PMP: Performance-Driven Multilevel Partitioning by Aggregating the Preferred Signal Directions of I/O Conduits

Chan-seok Hwang and Massoud Pedram

Abstract - In this paper, we present a new performance-driven multilevel partitioning algorithm, which calculates the timing gain of a move in the move-based partitioning strategies based on the aggregation of preferred signal directions. In addition, we propose a new timing-aware multilevel clustering algorithm that uses the connection strength of an edge as the primary objective, and the maximum depth or the maximum hop-count of any path containing the edge as a tiebreaker for the clustering step. These ideas are integrated into a general multilevel partitioning framework, which consists of three phases: uncoarsening, initial partitioning, and coarsening and refinement phases. The benchmarks show that, on average, we can reduce delay by 14.6%, while increasing the cutsize by 1.2% when compared to hMetis[1]**.**

I. Introduction

As the size and complexity of current circuits increase, partitioning plays an ever more important role in the VLSI design process. Partitioning affects not only the efficiency of the subsequence design optimization steps, but also the overall circuit performance. This is because, with the advances in CMOS process technology, circuit delay is heavily dominated by the interconnect delay and the difference between the intra-part (internal) delay and the inter-part (external) delay increases rapidly.

Traditional partitioning approaches [1-3] have been quite successful in reducing cutsize. However, these techniques cannot adequately control the cut count of timing-critical paths in the circuit. Recently, timing-driven partitioning approaches [4-11] have been proposed to simultaneously consider cutsize and the circuit delay. These works may be classified into two categories depending on whether they modify the netlist or not. Most of these reported works modify the netlist, i.e., they use logic replication, retiming, or buffer insertion techniques to reduce the circuit delay while minimizing the cutsize [6][7][8][11]. These methods tend to reduce the circuit delay significantly, compared to cutsize-oriented methods such as hMetis [1]. However, gate replication used by these methods and can result in a significant increase in the chip area. In addition, some of these methods suffer from large cutsize [8] or large runtime [7][11]. Techniques that do not alter the circuit netlist, normally give more weight to the edges that lie on the timing-critical paths in a circuit [9][10]. These techniques require an a priori classification of signal nets into timing-critical and non-critical ones based on a timing analysis of the circuit prior to partitioning. However, these methods suffer from choosing the *K*-most critical paths since the performance in terms of cutsize, delay and runtime heavily depends on the value of *K* [9]. A number of researchers [4][5] have used the signal direction as an indicator of the *timing gain* function during the move-based partitioning process. Examples include "backward edges" [4] and "V-shaped nodes" [5]. These early results motivate the use of signal direction to guide the performance-driven partitioning process.

In this paper, we present a new performance-driven multilevel partitioning algorithm, called PMP, which can minimize circuit delay efficiently by *aggregating* the preferred signal directions as implied by all *input-output conduits* in the circuits (see Section III for a formal definition of I/O conduits.) Unlike the previous approaches [4][5], our timing gain function accounts for the effect of a cell move on the delay of all input-output conduits that go through that cell. This enables PMP to compute the timing gain for each candidate cell to move accurately and efficiently. In addition, we propose a new timing-aware multilevel clustering algorithm that uses the connection strength of an edge as the primary objective, and the maximum depth or the maximum hop-count of any path containing the edge as a tiebreaker for the matching step. These ideas have worked remarkably well in combination with a general multilevel partitioning algorithm such as hMetis [1]. The benchmark results show that, when compared with hMetis, PMP reduces the circuit delay by 14.8%, while increasing the cutsize by only 1.2% on average.

II. Problem Statement

The performance-driven partitioning problem can be stated as follows. Consider a sequential circuit, represented by a directed graph G=(*V*, *E*). Each node $v_i \in V$ has a weight $\delta(v_i)$ which specifies the intrinsic delay associated with v_i. If an edge e is cut, then $d(e)=d_{ext}$ where d_{ext} is the delay of an external edge; otherwise, $d(e)= d_{int}$ where d_{int} is the delay of an internal edge. Let $c(\pi)$ be the number of cut edges along a topological path π, and $|\pi|$ be the number of nodes on that path. The path delay $d(\pi)$ can be calculated as follows [11]:

$$d(\pi) = d_{ext} \cdot c(\pi) + d_{int} \cdot (|\pi| - 1 - c(\pi)) + \sum_{v_i \in \pi} \delta(v_i)$$

The cycle time for graph G is $\Phi_G = max\ d(S)$ where S is a set of all topological paths in a circuit. Let Λ_G denote the cutsize of graph G. The problem of performance-driven bipartitioning is to find a partitioning solution $(V_0|V_1)$ that minimizes a combination of Φ_G and Λ_G while satisfying capacity constraints.

III. Signal Direction Constraints

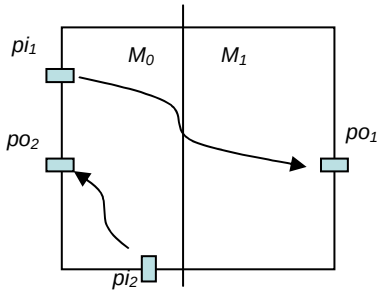
In this section, we introduce the notion of a (preferred) signal direction, and the resulting constraint, which will be used to optimize circuit delay efficiently at the minimum cost of cutsize.

Let's denote the set of primary inputs of a circuit as PI, the set of primary outputs as PO, and the set of flip-flops as FF.

Definition: We define an input-output (I/O) conduit as the set of all topological paths from some input node (PI or FF) to some output node (PO or FF.) An I/O conduit, σ, is simply identified by the corresponding input (PI or FF) and output (PO or FF).

Notice that the maximum number of I/O conduits in a sequential circuit netlist is $z = (n_I+n_F)*(n_O+n_F)$ where n_I, n_O and n_F denotes the number of circuit PI's, PO's and FF's, respectively. An I/O conduit then belongs to one of the following types: PI→PO, PI→FF, FF→FF, or FF→PO. If we assume that the locations of PI's, PO's and FF's are fixed in one part or the other, then all I/O conduits will have a unique signal direction in a given bipartition. In a bipartition, if signal directions of all I/O conduits are satisfied, then the bipartition will be optimum in terms of the path delay.

Figure 1 shows the signal direction constraints between a signal source node $s(e)$ and a signal target node $t(e)$ for an edge e. An I/O conduit σ_1 from pi_1 to po_1 comprises of a single topological path $pi_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow po_1$. The minimum achievable cut count of σ_1 is one since pi_1 and po_1 are located in different parts. The signal directions of edges of σ_1 should be from part M_0 to part M_1 in order to obtain this minimum cut count. Let $P(v_i)$ denote the part that node v_i is assigned to i.e., $P(v_i) = 0$ if v_i is put in M_0, otherwise,



σ_1: $pi_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow po_1$

$e_1(pi_1,v_1)$, $e_2(v_1,v_2)$, $e_3(v_2,v_3)$, $e_4(v_3,po_1)$

σ_2: $pi_2 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow po_2$

$e_5(pi_2,v_4)$, $e_6(v_4,v_5)$, $e_7(v_5,v_6)$, $e_8(v_6,po_2)$

Signal Direction Constraints:

$P(s(e_i)) \leq P(t(e_i))$, $1 \leq i \leq 4$ for σ_1

$P(s(e_i)) = P(t(e_i)) = 0$, $5 \leq i \leq 8$ for σ_2

where $P(v_i)$ is a part number (0 or 1) of v_i

Figure 1. Signal direction constraints.

$P(v_i) = 1$. Notice that $P(v_i)$ of the source node v_i of an edge e of σ_1 should not be any larger than $P(v_j)$ of the target node v_j of that edge. For an I/O conduit σ_2, comprising of a single path $pi_2 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow po_2$, both source and target nodes of edges on σ_2 should be put in M_0 in order to satisfy the signal direction constraint of σ_2 (the minimum achievable cut count of σ_2 is zero). Based on this discussion, we define *signal direction constraints* (SDC's) for a vertical cut line as follows:

SDC^1: if $SD(\sigma)=LL$, $\forall\, e_i \in \sigma$, $P(s(e_i)) = P(t(e_i)) = 0$

SDC^2: if $SD(\sigma)=RR$, $\forall\, e_i \in \sigma$, $P(s(e_i)) = P(t(e_i)) = 1$

SDC^3: if $SD(\sigma)=LR$, $\forall\, e_i \in \sigma$, $P(s(e_i)) \leq P(t(e_i))$

SDC^4: if $SD(\sigma)=RL$, $\forall\, e_i \in \sigma$, $P(s(e_i)) \geq P(t(e_i))$

where $SD(\sigma)$ denotes the signal direction of σ, which is one of *LL*, *RR*, *LR* or *RL*. Clearly, *LL* (*RR*) implies that both start and end nodes of the conduit are located in M_0 (M_1), whereas *LR* (*RL*) means that the start node of the conduit is in M_0 (M_1) while the end node of the conduit is in M_1 (M_0). Based on the above definitions, each edge on any topological path in an I/O conduit has the same preferred signal direction. Even though many topological paths of a conduit pass through an edge, the edge has only one signal direction constraint (SDC) for the conduit. This is because all topological paths of a conduit also have the same preferred signal direction. In general, an edge may belong to many conduits, say n I/O, in the circuit, each assigning a preferred signal direction to the edge. Therefore, we need only I/O conduits rather than all topological paths in a circuit when computing SDC's of all edges. Let's assume that n_1 of these conduits are of type *LL* whereas n_2, n_3 and n_4 are of types *RR*, *LR* and *RL*, respectively. ($n=n_1+n_2+n_3+n_4$.) Clearly, n is polynomially bounded. Violating these signal directions cause signal direction violations (SDV's).

A. Timing gain function of a move

A bipartition that satisfies all of the SDC's associated with the I/O conduits seldom exists for any realistic netlist. Even when such a solution exists, it tends to have a huge cutsize, which will require either large routing space overhead or additional metal layers to complete the design. This in turn is likely to cause violation of design specifications such as chip size, yield, and manufacturing cost. Therefore, we must relax the constraints in order to obtain a smooth tradeoff between the circuit delay and cutsize.

Our proposed partitioner, PMP, employs an FM heuristic [12] in the uncoarsening phase, with a modified move gain function accounting for both the signal directions and the cutsize. To manage delay as the optimization objective rather than a constraint to be satisfied, we make use of the violation counts as defined above. More precisely, we define a *timing gain function*, $TG(v_i)$, to quantitatively evaluate the desirability of moving v_i from M_0 to M_1. This gain function is defined as:

$$TG(v_i) \equiv TG(M_0 - \{v_i\}, M_1 + \{v_i\}) = VC(v_i : P(v_i) = 0) - VC(v_i : P(v_i) = 1)$$

Theorem: Given a partitioning solution $V=\{M_0,M_1\}$, the reduction in the total violation count, TVC, of G(V,E) as a result of moving some node v_i from M_0 to M_1 is equal to $TG(v_i)$.

Proof is straightforward and is omitted to save space. Notice that to calculate the timing gain for node v_i, contributions to delay of all I/O conduits containing v_i should be aggregated. Previous approaches [4][5] do not consider this point or assume that signal directions of all I/O conduits are the same [4]. However, this unidirectional assumption does not hold true for practical benchmark circuits, especially in a top-down design flow, where, for example, at least the circuit I/Os are considered as fixed terminals [12]. The timing gain for a node v_i is obtained by summing the counter of signal direction violation (SDV) of each edge e_i connected to the node v_i. The counter of SDV of each edge e_i can be computed using one of the following eight cases.

SDV^1: if $v_i = s(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 0$, then
$TG(v_i)$ −= (SDC^1-*count*(e_i) + SDC^3-*count*(e_i))

SDV^2: if $v_i = s(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 1$, then
$TG(v_i)$ −= (SDC^2-*count*(e_i) + SDC^4-*count*(e_i))

SDV^3: if $v_i = s(e_i)$ and $P(s(e_i)) > P(t(e_i))$, then
$TG(v_i)$ += (SDC^1-*count*(e_i) + SDC^3-*count*(e_i))

SDV^4: if $v_i = s(e_i)$ and $P(s(e_i)) < P(t(e_i))$, then
$TG(v_i)$ += (SDC^2-*count*(e_i) + SDC^4-*count*(e_i))

SDV^5: if $v_i = t(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 0$, then
$TG(v_i)$ −= (SDC^1-*count*(e_i) + SDC^4-*count*(e_i))

SDV^6: if $v_i = t(e_i)$ and $P(s(e_i)) = P(t(e_i)) = 1$, then
$TG(v_i)$ −= (SDC^2-*count*(e_i) + SDC^3-*count*(e_i))

SDV^7: if $v_i = t(e_i)$ and $P(s(e_i)) > P(t(e_i))$, then
$TG(v_i)$ += (SDC^1-*count*(e_i) + SDC^4-*count*(e_i))

SDV^8: if $v_i = t(e_i)$ and $P(s(e_i)) < P(t(e_i))$, then
$TG(v_i)$ += (SDC^2-*count*(e_i) + SDC^3-*count*(e_i))

where *SDC-count*(e_i) represents the count of each type of signal direction constraint for edge e_i, that is, the number of conduits with

the corresponding signal direction that go through the edge. These values are pre-computed before we start the FM partitioning process in order to maintain the polynomial time complexity of the FM iterations under signal direction constraints. The algorithm for setting the *SDC-count* is described in Section IV(D).

We explain the timing gain calculation with the help of an example in Figure 2. Consider moving node v_3 from M_0 to M_1. There are six I/O conduits that go through v_3, and there are four edges connected to this node (see figure for specification of conduits and edges). Before *v3* is moved, edges e_2 and e_4 do not satisfy SDC^2 of conduits σ_4 and σ_6, respectively. This is because $SD(\sigma_4)=SD(\sigma_6)=RR$ but the source and target nodes of these two edges are not in M_1. The number of *SDC* violations is thus four. After v_3 is moved, SDC^2 is satisfied for both e_2 and e_4 while edges e_1 and e_3 do not satisfy SDC^1 of conduit σ_1. This means that by moving v_3, we are able to reduce the cut counts of conduits σ_4 and σ_6 by two each, while the cut count of conduit σ_1 is increased by two. As a result, the total number of *SDC* violations are reduced by two, i.e., the timing gain for the *v_3-move* is two, $TG(v_3) = 2$. Previous methods [4][5] do not calculate the timing gain of this move correctly. More precisely, using "backward edges" of reference [4] does not result in correct calculation of the timing gain for the reason that there is no reduction of the "backward edges" after the *v3*-move when the topological ordering is from M_0 to M_1. Furthermore, since the *v3*-move results in another configuration of "V-shaped nodes" for σ_1, reference [5] would set the timing gain of this move to zero. In conclusion, to globally reduce the total cut count of all I/O conduits, the timing gain should be calculated as is proposed in our present work.

IV. The PMP Algorithm

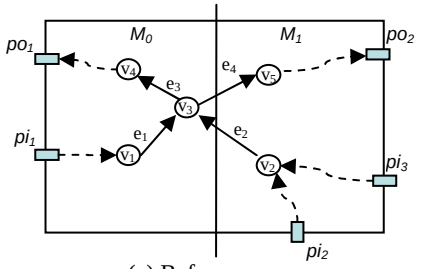
In this section, we describe our proposed *Performance-driven Multilevel Partitioning* (PMP) algorithm. Similar to the hMetis flow [12], PMP consists of three phases: Phase 1 - Timing-aware clustering for the coarsening phase; Phase 2 - Initial partitioning phase; Phase 3 - Performance-driven bipartitioning for the uncoarsening phase.

A. Clustering considering timing criticality

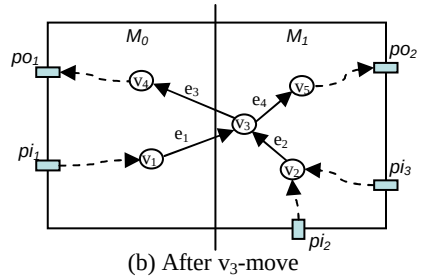
We use the *Heavy Edge Matching* (HEM) algorithm of [12] in order to find a maximal matching in the net list graph. The clustering phase greatly influences the quality of the final partitioning solution in terms of both delay and cutsize. In order to improve the delay, the clustering algorithm should consider the timing criticality of the edge as well as its connectivity strength. However, our experimental results have taught us that the connectivity is a more important consideration at this stage and that delay should be used only as a *tiebreaker* when two or more candidate edges have equal weights. We define terminology needed to quantify the timing criticality of edges. Path depth is defined as the number of intermediate nodes in a path (excluding PI's, PO's and FF's).

Definition: The *Maximum Depth of any Path* of an edge (referred to as the MDP of an edge) is defined as the maximum of the logical depth of any path that goes through that edge.

PMP uses the MDP of edges as a tiebreaker when selecting an unmatched adjacent node u_j of v_i. In particular, consider that there are m matching candidates $u_1,\ldots,u_m$. Suppose the first k of these matches have the same edge weight, which is higher than any other edge weight. Among the remaining candidate matches that have tied based on the HEM selection criterion, $u_1,\ldots,u_k$, PMP chooses the one whose corresponding edge to v_i has the highest MDP value.



(a) Before v_3-move



(b) After v_3-move

σ_1: $pi_1 \rightarrow v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow po_1$, σ_2: $pi_1 \rightarrow v_1 \rightarrow v_3 \rightarrow v_5 \rightarrow po_2$
σ_3: $pi_2 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow po_1$, σ_4: $pi_2 \rightarrow v_2 \rightarrow v_3 \rightarrow v_5 \rightarrow po_2$
σ_5: $pi_3 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow po_1$, σ_6: $pi_3 \rightarrow v_2 \rightarrow v_3 \rightarrow v_5 \rightarrow po_2$
$SDC^1\text{-}count(e_1) = 1$, $SDC^3\text{-}count(e_1) = 1$, $SDC^2\text{-}count(e_2) = 2$,
$SDC^4\text{-}count(e_2) = 2$, $SDC^1\text{-}count(e_3) = 1$, $SDC^4\text{-}count(e_3) = 2$,
$SDC^2\text{-}count(e_4) = 2$, $SDC^3\text{-}count(e_4) = 1$

$VC_{(V3:}P(v_3)=0) = 4$ *// SDC violations before the v_3 move*
$\Rightarrow$ *SDC^2 violated for e_2* and *e_4*, and others are all satisfied.
$VC_{(V3:}P(v_3)=1) = 2$ *// SDC violations after the v_3 move*
$\Rightarrow$ *SDC^1 violated for e_1* and *e_3*, and others are all satisfied.
$\therefore TG(v_3) = VC_{(V3:}P(v_3)=0) - VC_{(V3:}P(v_3)=1) = 2$

Figure 2. Computation of the timing gain for an example move by using aggregate signal direction constraints.

Figure 3 shows an example of how the MDP is used as a tiebreaker when we are searching for a match to node v_1. The MDP for each edge is represented in parenthesis. There are three adjacent nodes v_2, v_3 and v_4 to v_1 in this example. Among them, v_2 and v_4 survive as match candidates after the HEM's edge weight criterion is applied. Finally, v_2 is selected by PMP for the match of v_1 since e_2 connecting v_1 and v_2 has higher MDP value compared to e_4 connecting v_1 and v_4.

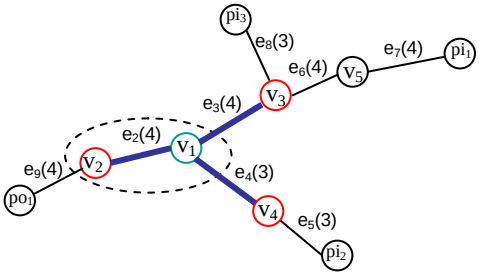


Figure3. Clustering with MDP as a tie-breaker

B. Initial partitioning

During the initial partitioning phase, a bisection of the coarsened hypergraph is computed to minimize cutsize while maintaining that each part contains roughly half of the node weight of the original graph. The node weight represents the area of a node. PMP does not consider delay at this stage because the coarse graph is very small (we set the threshold value of top-level size < 100) and too rough to calculate a meaningful timing gain function.

The initial partitioning solution is then used to decide the locations of FF's. Recall that, in a typical top-down design flow, the locations of the PI's and PO's of the circuit are fixed, whereas the FF's are floating. For sequential circuits, the locations of FF's should be fixed before calculating *SDC-counts* for all edges. Since the cutsize of a circuit greatly depends on the FF locations, we must carefully assign these locations. Therefore, we performed a number of experiments to assess how much the pre-fixed FF locations affect the cutsize in a multilevel partitioning scenario. To save paper space, we briefly mention the experiment results here. The increase rate of cutsize was kept within 10% on average for the benchmark circuits (See Table1) when prefixing FF locations according to the result of the initial partitioning, compared to the case of not pre-fixing FF locations.

C. Uncoarsening with a new gain function

During the uncoarsening phase, the partitioning solution of the coarser graph is projected back to the original graph by going through multiple hierarchies. The hierarchy is constructed during the coarsening phase. At each level, a bipartition refinement starts from the projected partition of its upper level as an initial partition. We use standard FM as a bipartition refinement algorithm. Since our goal is to smoothly exchange between the delay and cutsize, we use the following move gain function for moving node v_i:

$$MG(v_i) = \left\lceil \left(\alpha \cdot CG(v_i) + (1-\alpha) \cdot \frac{TG(v_i)}{\lambda(G)} \right) * 100 \right\rceil$$

where $CG(v_i)$ represents the standard cutsize gain function. $0 \le \alpha \le 1$ is a weight coefficient, and $\lambda(G)$ is the average number of I/O conduits going through any node in a circuit graph G. Note that we must normalize $TG(v_i)$ by $\lambda(G)$ in order to be able to pick a fixed weighting coefficient across different benchmark circuits. We multiply the linear combination of cutsize gain and normalized timing gain with 100 and then take the ceiling. The idea is to produce an integer value for the moving gain of a node that can differentiate between different moves. In our experiments, we used α=0.87 for best results.

D. Time complexity

The pseudo code of our PMP algorithm is shown in Figure 4. First, we compute the MDP. After steps 2 and 3 in Figure 4, we know the preferred signal directions of all I/O conduits because FF's have been assigned to a part. In order to maintain the time complexity of the FM iterations under signal direction constraints (SDC) in the uncoarsening phase, we pre-compute *SDC-counts* for all edges in step 4. We can compute *SDC-counts* as follows: First, all transitive PI's and FF's for each node v_i are calculated and stored as set S_i at that node during the reverse-DFS. Similarly, all transitive PO's and FF's are searched for and stored at set T_i at the node during the subsequent DFS. As a result, we can determine, C_{ij}, the set of all conduits that go thru any edge e_{ij} between nodes v_i and v_j in the circuit graph as the Cartesian product of the sets S_i and T_j. Now, we count the number of conduits of type *LL*, *LR*, *RL*, and *RR* in C_{ij} and thereby initialize the corresponding *SDC-count* for all edges in the circuit.

Theorem: The worst-case space complexity for setting the SDC-counts for all edges is $O(|V| \cdot y)$ whereas the worst-case time complexity is $O(|E| \cdot z)$ where $z=(n_I+n_F)*(n_O+n_F)$ **and** ***$y=n_I+n_F+n_O$.***

PMP ($G_0(V,E)$, TB)

1. If (TB== MDP)
   Compute and store the MDP for each edge of G_0
   Else
   Compute and store the MHP for each edge of G_0
2. Coarsen G_0 until the size of the top-level(m) < 100, producing G_m, G_{m-1}, …, G_1
3. Do the initial partitioning and fix locations of all FF's
4. Compute the SDC counter values for all edges in G_0
5. For i=1; i < m; i++
   Uncoarsen and refine G_i
6. Output the bipartition <G_{00}, G_{01}> of G_0 as the solution

Figure 4. PMP Algorithm

PMMP ($G_0(V,E)$) // *example four-way partitioning code*

1. <G_{00}, G_{01}> = PMP($G_0(V,E)$, MDP); // *first bisection*
2. <G_{000}, G_{001}> = PMP($G_{00}(V,E)$, MHP);
3. <G_{010}, G_{011}> = PMP($G_{01}(V,E)$, MHP);

Figure 5. The PMMP Algorithm

E. Extension to the multiway partitioning

The *Performance-driven, Multilevel, Multiway Partitioning* (PMMP) algorithm is also implemented simply by making iterative calls to PMP to generate two-way, four-way, and eventually K-way partitioning solutions. However, PMMP uses a different tie-breaker during clustering after the first bisection in which MDP is used for the tie-breaker as described in Section IV(A).

For the first bipartitioning step, we do not have enough information to define timing-critical paths in the circuit graph. So we take the topological depth of a path as an indication of its timing criticality and use the MDP of edges to prevent the long paths from being cut several times. On the other hand, in the

succeeding bipartitioning steps, critical paths can be identified more precisely by counting the existing cut count of the paths. Therefore, we can use the maximum of hop-count of any path that goes thru an edge as a measure of timing criticality of that edge (a *hop* means an edge that is been cut). In other words if there is a path that goes thru some edge *e* and that hop-count of that path is already high, we do not want to cut the edge *e* in the current bipartitioning step since it will likely increase the critical path delay after the PMP completes its job. PMMP algorithm is described in Figure 5.

Definition: The *Maximum Hop-count of any Path* of an edge (referred to as the MHP of an edge) is defined as the edge length of the longest path that goes through that edge.

The computation of MHP values can be computed in a similar way to that of computing *SDC-counter* as follows. During the reverse-DFS, the maximum cut count between PI (or FF) to each node v_i, *cut-count*$_{PI}(v_i)$, is calculated. Next, the maximum cut count between PO (or FF) to each node, v_i, *cut-count*$_{PO}(v_i)$, is calculated via the forward-DFS. Then, the MHP value of edge e_i is the sum of *cut count*$_{PI}(v_i)$ and *cut count*$_{PO}(v_j)$, where v_i and v_j represent the source and target node of e_i, respectively. Figure 6 shows the pseudo root node having connections with PI's and FF's, and a pseudo tail node having connections with all PO's and FF's. The total time needed for calculating the MHP values for all edges is $O(|V|+|E|)$.

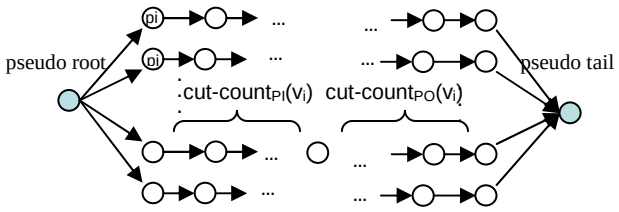


Figure 6. Computation of MHP values in polynomial time

V. Experimental Results

The PMP (or PMMP) algorithm was implemented in C++ on a Sun Ultra Sparc II machine, and tested on ISCAS89 [12] and ITC benchmarks [12]. The characteristics of the benchmark circuits are summarized in Table 1. The circuits were optimized by using the *script.rugged* in SIS [17]. We obtained these optimized circuits from authors of [9]. In our experiments, we arbitrarily assigned locations to the primary inputs and outputs and kept their locations fixed throughout the experiments.

Table 1. ISCAS'89 Benchmarks

Circuit	#Nets	#Nodes	#Primary I/O
cordic	1071	881	25
misex3	1335	1349	28
x3	1270	1369	234
c6288	2403	2435	64
s15850	4274	4326	101
frisc	4385	4501	136
elliptic	4600	4714	245
ex1010	4888	4898	20
pdc	4781	4821	56
s38417	7358	7410	134
b21s	15584	15606	56
b22s	23872	23894	56
b17s	39362	39459	136

Unfortunately, timing-driven partitioning software programs which do not alter netlist (e.g., [4][5]) were not available to us. At the same time, the partitioning algorithm of [9] does not support circuits with any fixed node. Therefore, to compare PMP with other timing-driven partitioning algorithms, we modified the algorithm of reference [9], which we will denote as TPA. In this algorithm, we give more weight to the edges that lie on the timing-critical paths in a circuit. TPA performs circuit-level static timing analysis at each partitioning level in order to determine critical paths and slack. We also compared PMP with hMetis [1]. Therefore, we compared PMP with TPA and hMetis for eight-way partitioning problem.

The maximum delay of the benchmark circuits, Φ_G, is calculated based on the delay model presented in [4]. The chip size for each circuit is assumed to be twice the total area of all nodes in the circuit. For the 8-way partitioning, we set the area skew to 5%, that is, each partition can contain between 0.45^3n and 0.55^3n vertices, where n is the number of nodes in the circuit. The *VCycle* option was turned on for the hMetis. For PMP, we set α parameter value to 0.87 (cf. Section IV(C).) Table 2 reports the results of comparing PMP with hMetis and TPA for the eight-way partitioning problem. All results in this table represent the average of 20 different runs for each partitioning algorithm. The cutsize, Λ_G, is calculated as the sum of costs of each net that is cut. In turn, the cost of a cut net is k-1 if that net has pins in k parts [18].

Based on the data in Table 2, we conclude that, in terms of the circuit delay, PMP outperforms hMetis and TPA by an average of 14.6% and 3.8%, respectively. In addition, we can see that compared to hMetis, PMP increases the cutsize by an average of 1.2%, while, on average, PMP obtained 3.7% lower cutsize compared to TPA. PMP runtime is on average three times higher than that of hMetis. Notice that PMP is on average 18.8% faster than TPA. PMP reduced the circuit delay by 14.6% with a negligible increase in the cutsize compared to hMetis. As a result, our experiments show that the move-based bipartitioning algorithm works remarkably with the gain function of the preferred signal direction, so that it optimizes circuit delay very efficiently at the minimum cost of cutsize.

VI. Conclusions

In this paper, we presented a new performance-driven multilevel partitioning algorithm. Our main contribution is a new and efficient timing gain function formulation for the move-based bipartitioning algorithm. In addition, we proposed a simple but efficient timing-aware clustering algorithm that uses the maximum logic depth and/or the hop-count of an edge as a tiebreaker. These new methods fit very nicely within the general framework of a multilevel partitioning algorithm. Our approach does not alter netlist causing area increase as previous approaches do. Consequently, we can reduce a circuit delay very efficiently with at the minimum cost of cutsize.

Reference

[1] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *ACM/IEEE Design Automation Conference*, 526-529, 1997.

[2] C. J. Alpert, J. H. Huang and A. B. Kahng, Multilevel Circuit Partitioning. In *ACM/IEEE Design Automation Conference*, 530-533, 1997.

[3] J. Cong, H. P. Hi, S. K. Lim, T. Shibuya and D. Xu, Large Scale Circuit Partitioning with Loose/Stable Net Removal and Signal Flow Based Clustering. In *IEEE International Conference on Computer-Aided Design*, 441-446, 1997.

[4] J. Cong and S.K. Lim, Performance Driven Multiway Partitioning. In *ACM/IEEE Asia South Pacific Design Automation Conference*, 441-446, 2000.

[5] A. B. Kahng and X. Xu, Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-driven bipartitioning. In *ACM/IEEE International Symposium Physical Design*, 81-86, 2003.

[6] J. Cong, S. Lim and C. Wu, Performance Driven Multi-level and Multiway Partitioning with Retiming. In *ACM/IEEE Design Automation Conference*, 274-279, 2000.

[7] J. Cong and C. Wu, Global Clustering-Based Performance Driven Circuit Partitioning. In *ACM/IEEE International Symposium Physical Design*, 149-154, 2002.

[8] J. Cong, H. Li and C. Wu, Simultaneous Circuit Partitioing/Clusterng with Retiming for Performance Optimization. In *ACM/IEEE Design Automation Conference*, 460-465, 1999.

[9] C. Ababei, S. Navaratnasothie, K. Bazargan and G. Karypis, Multi-objective Circuit Partitioning for Cutsize and Path-Based Delay Minimization. In *IEEE International Conference on Computer-Aided Design*, 181-185, 2002.

[10] C. Ababei and K. Bazargan, Statistical Timing Driven Partitioning for VLSI Circuits. In *Design Automation and Test in Europe*, 1109, 2002.

[11] Shihliang Ou and Massoud Pedram, Timing-driven bipartitioning with replication using iterative quadratic programming. In *ACM/IEEE Asia South Pacific Design Automation Conference*, 105-108, 1999.

[12] C. Fiduccia and R. Mattheyses, A Linear Time Heuristic for Improving Network Partitions. In *ACM/IEEE Design Automation Conference*, 175-181, 1988.

[13] A. E. Caldwell, A. B. Kahng, I. L. Markov, Hypergraph Partitioning with Fixed Vertices. In *ACM/IEEE Design Automation Conference*, 355-359, 1999.

[14] G. Karypis and V. Kumar, hMetis 1.5: A Hypergraph Partitioning Package. *Technical report, Department of Computer Science, Univ. of Minnesota*, 1998. Available on the WWW at URL *http:///www.cs.umn.edu/~metis*.

[15] ISCAS89 benchmarks. At http://www.cbl.ncsu.edu

[16] ITC benchmarks. At http://www.cad.polito.it/tools/9.html

[17] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, A. Sangiovanni-Vincentelli, 'SIS: A System for Sequential Circuit Synthesis', Technical Report UCB/ERL M92/41, Univ. of California, Berkeley, May 1992.

[18] L. A. Sanchis, Multiple-way network partitioning. In IEEE Trans. Computer, vol. 38, pp.62-81, Jan. 1989.

Table 2. Comparison of PMP with hMetis and TPA on 8-way partitioning (Λ_G denotes cutsize, Φ_G denotes the delay).

Benchmark	hMetis			TPA			PMP		
	Φ_G	Λ_G	cpu (s)	Φ_G	Λ_G	cpu (s)	Φ_G	Λ_G	cpu (s)
cordic	15.0	305.3	0.2	10.7	301.4	0.6	9.9	278.9	0.5
misex3	72.5	760.7	0.2	69.4	763.1	0.8	57.9	764.2	0.6
x3	32.4	368.9	0.2	28.6	386.8	0.5	28.4	342.4	0.5
c6288	52.8	199.2	0.3	52.6	202.6	1.1	51.9	214.3	0.9
s15850	102.4	284.1	0.6	100.1	277.4	1.9	104.9	275.2	1.7
frisc	157.2	1233.3	0.7	159.4	1238.9	2.3	146.2	1183.2	2.1
elliptic	372.1	905.1	0.6	365.9	950.8	2.5	388.9	812.2	2.2
ex1010	410.9	1053	1.2	406.8	1162.7	5.5	399.3	1086.8	5.2
pdc	356.4	1887	0.8	317.3	1931.5	4.2	265.2	1883.2	3.7
s38417	146.1	312.9	0.9	145.4	419.2	4.8	147.6	415.8	3.8
b21s	357.2	806.1	2.8	289.7	888.4	12.8	283.7	900.5	10.3
b22s	472.8	922.9	5.2	363.1	978	23.35	372.3	957.4	15.2
b17s	419.7	1288.7	10.7	397.9	1307.1	32.3	392.3	1316.6	29.4
Average	**14.6%**	**-1.2%**	**-68.6%**	**3.8%**	**3.7%**	**18.8%**	**1**	**1**	**1**