  is a sequence  $\langle i_0, i_1, \dots, i_k \rangle$  of nodes such that  $i = i_0$ ,  $j = i_k$ , and  $(i_{l-1}, i_l) \in E$  for  $l = 1, 2, \dots, k$ . A path is simple, if all nodes on the path are distinct. A path  $\langle i_0, i_1, \dots, i_k \rangle$  forms a loop if  $i_0 = i_k$  and the path contains at least one edge. The loop is simple if, in addition,  $i_1, \dots, i_k$  are distinct. In the remainder of the paper, the terms path and loop are used to refer to simple paths and simple loops respectively.

A two-way partition  $P = (V_1, V_2)$  maps  $V$  into two modules, such that  $V_1 \cup V_2 = V$  and  $V_1 \cap V_2 = \emptyset$ . The upper size limits of these modules are denoted by  $S_1$  and  $S_2$ , respectively. An edge  $(i, j)$  is a *crossing edge* of  $P$  if node  $i$  and node  $j$  are in different subsets  $V_1$  and  $V_2$ . In this paper, the term “cut” stands for the crossing edge count. Each crossing edge is associated with an intermodule delay  $\delta$  determined by the technology. By contrast, noncrossing edges incur zero delay. Furthermore, the intermodule delay on crossing edges is inherently coarse-grained and cannot be split.

### B. System Latency and Retiming

In this subsection, we first introduce system latency and retiming. Then, a theorem formalizing the conditions under which the required clock period and system latency can be achieved for a given circuit via retiming is presented.

Given a path  $p$ , we use  $r_p$  to denote the number of registers on the path. Let  $W(i, j)$  denote the minimum number of

registers among all possible paths  $p$  from  $i$  to  $j$ , i.e.,

$$W(i, j) = \min\{r_p \mid p \in P\}$$

where  $P$  is the set of all paths from node  $i$  to  $j$ . We define a path  $p$  from  $i$  to  $j$  as a *critical path* if  $r_p$  equals  $W(i, j)$ ;  $p$  is also called an *IO-critical path* if node  $i$  and  $j$  are the primary input and output nodes, respectively. Let  $d_p$  and  $\hat{d}_p$  be the sum of combinational block delays and the sum of intermodule delays on path  $p$ , respectively. Let

$$D(i, j) = \max\{d_p + \hat{d}_p \mid p \in P_C\} \quad (1)$$

where  $P_C$  is the set of all critical paths from node  $i$  to  $j$  and  $D(i, j)$  denotes the maximum delay over all critical paths from node  $i$  to  $j$ .

*System Latency*: Given a path  $p$  from primary input  $i$  to primary output  $j$ , we define the IO latency between  $i$  and  $j$  to be the minimum number of clock periods that pass between the signal arrival at node  $i$  and its first effect on the output signal of node  $j$ , as  $(W(i, j) - 1) * T$ , where  $T$  denotes the clock period. Note that the primary inputs and outputs are register nodes. In other words, the signal at primary input  $i$  takes  $W(i, j) - 1$  clock periods to first change the output signal of the primary output node  $j$ . However, if there is no path between  $i$  and  $j$ , we set its latency to zero. Thus, we define *system latency*  $N$  of the whole system as the maximum IO latency among all possible IO pairs [17], [9], [22], i.e.,

$$N = \max\{W(i, j) - 1\} * T \mid i \in I, j \in O \quad (2)$$

where  $I$  and  $O$  are the sets of all primary inputs and outputs, respectively.

We use the example in Fig. 1 to illustrate the essence of latency. It takes at least three registers,  $B$ ,  $C$ , and  $F$  to walk from  $A$  to  $F$ . Thus, the IO latency between  $A$  and  $F$  is three clock periods. There are two simple paths from  $E$  to  $F$ . One has three registers and the other has five. The IO latency between  $E$  and  $F$  is equal to two clock periods. The system latency of the circuit is dominated by the IO latency between  $A$  and  $F$ , the value of which is three clock periods.

*Retiming*: Given a data flow graph  $G = (V, E)$  where  $V = R \cup C$ , let  $I_R \subseteq R$  and  $O_R \subseteq R$  be the set of primary inputs and outputs, respectively. A retiming of a data flow graph  $G$  is an integer labeling of combinational, primary input, and primary output nodes:  $\Pi: C \cup I_R \cup O_R \rightarrow \mathbb{Z}$ . The retiming specifies a transformation of the original graph in which registers, except primary inputs and primary outputs, are moved across combinational blocks (and IO registers in our case) so as to change the graph  $G$  into a new graph  $G^\Pi = (V^\Pi, E^\Pi)$  where  $V^\Pi = R^\Pi \cup C$ . Let  $r_p^\Pi$  denote the number of registers on path  $p$  from  $i$  to  $j$  after retiming  $\Pi$ . According to [19], we have (3).

$$r_p^\Pi = r_p + \Pi(j) - \Pi(i). \quad (3)$$

If  $\Pi(j) - \Pi(i)$  is positive then the register count of each path from  $i$  to  $j$  is increased by the amount  $\Pi(j) - \Pi(i)$  by retiming. Otherwise,  $\Pi(i) - \Pi(j)$  registers are deleted. Furthermore, we have Theorem 1 (derived from [19, Theorem 7]):

*Theorem 1:* Let  $G = (V, E)$  be a data-flow graph, where  $V = R \cup C$  and let  $K$  be an arbitrary positive integer. Further, let  $\Pi$  be an integer function.  $\Pi$  is a legal retiming of  $G$  achieving a clock period of  $K$  iff:

- i)  $\Pi(i) - \Pi(j) \leq W(i, j)$  for any two nodes  $i$  and  $j \in C \cup I_R \cup O_R$ .
- ii)  $\Pi(i) - \Pi(j) \leq W(i, j) - 1$  for any two nodes  $i$  and  $j \in C \cup I_R \cup O_R$  such that  $D(i, j) > K$ .

Statement i) expresses that the register count between nodes  $i$  and  $j$  cannot be negative. Statement ii) shows that at least one register must be kept on the critical path from  $i$  to  $j$  if it has a delay larger than  $K$ . Once statement ii) holds, [19] states that every path from  $i$  to  $j$  with delay larger than  $K$  will have at least one register.

If i) and ii) hold, the retiming  $\Pi$  achieves clock period  $K$ . By (3), the register count of each path from  $i$  to  $j$  is augmented by an equal amount  $\Pi(j) - \Pi(i)$  through retiming. Therefore, if a path  $p$  from  $i$  to  $j$  is critical before retiming,  $p$  is still critical after retiming.

In order to achieve a system latency of  $H$  clock periods, we incorporate the following condition:

- iii)  $\Pi(j) - \Pi(i) \leq H - W(i, j)$  for any  $i \in I_R$  and any  $j \in O_R$ .

Statement iii) enforces that each IO-critical path has no more than  $H$  registers. Thus,  $\Pi$  can achieve a system latency of  $H$  clock periods. Therefore, if  $\Pi$  satisfies the conditions i), ii), and iii),  $\Pi$  is a legal retiming of  $G$ , achieving a clock period of  $K$  and a system latency of  $H$  clock periods.

*Time Complexity:* We first present an algorithm to determine if a circuit achieves a clock period of  $K$  and a system latency of  $H$  clock periods by using retiming. From the statements in the preceding paragraph, the problem of finding a retiming which can achieve a required clock period and system latency is equivalent to finding a function  $\Pi$  from  $C \cup I_R \cup O_R$  to integers which satisfies all constraints i), ii), and iii). In other words, we need to determine the feasible values for all the unknowns  $\Pi(i)$  under a set of inequality constraints with the form of  $\Pi(i) - \Pi(j) \leq u_{i,j}$ , where  $u_{i,j}$  is a constant. Constraint systems with such format arise in shortest paths problems. We can use the Bellman-Ford algorithm (see [5, p. 532]) to determine if there exists a  $\Pi$  which satisfies all the constraints. Given a graph  $G = (V, E)$ , the time complexity of the Bellman-Ford algorithm is  $O(|V||E|)$ .

Given a data flow graph  $G = (V, E)$  where  $V = R \cup C$ , let  $n = |C|$ , and  $d$  and  $d_{\text{unit}}$  denote the maximum and minimum delay value among all combinational block delays, respectively. We show that finding a retiming which can achieve a required clock period and system latency (or determining that no such retiming exists) can be done in  $O((n \frac{d}{d_{\text{unit}}} + |R|)^3)$  time. Before enumerating all constraints of i), ii), and iii), the values of all  $W(i, j)$  and  $D(i, j)$  for each pair of nodes  $i$  and  $j$  in  $C \cup I_R \cup O_R$  are computed. The Floyd-Warshall all-pairs shortest-paths algorithm (see [5, p. 558]) can be adopted to compute the values of all  $W(i, j)$  and  $D(i, j)$ . Given a graph, the Floyd-Warshall algorithm takes  $O(|V|^3)$  time, where  $V$  denotes the set of nodes of the given graph. The reader can see the **Algorithm WD** in [19, Section

4] for greater detail. Since the combinational blocks are fine grained, a combinational block node  $i$  with delay  $d_i$  is split into  $\frac{d_i}{d_{\text{unit}}}$  combinational block nodes each having unit delay. Given a data flow graph  $G = (V, E)$ , a new graph  $G' = (V', E')$  is constructed, where each combinational block in  $V'$  is of unit delay and  $E'$  denotes the extended set of edges from  $E$ . Therefore,  $|V'|$  is given by  $n \frac{d}{d_{\text{unit}}} + |R|$ . Given the graph  $G' = (V', E')$ , it takes  $O((n \frac{d}{d_{\text{unit}}} + |R|)^3)$  time to calculate the values of all  $D(i, j)$ . After all  $D(i, j)$  have been calculated, we can enumerate all constraints of i), ii), and iii). Then, the Bellman-Ford algorithm is applied to determine if retiming can achieve the target clock period and system latency. Since the Bellman-Ford algorithm takes  $O((n \frac{d}{d_{\text{unit}}} + |R|)|E'|)$  time, the time complexity of the algorithm for finding a feasible retiming is dominated by the time complexity of calculating the values of all  $D(i, j)$ , which is  $O((n \frac{d}{d_{\text{unit}}} + |R|)^3)$ . Note that the retiming algorithm in [19] satisfies only conditions i) and ii). Therefore, the time complexity of the algorithm in [19] is different from ours.

### C. Iteration Bound and Latency Bound

*Iteration Bound:* While retiming can reduce the clock period of a circuit, there is a limit to the amount of improvement possible imposed by feedback loops in the circuit [21]. Given a loop  $l$ , let  $d_l$ ,  $\hat{d}_l$ , and  $r_l$  be the sum of combinational block delays, the sum of intermodule delays, and the number of registers in loop  $l$ , respectively. The delay-to-register ratio of a loop  $l$  is equal to  $\frac{d_l + \hat{d}_l}{r_l}$ . The *iteration bound* of a circuit is defined as the maximum delay-to-register ratio, i.e.,

$$J = \max \left\{ \frac{d_l + \hat{d}_l}{r_l} \mid l \in L \right\} \quad (4)$$

where  $L$  is the set of all loops in the circuit. Note that the iteration bound of a given circuit yields a lower bound on the achieved clock period by retiming.

*Minimum Cycle Mean Problem:* For the special case where each edge contributes exactly one register, the iteration bound becomes a minimum cycle mean [13]. In our test cases, each combinational block node connects two register nodes. Thus, we can adopt the minimum cycle mean algorithm proposed by Karp [13] to calculate the iteration bound and the edges that contribute to the bound. Given a graph, the time complexity of Karp's algorithm is  $O(nm)$ , where  $n$  and  $m$  denote the number of nodes and edges, respectively.

*Latency Bound:* Let  $p$  denote the IO-critical path with maximum path delay among all IO-critical paths from  $i$  to  $j$ . Since the number of registers on path  $p$  is equal to  $W(i, j)$ , the IO latency (i.e.,  $(W(i, j) - 1) * T$ ) between  $i$  and  $j$  is not less than  $d_p + \hat{d}_p$ , where  $T$  denotes the clock period, and  $d_p$  and  $\hat{d}_p$  are the sum of combinational block delays and the sum of intermodule delays on path  $p$ , respectively. Thus, we define *latency bound*  $M$  as follows:

$$M = \max \{ d_p + \hat{d}_p \mid p \in P_{\text{IOC}} \} \quad (5)$$

where  $P_{\text{IOC}}$  is the set of all IO-critical paths. The latency bound also imposes a lower bound on the system latency

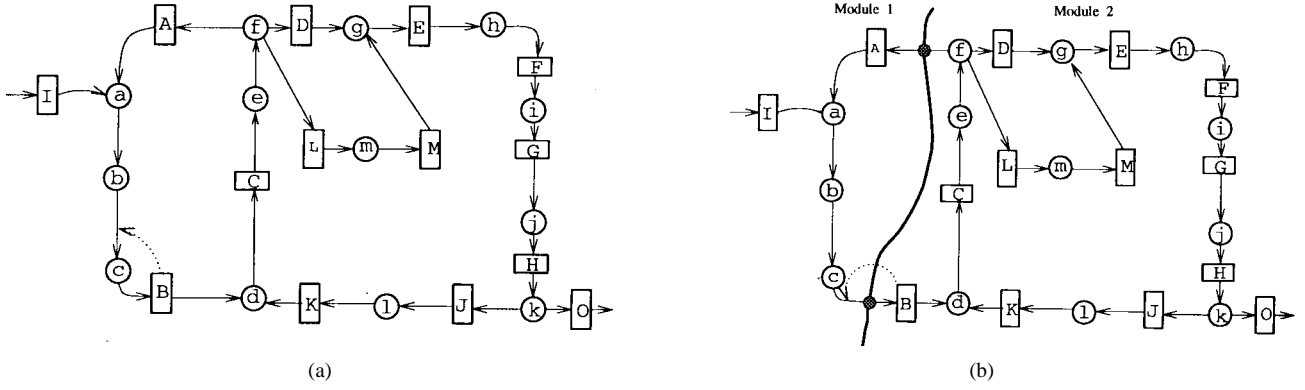


Fig. 2. Effect of retiming. (a) Retiming reduces clock period from three units of original circuit to two units and system latency from 24 units to 16 units. (b) Retiming reduces clock period from five units of the partitioned circuit to four units and achieves a system latency of 32 units.

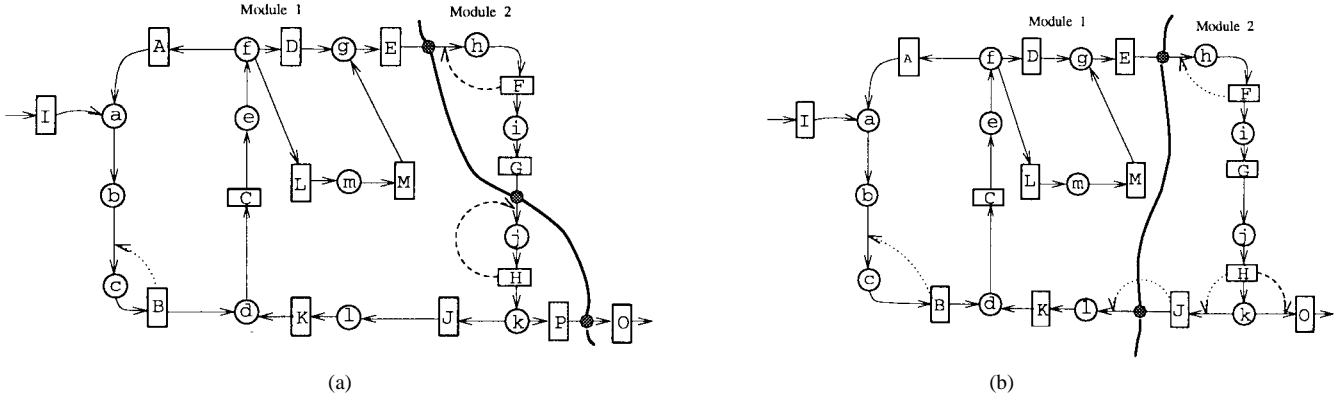


Fig. 3. Effect of partitioning. (a) A partition with two units clock period and 18 units latency. (b) A partition with two units clock period and 16 units latency.

achieved by using retiming. An all-pairs shortest-paths algorithm can be used to calculate the latency bound.

**Timing Constraint:** The timing constraints adopted by our algorithm are based on the limits imposed by the iteration and latency bounds. In other words, given two numbers  $K$  and  $H$ , our performance-driven partitioning algorithm will generate a partition with iteration and latency bounds not greater than  $K$  and  $H$ , respectively. We have two reasons for using the iteration and latency bounds, instead of actually performing the retiming during the process of the partitioning. i) It is faster to calculate these bounds. ii) The iteration and latency bounds give lower bounds on the clock period and system latency achievable by applying retiming. Furthermore, from our experience these bounds are very good predictors of system performance after retiming is applied, particularly in the fine-grained case. Therefore, we want to generate a partition with small iteration and latency bounds.

#### D. The Performance-Driven Partitioning Problem

Since the performance of a circuit is measured by both clock period and system latency, we want to generate a partition which achieves a good clock period and system latency by adopting retiming. In practice, we use the iteration and latency bound limits as the timing constraints. Retiming is then performed after partitioning to derive the exact clock period and system latency.

Given a partition  $P = (V_1, V_2)$ , let  $J(P)$  denote the iteration bound and  $M(P)$  denote the latency bound with respect to  $P$ . Now we state the performance-driven partitioning problem as follows.

Given a data flow graph  $G = (V, E)$ , where  $V = R \cup C$ , and each node  $i$  has size  $v_i$ , two numbers  $K$  and  $H$ , size constraints  $S_1$  and  $S_2$ , and intermodule delay  $\delta$ , find a partition  $P = (V_1, V_2)$  with the minimum number of crossing edges, subject to  $\sum_{i \in V_1} v_i \leq S_1$ ,  $\sum_{i \in V_2} v_i \leq S_2$ ,  $J(P) \leq K$ , and  $M(P) \leq H$ .

If  $K$  and  $H$  are large enough, the above problem becomes the traditional two-way partitioning problem of minimizing the number of the crossing edges while satisfying size constraints. Since our problem is in  $\mathcal{NP}$  and the traditional partitioning problem is  $\mathcal{NP}$ -Hard, the performance-driven partitioning problem is also  $\mathcal{NP}$ -Hard.

Examples in Figs. 2 and 3 illustrate the essence of the performance-driven partitioning problem. In Figs. 2 and 3, shaded octagons denote crossing edges. In these examples, we assume combinational block delays are one unit and intermodule delays  $\delta$  are two units. Register  $I$  and register  $O$  denote primary input and primary output, respectively.

Given the circuit in Fig. 2(a), the clock period is dominated by the longest combinational node delay between registers, which is from  $A$  to  $B$  with a delay of three units. There are two simple paths from nodes  $I$  to  $O$ . One path has nine registers, the other has 10 registers. Therefore, the system latency of

this circuit is eight clock periods, i.e., 24 units. However, using retiming, we can move register  $B$  to a new location as indicated by the dashed line. The resulting longest paths are from  $A$  to  $B$  and from  $B$  to  $C$ . Both paths have an improved delay of two units. Furthermore, the system latency becomes 16 units, substantially less than the original 24 unit. Because the iteration bound, which is determined by the left loop, is two units, we cannot obtain a smaller clock period.

Suppose the circuit is partitioned into two modules [Fig. 2(b)]. The clock period is five units before retiming because of the delay on the longest path from  $A$  to  $B$ . After retiming shifts  $B$  to its new location as indicated by the dashed line, the delay is four units and the system latency increases to 32 units. On the other hand, in Fig. 3(a), before retiming, the clock period is three units; hence, compared to Fig. 2(b), a better partition can automatically improved performance. If we perform retiming as shown by the dashed lines, the delay in Fig. 3(a) is reduced to two units. Retiming adds one extra register  $P$  between the combinational node  $k$  and register  $O$ . This results in a system latency of nine clock periods or 18 units. However, in Fig. 3(b), we also achieve a clock period of two units while achieving a system latency of only eight clock periods. Hence, Fig. 3(b) is preferred.

### III. QUADRATIC PROGRAMMING FORMULATION OF PERFORMANCE-DRIVEN PARTITIONING PROBLEM

In this section, we introduce a vector of Boolean variables to represent a partition. Given the timing and size constraints, the performance-driven partitioning problem can thus be represented by a Quadratic Boolean Programming formulation with linear constraints capturing the size limits and nonlinear constraints for the iteration and latency bounds. The nonlinear constraints are then absorbed into the objective function as a Lagrangian problem. Finally, the Lagrangian problem is decomposed into primal and dual subproblems.

#### A. Quadratic Boolean Programming Formulation

Let  $x_{b,i}$  denote a Boolean variable, where  $x_{b,i}$  is 1 if node  $i$  is assigned to module  $b$ , and  $x_{b,i}$  is otherwise 0. Therefore, given a data flow graph  $G = (V, E)$ , a two-way partition can be described by a vector of Boolean variables  $x = (x_{1,1}, \dots, x_{1,n}, x_{2,1}, \dots, x_{2,n})$ , where  $|V| = n$ .

Given a partition represented by a vector of Boolean variables, we express the delay-to-register ratio of a loop and the total delay of a path in terms of the given Boolean variables. If  $(i, j)$  is a crossing edge, the value of the term  $x_{1,i}x_{2,j} + x_{2,i}x_{1,j}$  is equal to 1; otherwise,  $x_{1,i}x_{2,j} + x_{2,i}x_{1,j}$  equals 0. Given a loop  $l$ , let  $g_l(x)$  denote the delay-to-register ratio of  $l$ .  $g_l(x)$  can be written as

$$g_l(x) = \frac{d_l + \sum_{(i,j) \in l} \delta(x_{1,i}x_{2,j} + x_{2,i}x_{1,j})}{r_l} \quad (6)$$

where  $d_l$ ,  $r_l$ , and  $\sum_{(i,j) \in l} \delta(x_{1,i}x_{2,j} + x_{2,i}x_{1,j})$  represent the sum of combinational block delays, the number of registers and the sum of intermodule delays in loop  $l$ , respectively. Similarly, the total delay  $h_p(x)$  of path  $p$  in terms of the given

Boolean variables is the following:

$$h_p(x) = d_p + \sum_{(i,j) \in p} \delta(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}) \quad (7)$$

where  $d_p$  and  $\sum_{(i,j) \in p} \delta(x_{1,i}x_{2,j} + x_{2,i}x_{1,j})$  denote the sum of combinational block delays and the sum of intermodule delays on path  $p$ , respectively.

Let us formulate the performance-driven partitioning problem by adopting a vector of Boolean variables to represent a partition. The objective function is to minimize the total costs of crossing edges

$$\min \sum_{(i,j) \in E} c_{i,j}(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}). \quad (8)$$

Subject to the following constraints:

C1: (Size Constraints)

$$\sum_{i=1}^n x_{b,i} s_i \leq S_b \quad \forall \text{ modules } b \in \{1, 2\}. \quad (9)$$

C2: (Generalized Upper Bound Constraints)

$$\sum_{b=1}^2 x_{b,i} = 1 \quad \forall \text{ nodes } i \in V. \quad (10)$$

C3: (Iteration Bound Constraints)

$$g_l(x) \leq K \quad \forall \text{ loops } l. \quad (11)$$

C4: (Latency Bound Constraints)

$$h_p(x) \leq H \quad \forall \text{ IO-critical path } p. \quad (12)$$

Recall that if  $(i, j)$  is a crossing edge,  $x_{1,i}x_{2,j} + x_{2,i}x_{1,j}$  equals to 1. Therefore,  $c_{i,j}$  is contributed to the objective function. Constraint C1 expresses that the total node sizes assigned to module  $b$  cannot be beyond the size limit  $S_b$ . C2 states that each node should be placed into one and only one module. Constraint C3 enforces that the delay-to-register ratio of each loop is not greater than  $K$ . Thus, the iteration bound of the partitioned circuit is not greater than  $K$ . Similarly, C4 limits the total delay of each IO-critical path.

We don't have to specify all loops in constraint C3. Since every loop is composed of simple loops, we have the following lemma:

*Lemma 1:* Given a number  $K$ , if  $g_l(x)$  is not greater than  $K$  for all simple loops  $l$ , then  $g_l(x)$  is not greater than  $K$  for all loops  $l$ .

Lemma 1 implies that only the simple loops need be considered in constraint C3.

#### B. Lagrangian Problem

A vector of Lagrange multipliers is used to merge the nonlinear constraints of the performance-driven partitioning problem into the objective function. Let  $\pi_c$  and  $\pi_p$  represent the number of the simple loops and IO-critical paths, respectively, and  $\lambda$  denote the vector of Lagrange multipliers  $(\lambda_{g_1}, \dots, \lambda_{g_{\pi_c}}, \lambda_{h_1}, \dots, \lambda_{h_{\pi_p}})$ . Using Lagrangian relaxation

[24], we absorb the constraints (11) and (12) into the objective function (8) as a Lagrangian problem as follows:

$$\max_{\lambda \geq 0} \min_x L(x, \lambda) \quad (13)$$

subject to constraints  $C1$  and  $C2$ , where

$$\begin{aligned} L(x, \lambda) = & \sum_{(i,j) \in E} c_{i,j}(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}) \\ & + \sum_{\forall \text{ simple loop } l} \lambda_{g_l}(g_l(x) - K) \\ & + \sum_{\forall \text{ IO-critical path } p} \lambda_{h_p}(h_p(x) - H). \end{aligned} \quad (14)$$

The Lagrangian problem is decomposed into two subproblems.

*Dual Problem:* Given a vector  $x$ , we can represent (14) as a function of a vector  $\lambda$ , i.e.,  $L_x(\lambda)$ . Thus, the dual problem can be written as

$$\max_{\lambda \geq 0} L_x(\lambda). \quad (15)$$

*Primal Problem:* Let  $F_{i,j}$  and  $Q_{i,j}$  denote the set of the simple loops and IO-critical paths passing edge  $(i,j)$ . We rewrite (14) in terms of edges. Let us define  $a_{i,j}$  and  $\beta$  as follows:

$$a_{i,j} = c_{i,j} + \sum_{l \in F_{i,j}} \frac{\delta}{r_l} \lambda_{g_l} + \sum_{p \in Q_{i,j}} \delta \lambda_{h_p} \quad (16)$$

$$\begin{aligned} \beta = & \sum_{\forall \text{ simple loop } l} \lambda_{g_l} \left( \frac{d_l}{r_l} - K \right) \\ & + \sum_{\forall \text{ IO-critical path } p} \lambda_{h_p} (d_p - H). \end{aligned} \quad (17)$$

Given a vector  $\lambda$ , we can represent (14) as a function of vector  $x$ , i.e.,  $L_\lambda(x)$ . Thus, the primal problem can be rewritten as

$$\min L_\lambda(x) = \min \sum_{(i,j) \in E} a_{i,j}(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}) + \beta \quad (18)$$

subject to constraints  $C1$  and  $C2$ , where  $\beta$  represents the constant contributed by  $\lambda$ , the number of registers  $r_l$ , the node delay  $d_l$  of loop  $l$ , etc.

#### IV. LAMP

We adopt a Lagrangian Approach on Modular Partitioning (LAMP) which solves the performance-driven partitioning problem through primal and dual iterations on the Lagrangian problem. A heuristic Quadratic Boolean Programming algorithm [2], [26] is used to solve the primal problem and generate a solution  $x$ . For the dual problem based on  $x$ , the values  $a_{i,j}$  for the next primal-dual iteration are updated using the subgradient approach. The iteration proceeds until the bound of all loops and paths are within the given limits.

#### A. QBP Method for Primal Problem

Let each edge  $(i,j)$  be associated with a cost  $a_{i,j}$ . From (18), the primal problem becomes a traditional partitioning problem of minimizing the total costs of crossing edges while satisfying the size constraint. Note that the primal problem does not incorporate any timing constraint. Any traditional partitioning algorithm [4], [8], [11], [14], [23], [27] can be adopted to solve the primal problem. We adopt a heuristic Quadratic Boolean Programming method (QBP) [2], [26] to optimize the primal problem, since QBP is easily extended to handle multiway partitioning.

#### B. Subgradient Method for Dual Problem Using Cycle Mean Approach

Once a solution  $\lambda$  of the dual problem (15) is generated, formula (16) is applied to update the edge costs. Given an edge  $(i,j)$ , we need all  $\lambda_{g_l}$  of the simple loops passing  $(i,j)$  and all  $\lambda_{h_p}$  values of the IO-critical paths passing  $(i,j)$  to calculate  $a_{i,j}$  as in (16). The number of such simple loops and IO-critical paths may be exponential in the number of the nodes of the given data flow graph.

However, we do not enumerate all loops and paths when calculating edges costs by (16). Instead, only some particular loops and paths are considered. For an optimal solution to problem (15), the vector of Lagrange multipliers  $\lambda$  has the properties that  $\lambda_{g_l}$  and  $\lambda_{h_p}$  are larger than zero only if  $g_l(x)$  and  $h_p(x)$  are not less than  $K$  and  $H$ . We define these loops and paths as *active* loops and paths, respectively.

*Active Loops and Paths:* A simple loop  $l$  is called active, if  $g_l(x)$  is not less than  $K$ . An IO-critical path  $p$  is called active, if  $h_p(x)$  is not less than  $H$ . If a loop or path is not active, we call it inactive.

*Active Edges:* An edge  $(i,j)$  is active, if it is traversed by an active loop or an active path. If  $(i,j)$  is not active, we call it inactive.

*Dominant Loops and Paths:* A simple loop  $l$  is called a dominant loop of edge  $(i,j)$ , if  $l$  has the maximum delay-to-register ratio among all loops passing  $(i,j)$ . We define an IO-critical path  $p$  to be a dominant path of  $(i,j)$  if the total delay of path  $p$  is the maximum among all IO-critical paths passing  $(i,j)$ .

*Dominant Prices:* If  $l$  is a dominant loop of  $(i,j)$ , we define the dominant-loop price of  $(i,j)$  as the delay-to-register ratio of  $l$ . If  $p$  is a dominant path of  $(i,j)$ , then the dominant-path price of  $(i,j)$  is given by the total delay of  $p$ .

Given an edge  $e = (i,j)$ , let  $f_e$  be the dominant-loop price of  $e$  and let  $r_e$  be the number of registers on the dominant loop of  $e$ . We randomly choose a dominant loop to calculate  $r_e$  if  $e$  has multiple dominant loops. Let  $q_e$  represent the dominant-path price of  $e$  defined analogously.

*Updating Edge Costs:* In practice, only dominant prices of each edge are determined and the edge cost is updated by the subgradient approach [24]. Equation (16) is not used to calculate the edge costs, since it needs to generate all the loops and paths. We utilize the minimum cycle mean algorithm [13] to calculate the dominant-loop price of each edge. An all-

pairs shortest-paths algorithm can be adopted to compute the dominant-path price of each edge.

At the  $k$ th iteration, let  $x^k$  and  $Cut^k$  denote the derived  $x$  vector and crossing edge count of the primal problem, respectively. Let  $a_{i,j}^k$  be the cost of edge  $e$  at the  $k$ th iteration. We adopt the subgradient method to generate the new edge cost  $a_{i,j}^{k+1}$  as

$$a_{i,j}^{k+1} = \max \left\{ 0, a_{i,j}^k + t^k * (f_e - K) * \frac{\delta}{r_e} + t^k * (q_e - H) * \delta \right\} \quad (19)$$

where  $t^k$  is defined as

$$t^k = \frac{\alpha |Cut^* - Cut^k|}{\sum_{e \in E} |f_e - K|^2 + \sum_{e \in E} |q_e - H|^2} \quad (20)$$

and  $\alpha$  and  $Cut^*$  are two given positive numbers. By (19), we increase the costs of active edges and decrease those of inactive edges, using subgradient approach. This captures the actual direction of the edge cost changes of active and inactive edges with respect to (16).

### C. Algorithm

We perform primal-dual iterations to solve the performance-driven partitioning problem. Let  $K$  and  $H$  denote the iteration and latency bound limits, respectively, and  $Cut^*$  and  $\alpha$  represent the estimated crossing edge count and a positive value, respectively. The LAMP algorithm is described as follows:

*LAMP Algorithm:*

1. Assign values to  $K$ ,  $H$ ,  $\alpha$  and  $Cut^*$ .
2. Initialize  $k \leftarrow 0$ ;  $a_{i,j}^0 = c_{i,j}$ .
3. Call QBP to generate partition  $P^k = (V_1^k, V_2^k)$  with crossing edge cost  $Cut^*(P^k)$ .
4. a. Use the minimum cycle mean algorithm to calculate the iteration bound of the partitioned circuit and  $f_e$  and  $r_e$  of each edge  $e$ ;  
b. Use an all-pair shortest-path algorithm to calculate the latency bound and  $q_e$  of each edge  $e$ ;  
c. If iteration and latency bounds are not greater than  $K$  and  $H$  respectively, then stop.
5. Compute  $t^k$  by (20).
6. For each edge  $e = (i, j) \in E$ :  
Compute  $a_{i,j}^{k+1}$  by (19).
7. Set  $k \leftarrow k + 1$  and goto 3.

A heuristic Quadratic Boolean Programming algorithm [2], [26] is called to solve the primal problem and generate a solution  $x$  at Step 3. For the dual problem based on  $x$ , we call a minimum cycle mean algorithm [13] and an all-pairs shortest-paths algorithm to obtain the dominant prices of each edge at Step 4. If the bounds of all loops and paths satisfy the constraints, the algorithm stops; otherwise, we calculate the subgradient on the dominant prices and update the values  $a_{i,j}$  for the next primal-dual iteration at Step 5 and 6. By following the suggestion of [24], the parameter  $\alpha$  is set to 1.3 in our experiments.

TABLE I  
CHARACTERISTICS OF TEST CASES

cir.	#reg.	#comb.	$J$	$B$
sys1	342	8,280	6,373	5,447
sys2	472	3,378	0	4,421
sys3	521	6,325	2,527	3,238
sys4	380	3,850	4,922	5,545
sys5	545	12,172	4,241	4,876
sys6	357	3,026	0	3,724
sys7	607	4,990	996	3,563

### D. Algorithm Complexity

Given a data flow graph  $G = (V, E)$ , let  $n = |V|$  and  $m = |E|$ . Thus, the time complexity of the all-pair shortest-path algorithm is  $O(n^3)$ . The minimum cycle mean algorithm takes  $O(nm)$  time. The complexity of QBP is  $O(n^2)$ . The time complexity of each iteration of LAMP is dominated by the all-pair shortest-path algorithm. For simplicity, the steps 3–6 of the algorithm are iterated a maximum number of 20 times. Therefore, the time complexity of our algorithm is  $O(n^3)$ .

## V. EXPERIMENTAL RESULTS

In our experiments we used the same seven industrial circuits from [25], [26] as our test cases. The test cases range in size from from 3026 to 12 172 combinational blocks and from 342 to 607 registers. All combinational blocks are of unit size with the exception of some in test case *sys1* which have size 2. Five of these circuits contain feedback loops.

*External-Loop Constraint:* Even though a system may have no feedback from its primary outputs to its primary inputs, it can interact with external systems. Hence, macroscopically, there possibly exist external feedback loops from primary outputs to 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 primary input to primary output, let  $d_p$ ,  $\hat{d}_p$ , and  $r_p$  be the sum of combinational block delays, the sum of intermodule delays, and the number of registers on path  $p$ . The *path delay bound* of a circuit is defined by

$$B = \max \left\{ \frac{d_p + \hat{d}_p}{r_p} \mid p \in P_{IO} \right\} \quad (21)$$

where  $P_{IO}$  is the set of all paths from the primary input to the primary output.

Table I shows the characteristics of these test cases. The second and third columns list the numbers of registers and combinational blocks, respectively. In the fourth and fifth columns we have the *iteration bound*  $J$  and the *path delay bound*  $B$  defined in the following subsection, respectively. Since *sys2* and *sys6* don't have feedback loops, their iteration bounds are equal to zero.

For performance-driven partitioning, one parameter to be decided is the value of intermodule delay  $\delta$ . As indicated in

TABLE II  
TIMING CONSTRAINTS WITHOUT EXTERNAL-LOOP CONSTRAINT

cir.	$K$	$H$	<i>iteration</i>	<i>exe</i>
sys1	6,373	18,195	1	665
sys2	2,652	9,758	3	173
sys3	2,527	13,624	1	553
sys4	4,922	25,807	2	244
sys5	4,241	32,347	1	949
sys6	2,236	10,239	1	181
sys7	2,137	10,673	1	263

[6], the intermodule delay can increase to nearly 100% of the clock cycle period. Therefore, we set  $\delta$  to be of 60% of  $T^* = \max\{J, B\}$ , which is calculated before partitioning. For example, the intermodule delay  $\delta$  associated with *sys7* is equal to 2137, since the value  $T^*$  of *sys7* equals to 3563.

We compared our algorithm LAMP to the Fiduccia-Mattheyses (FM) algorithm [8], Topological Timing Cut (TTC) [20], Flow Timing Cut (FTC) [20] and the Shih-Kuh (SK) algorithm [26]. The SK algorithm was devised to generate a partition such that the delay between registers of the resulted partition is within a given timing limit. However, the SK algorithm does not take retiming into consideration. The TTC and FTC algorithms are the first algorithms to take retiming into consideration. The objective of the FM algorithm is to minimize the crossing edge count without any timing constraints. All algorithms were executed on a single-processor SUN SPARC 10 workstation under the C/UNIX environment. The results of FM are chosen from the best of 20 runs each. The size limit of each module was set to 60% of the circuit size. In the following, we present both the experimental results with and without the external-loop constraint.

#### A. Experiments without External-Loop Constraint

Table II shows the timing constraints used in the experiments. In the second column,  $K$  represents the iteration bound limit (i.e., the iteration bound with no intermodule delays). For most test cases, the value of  $K$  is equal to the iteration bound  $J$  before partitioning shown in Table I. For *sys2*, *sys6*, and *sys7*,  $K$  is equal to the intermodule delay  $\delta$ , since their iteration bounds are less than  $\delta$  and the clock period is dominated by the intermodule delay. In the third column,  $H$  stands for the latency bound limit. The LAMP algorithm has another parameter  $cut^*$ , which denotes the estimated crossing edge count. We set  $H$  and  $cut^*$  from the results of TTC. Computational experiments shows that the determination of  $cut^*$  did not have a great influence on the performance of the algorithm. Therefore, if the result of TTC is not available, the reader can use any well-known bipartitioning package to estimate crossing edge count. The fourth column shows the number of iterations LAMP takes to stop. The last column reports the average execution time of each iteration of LAMP in seconds. Generally speaking, the execution time of LAMP is comparable to that of the FM algorithm. However, TTC and FTC are much faster than LAMP.

TABLE III  
CROSSING EDGE COUNT WITHOUT EXTERNAL-LOOP CONSTRAINT

circuit	FM cut	TTC cut	FTC cut	LAMP cut
sys1	2,860	3,144	3,043	2,991
sys2	875	878	948	663
sys3	1,422	2,236	1,952	1,254
sys4	1,045	1,467	1,258	820
sys5	3,465	4,889	4,889	3,542
sys6	848	1,175	1,004	827
sys7	1,103	1,304	1,304	1,007

*Crossing Edge Count:* Table III shows the experimental results regarding crossing edge count without the external-loop constraint. The reductions in crossing edge counts are as follows. When compared to FM, LAMP achieved  $-4.58 \sim 24.23\%$  with an average reduction of 8.85%. LAMP achieved  $4.87 \sim 44.10\%$  with an average reduction of 24.26%, versus TTC. When compared to the FTC, LAMP achieved  $1.71 \sim 35.76\%$  with an average reduction of 21.81%.

*Clock Period:* Table IV gives detailed information for our experiments on clock period. In the first row,  $\hat{T}$  associated with FM is the maximum delay between registers before retiming. In Table IV, with the exception of the first and second columns, each column contains two subcolumns. The data in the first subcolumn represents the  $J$  value derived from (4) after partitioning. The  $T$  value in the second subcolumn is the clock cycle period of the partitioned circuit achieved by retiming. Note that we adopt the retiming algorithm of [19].  $J$ , calculated after partitioning, will dominate the optimal clock cycle period during retiming. However, if  $J < \delta$ , then  $\delta$  will dominate the clock cycle period because  $\delta$  is not decomposable.

Because all of the loops in our test cases are quite small and strongly connected, no loops are cut by TTC, FTC, and LAMP for all test cases. However, FM cuts the loops in *sys3* and *sys5*. Therefore, LAMP, TTC, and FTC obtain the same iteration bound and clock period for all test cases. Compared with FM, LAMP achieves 38.41% and 38.48% clock period reductions for *sys3* and *sys5*, respectively.

By using the clock period generated by LAMP to be the timing constraints, we made comparisons with Shih and Kuh's (SK) performance-driven partitioning algorithm [26]. For all test cases, SK does not produce feasible solutions. Note that SK generates a partition where the delay between registers is within the timing constraint. Therefore, we have demonstrated that retiming can achieve better clock period.

*System Latency:* Table V gives the detailed information regarding our experiments on system latency. In Table V, with the exception of the first column, each column contains two subcolumns. The data in the first subcolumn is  $M$  derived from (5) after partitioning.  $M'$  in the second subcolumn is the system latency of the partitioned circuit obtained through retiming.

TABLE IV  
ITERATION BOUND  $J$  AND CLOCK PERIOD  $T$  AFTER RETIMING WITHOUT EXTERNAL-LOOP CONSTRAINT

circuit	FM ( $\hat{T}$ )	FM		TTC		FTC		LAMP	
		J	T	J	T	J	T	J	T
sys1	9,456	6,373	6,373	6,373	6,373	6,373	6,373	6,373	6,373
sys2	7,074	0	2,652	0	2,652	0	2,652	0	2,652
sys3	6,801	3,908	4,103	2,527	2,527	2,527	2,527	2,527	2,527
sys4	9,258	4,922	4,922	4,922	4,922	4,922	4,922	4,922	4,922
sys5	9,435	6,268	6,894	4,241	4,241	4,241	4,241	4,241	4,241
sys6	8,656	0	2,236	0	2,236	0	2,236	0	2,236
sys7	6,597	2,137	2,137	2,137	2,137	2,137	2,137	2,137	2,137

TABLE V  
LATENCY BOUND  $M$  AND SYSTEM LATENCY  $M'$  AFTER RETIMING WITHOUT EXTERNAL-LOOP CONSTRAINT

cir.	FM		TTC		FTC		LAMP	
	M	M'	M	M'	M	M'	M	M'
sys1	20,418	25,492	18,195	25,492	18,195	25,492	18,195	19,119
sys2	11,785	15,912	9,758	10,608	9,789	10,608	8,127	10,608
sys3	16,897	20,515	13,624	17,689	14,872	17,689	13,308	17,689
sys4	25,951	34,454	25,807	34,454	25,843	34,454	25,594	29,532
sys5	37,853	41,364	29,014	29,687	29,014	29,687	28,435	29,687
sys6	11,147	13,416	10,239	13,416	9,973	13,416	9,502	13,416
sys7	13,283	19,233	10,673	12,822	10,673	12,822	10,140	12,822

When compared to the FM, TTC, and FTC algorithms, the latency reductions are as follows. Versus FM, LAMP reduced  $M$  by 1.37 ~ 31.03% with an average reduction of 18.25% and reduced  $M'$  by 0 ~ 33.33% with an average reduction of 19.54%. When compared with TTC, LAMP reduced  $M$  by 0 ~ 16.71% with an average reduction of 4.85% and reduced  $M'$  by 0 ~ 25.00% with an average reduction of 5.61%. When compared with FTC, LAMP reduced  $M$  by 0 ~ 16.97% with an average reduction of 5.73% and reduced  $M'$  0 ~ 25.00% with an average reduction of 5.61%.

### B. Experiments with External-Loop Constraint

According to the external-loop constraint, we have to take into account the path delay in the calculation of iteration bound. In this case the clock period of a given partitioned circuit is dominated by

$$A = \max\{J, B\}.$$

Since there possibly exist external feedback loops from primary outputs to primary inputs, the register count of each path from primary input to primary output cannot be changed after retiming; otherwise, the functionality of the circuit will be modified. Thus, according to (2), the system latency after retiming is proportional to the clock period after retiming. We imposed only the iteration bound limit in our experiment with external-loop constraint. Therefore, LAMP does not call an all-pair shortest-path algorithm to compute the latency bound.

Table VI shows the timing constraint. In the second column,  $K$  represents the iteration bound limit. The third and fourth columns stand for the number of iterations that LAMP takes

TABLE VI  
TIMING CONSTRAINT WITH EXTERNAL-LOOP CONSTRAINT

cir.	$K$	iteration	exe
sys1	6,373	18	82
sys2	5,207	7	20
sys3	4,020	11	99
sys4	6,360	19	47
sys5	6,367	1	114
sys6	4,502	5	12
sys7	3,644	1	38

TABLE VII  
CROSSING EDGE COUNT WITH EXTERNAL-LOOP CONSTRAINT

circuit	FM cut	TTC cut	FTC cut	LAMP cut
sys1	2,860	3,144	3,043	3,134
sys2	875	878	948	847
sys3	1,422	2,236	1,952	1,629
sys4	1,045	1,467	1,258	1,032
sys5	3,465	4,889	4,889	3,478
sys6	848	1,175	1,004	817
sys7	1,103	1,304	1,304	1,141

and the average execution time of each iteration, respectively. Since LAMP does not calculate the iteration bound, the execution time listed in Table VI is much less than that in Table II.

TABLE VIII  
ITERATION BOUND  $A$  AND CLOCK PERIOD  $T$  AFTER RETIMING WITH EXTERNAL-LOOP CONSTRAINT

cir.	FM ( $\hat{T}$ )	FM		TTC		FTC		LAMP	
		A	T	A	T	A	T	A	T
sys1	9,456	8,371	9,238	6,373	6,653	7,076	7,076	6,373	6,653
sys2	7,074	7,074	7,215	5,207	5,310	5,342	5,342	5,206	5,310
sys3	6,801	5,338	5,444	4,760	4,760	4,976	4,976	4,019	4,099
sys4	9,258	8,239	8,631	7,661	7,661	8,083	8,083	6,360	7,505
sys5	9,435	7,666	8,432	7,674	7,827	7,674	7,827	6,366	6,493
sys6	8,656	6,544	6,544	5,611	5,611	5,334	5,429	4,502	5,042
sys7	6,597	5,103	5,227	3,897	3,897	3,897	3,897	3,644	3,935

*Crossing Edge Count:* Table VII gives our experiments regarding crossing edge count. The reductions in crossing edge counts are as follows. When compared to FM, LAMP achieved  $-14.55 \sim 3.65\%$  reduction with an average of  $-1.85\%$ . Versus TTC, LAMP achieved crossing count reduction of  $0.31 \sim 30.46\%$  with an average of  $18.92\%$ . When compared to the FTC, LAMP achieved  $-2.99 \sim 28.86\%$  reduction with an average of  $14.59\%$ . In general, LAMP produced crossing count results comparable to FM and much better than TTC and FTC.

*Clock Period:* Table VIII gives the detailed information of our experiments. The data in the first subcolumn represents the  $T$  derived from (22) after partitioning. The  $T$  in the second subcolumn is the clock cycle period of the partitioned circuit after retiming. When compared to the FM, TTC, and FTC, the clock cycle period reductions are as follows. When compared with FM, LAMP achieved  $16.95 \sim 31.20\%$  reduction in  $A$  with an average reduction of  $24.92\%$  and  $13.04 \sim 27.98\%$  reduction in  $T$  with an average reduction of  $23.25\%$  for  $T$ . When compared with TTC, LAMP achieved  $0 \sim 19.76\%$  reduction in  $A$  with an average of  $10.44\%$  and  $-0.97 \sim 17.04\%$  reduction in  $T$  with an average reduction of  $6.01\%$ . When compared with FTC, LAMP achieved  $2.54 \sim 21.31\%$  reduction in  $A$  and an average reduction of  $12.25\%$  and  $-0.97 \sim 17.62\%$  reduction in  $T$  with an average improvement of  $7.70\%$ .

Using the clock period generated by LAMP to be the timing constraints as shown in second column of Table IX, Table IX shows the experiment results of SK algorithm. For most test cases, SK does not produce feasible solutions.

*System Latency:* Because the system latency after retiming is proportional to the clock period after retiming with the external-loop constraint, LAMP achieves  $23.25\%$ ,  $6.01\%$ , and  $7.70\%$  reductions, respectively, compared with FM, TTC and FTC.

## VI. CONCLUDING REMARKS

We have proposed an efficient performance-driven two-way partitioning algorithm which considers clock period and latency simultaneously based on the limits of the iteration and latency bounds. Experimental results on seven industrial circuits have demonstrated our algorithm is able to achieve an average of  $23.25\%$  clock cycle period and  $23.25\%$  latency reduction compared to the best results produced by 20 runs on

TABLE IX  
SK'S RESULTS WITH EXTERNAL-LOOP CONSTRAINT

cir.	$T$	SK cut
sys1	6,653	X
sys2	5,310	923
sys3	4,099	X
sys4	7,505	X
sys5	6,493	3994
sys6	5,042	X
sys7	3,935	1,275

each test case using a Fiduccia-Mattheyses algorithm. In terms of the number of the crossing edges, our result is only  $1.85\%$  more than the result of the Fiduccia-Mattheyses algorithm without timing constraints.

Furthermore, we can easily expand our algorithm to  $K$ -way partitioning, since QBP can handle  $K$ -way partitioning.

## ACKNOWLEDGMENT

The authors would like to thank F. Berman, E. A. Lee, and K. K. Parhi for their helpful discussions about system latency. Thanks are also given to the reviewers for their valuable suggestions.

## REFERENCES

- [1] T. Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms," in *Proc. ACM/IEEE 26th Design Automation Conf.*, 1989, pp. 775–778.
- [2] R. E. Burkard and T. Bonniger, "A heuristic for quadratic boolean programs with applications to quadratic assignment problems," *Euro. J. Operational Res.*, vol. 13, pp. 372–386, 1983.
- [3] P. K. Chan, M. D. Schlag, and J. Y. Zien, "Spectral  $K$ -way ratio-cut partitioning and clustering," in *Proc. 30th ACM/IEEE Design Automation Conf.*, 1993, pp. 749–754.
- [4] C. K. Cheng and Y. C. Wei, "An improved two-way partitioning algorithm with stable performance," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1502–1511, Dec. 1991.
- [5] T. H. Cormen, C. E. Lerserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
- [6] D. A. Doane and P. D. Franzon, Eds., *Multichip Module Technologies and Alternatives—The Basics*. New York: Van Nostrand Reinhold, 1993, pp. 666–667.

- [7] W. E. Donath and A. J. Hoffman, "Lower bounds for the partitioning of graphs," *IBM J. Res. Dev.*, pp. 420–425, 1973.
- [8] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. 19th ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
- [9] F. Berman, personal communication, 1993.
- [10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of  $\mathcal{NP}$ -Completeness*. San Francisco, CA: W. H. Freeman, 1979.
- [11] L. Hagen and A. B. Kahng, "New spectral method for ratio cut partitioning and clustering," *IEEE Trans. Computer-Aided Design*, pp. 1074–1085, 1991.
- [12] J. Hwang and A. E. Gamal, "Optimal replication for min-cut partitioning," in *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, Nov. 1992, pp. 432–435.
- [13] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.
- [14] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.
- [15] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438–446, May 1984.
- [16] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.
- [17] E. A. Lee, personal communication, 1993.
- [18] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI and Comput. Syst.*, vol. 1, no. 1, pp. 41–67, 1983.
- [19] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.
- [20] L. T. Liu, M. Shih, N. C. Chou, C. K. Cheng, and W. Ku, "Performance-driven partitioning using retiming and replication," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1993, pp. 269–299.
- [21] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, pp. 178–195, Feb. 1991.
- [22] K. K. Parhi, personal communication, 1993.
- [23] C. Sechen and D. Chen, "An improved objective function for mincut circuit partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1988, pp. 502–505.
- [24] J. F. Shapiro, *Mathematical Programming: Structures and Algorithms*. Wiley, New York, 1979.
- [25] M. Shih, E. S. Kuh, and R.-S. Tsay, "Performance-driven system partitioning on multi-chip modules," in *Proc. 29th ACM/IEEE Design Automation Conf.*, 1992, pp. 53–56.
- [26] M. Shih and E. S. Kuh, "Quadratic boolean programming for performance-driven system partitioning," in *Proc. 30th ACM/IEEE Design Automation Conf.*, 1993, pp. 761–765.
- [27] Y. C. Wei and C. K. Cheng, "Ratio cut partitioning for hierarchical designs," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 911–921, July 1991.



**Lung-Tien Liu** received the B.S. and M.S. degrees in 1987 and 1989, respectively, from the National Taiwan University, Taipei, Taiwan, and the Ph.D. degree from the University of California, San Diego, in 1994, all in computer science.

Since July 1994, he has been a Member of Technical Staff of AT&T Bell Laboratories, Murray Hill, NJ. His research interests include optimization algorithms for CAD, timing-driven placement, and circuit partitioning.



**Minshine Shih** received the B.S. and M.S. degrees in electrical engineering from National Taiwan University and the University of Southern California, and the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1995.

From 1994 to 1995, he was a Member of Technical Staff with Digicom Systems Inc., Milpitas, CA, working on modem and various multimedia DSP algorithms. Currently, he is a Member of Technical Staff of Chromatic Research Inc., developing ITU-T V.34 modem for a VLIW processor. His research interests include combinatorial optimization, integer programming, DSP algorithms, and architectures.



**John Lillis** received the B.S. degree in computer science from the University of Washington in 1989 and the M.S. degree in computer science from the University of California, San Diego in 1992. He is currently pursuing the Ph.D. degree in the Computer Science Department of the University of California, San Diego. His research interests are in computer-aided design of VLSI circuits and combinatorial optimization.



**Chung-Kuan Cheng** (S'82–M'84–SM'95) received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, and the Ph.D. degree in electrical engineering and computer sciences from University of California, Berkeley in 1984.

From 1984 to 1986 he was a senior CAD engineer at Advanced Micro Devices Inc. In 1986, he joined the University of California, San Diego, where he is currently a Professor with the Computer Science and Engineering Department. His research interests include network optimization and design automation

on microelectronic circuits.

Dr. Cheng is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN.