Global Clustering-Based Performance-Driven Circuit Partitioning

Jason Cong
University of California at Los Angeles
Los Angeles, CA 90095
cong@cs.ucla.edu

Chang Wu
Aplus Design Technologies, Inc.
Los Angeles, CA 90024
changwu@aplus-dt.com

Abstract

In this paper, we propose a new global clustering based multi-level partitioning algorithm for performance optimization. Our algorithm computes a delay minimal K-way partition first, then gradually reduces the cutsize while keeping the circuit delay by de-clustering and refinement. Our test results on a set of MCNC sequential examples show that we can reduce the delay by 30%, while increasing the cutsize by 28% on average, when compared with hMetis [5]. Our algorithm also consistently outperforms other state-of-the-art partitioning algorithms [2, 5, 3] on circuit delay with reasonable cost on the cutsize.

1. INTRODUCTION

Recent trend in deep submicron VLSI technology indicates that circuit delay is increasingly dominated by the interconnect delay. National Technology Roadmap for Semiconductor predicts that 80% or more circuit delay will be directly linked to interconnect in deep submicron designs [1]. Furthermore, as the feature size (minimum wire width) continus to shrink, the circuit size becomes larger and larger and increasingly difficult to handle by most current synthesis and layout tools. Sylvester and Keutzer [13] proposed to partition a large circuit into small blocks so that the local intra-block delay can be small compared to global inter-block delay. In general, circuit partitioning can be used for either global planning or divide-and-conquer purposes as pointed out in [2, 3]. For high performance designs, circuit partitioning with consideration of performance optimization becomes indispensable.

In this paper, we study the circuit partitioning for performance optimization problem with consideration of retiming. Retiming is a sequential optimization technique, usually performed at logic domain. By combining partitioning and retiming, we can make a better plan for large designs with the mutual benefit of both logic and physical optimization capabilities.

There are a number of works on performance-driven partitioning or clustering [9, 8, 6, 10, 14, 11, 2, 3]. The partitioning based approaches can have better control on the cutsize, but no guarantee on the performance. The min-delay clustering based approaches have stronger capability in optimizing the performance, but tend to increase cutsize significantly. Furthermore, node duplication is assumed for those clustering based algorithms to guarantee the best performance, which in practice may significantly increase the circuit size.

For cutsize minimization, hMetis [5] significantly improves the solution quality based on a very efficient multi-level optimization method. It performs a sequence of coarsening and uncoarsening steps followed by FM [4] based refinement. Compared with other algorithms, this algorithm can consistently achieve a much smaller cutsize. However, its coarsening step only explores very local connectivity information. Thus, it cannot guarantee to produce small delay in general. Recently, Cong et al. [2] proposed a performance-driven clustering/partitioning with retiming algorithm. Given an area bound for each cluster, PRIME can compute a quasi-optimal solution if node duplication is allowed. For circuit partitioning without node duplication, a heuristic version of PRIME can still achieve the smallest delay compared with other partitioning algorithms followed by separate retiming [2]. But PRIME suffers a serious problem of producing huge cutsize. Later on, Cong et al. [3] proposed to combine PRIME and the multi-level cutsize minimization technique together for both performance and cutsize optimization. Their algorithm, named HPM [3], first performs PRIME to form small clusters for delay minimization, then a multi-level edge-separability based clustering algorithm and a multi-way partitioning algorithm for cutsize minimization. Their results show that HPM [3] can achieve a reasonable balance in performance and cutsize compared with hMethis [5] and PRIME [2]. However, in HPM [3], the performance-driven clustering and the multi-way partitioning are separated into two steps. To achieve a balance on performance and cutsize, HPM [3] must sacrifice the full performance optimization capability of PRIME by giving a very small area bound [1] to PRIME for larger freedom on cutsize reduction in later steps. As a result, HPM only has limited performance optimization capability, and in some cases can even produce results with larger delay than that by hMetis.

The multi-level optimization scheme of hMetis [5] can be

ISPD'02 April 7-10, 2002, San Diego, California, USA.

[1] Instead of setting the area bound to be the full block size for best possible delay, the authors of [3] chose a constant area bound of 10 for running PRIME [3] for best area delay tradeoff based on their experimental results.

depicted in Figure 1. It performs gradual coarsening followed by uncoarsening and refinement. For cutsize minimization, the hyperedge based coarsening with consideration of local connectivity may be good enough. For performance optimization or other objectives needing the information of the entire circuit, gradual coarsening is hard to predicting the impact to the final circuit performance. Most of all, any errors made at lower levels will propagate to the top level and be very difficult to correct until the uncoarsening gets back to the bottom levels. As a result, any bad coarsening at lower levels will very likely entrap the algorithm to a local optimal solution with inferior quality.

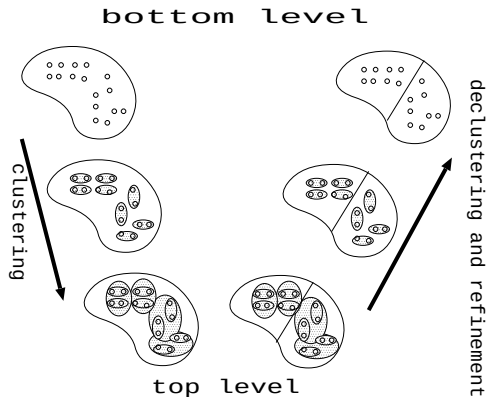

Figure 1: The V-Cycle Multi-Level Partitioning Method in hMetis.

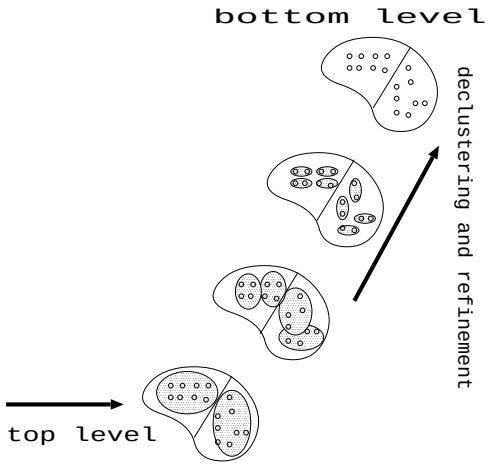

Figure 2: Top-Down Multi-Level Partitioning Method.

In this paper, we propose a new multi-level optimization scheme for performance-driven circuit partitioning based on global clustering. We overcome the problem on gradual coarsening by directly starting from the top level partitioning as shown in Figure 2. We perform the global clustering PRIME algorithm [2] to get an initial partitioning solution with minimal circuit delay. Then, we perform multi-level de-clustering and refinement to gradually reduce the cutsize while preserving the minimal circuit delay. The objective of the refinement is changed to cutsize minimization, rather than the more difficult delay minimization. The cutsize minimization problem can be handled easily by many existing partitioning algorithms like FM [4]. Our main contribution is a new global clustering based multi-level optimization scheme starting directly from the top level. For performance optimization, we think our new multi-level optimization scheme is more suitable than the coarsening and uncoarsening method in [5]. We believe our new scheme can be applied for performance optimization at various stages of logic synthesis and physical design.

We have tested our algorithm on a set of MCNC examples. Our results show that our algorithm consistently outperforms other state-of-the-art partitioning algorithms with 15% to 30% smaller delay for either 16-way partitioning or bi-partitioning. Our cutsize is similar to that by HPM [3], but 28% larger than that by hMetis [5].

The remainder of the paper is organized as follows. Section 2 presents the problem formulation and preliminaries. Section 3 presents the details of our algorithm. The experimental results are presented in Section 4. Discussions and conclusions are given in Section 5.

2. PROBLEM FORMULATION AND PRELIMINARIES

The partitioning problem is to decompose a given circuit into K blocks for a given K with balanced area. Retiming is a well-known sequential circuit optimization technique for delay optimization by moving flipflops without changing the circuit behavior. Retiming is optimally solved first by Leiserson and Saxe [7]. For performance-driven circuit partitioning and retiming, we represent a sequential circuit by the retiming hypergraph which is a modification of the retiming graph defined in [7]. A retiming hypergraph is a directed graph $G(V, E, W, H)$, where V is the set of nodes representing gates in the circuits, E is the set of edges representing connections between nodes, $W = \{w(e) \mid \forall\ e \in E\}$ is the set of edge weights representing the numbers of flipflops on the edges, and H is the set of hyperedges. A hyperedge is a set of edges driven by the same node. PI and PO denote the sets of primary inputs and primary outputs of the circuit. A retiming solution is denoted as $R(V) = \{r(v) \mid \forall\ v \in V\}$, where $r(v)$ is an integer representing the number of flipflops moved from v's output edges to input edges.

PROBLEM 1. *For a sequential circuit $G(V, E, W, H)$, a balanced duplication-free K-way partitioning and retiming solution is represented as $P = \{P_1, P_2, .., P_K\}$ and $R(V) = \{r(v) \mid \forall\ v \in V\}$ satisfying the following conditions:*

1. $P_i \bigcap P_j = \emptyset$ *for* $i \neq j$ *and* $P_1 \bigcup .. \bigcup P_K = V$,
2. $\alpha_i \leq \mid P_i \mid \leq \beta_i$ *for given* α_i *and* β_i, $1 \leq i \leq K$,
3. $w^r(e(u,v)) = w(e(u,v)) + r(v) - r(u) \geq 0$,
4. $r(v) = 0$ *for all* PI, PO *nodes* v.

The cutsize of a partitioning solution is the number of hyperedges spanning more than one block. As in [2, 3], the *general delay model* is used in this paper, which assumes that each gate v has a delay of d_v, intra-block delay (local interconnect delay within each block) of d and inter-block delay (interconnect delay between blocks) of D, where $d < D$. The size of each gate v is a_v. We ignore the setup and hold times, as well as the size of flipflops (FFs) as assumed in

previous works on retiming [7, 2, 3].[2]

The circuit delay is measured as the longest combinational path delay from a PI or an FF output to a PO or an FF input. Our objective is to perform a K-way balanced partitioning with retiming for the minimum delay, while reducing the cutsize as much as possible. In the remainder of the paper, we use both circuit delay and clock period interchangeably for the same meaning without further notice.

3. THE MULTI-LEVEL PERFORMANCE-DRIVEN PARTITIONING WITH RETIMING ALGORITHM

3.1 Overview

In this section, we present the details of our algorithm named MLPR. For a circuit with total area of A and a given constant K for K-way partitioning, we first perform a performance-driven clustering with area bound of $\frac{A}{K}$ to achieve the minimal possible clock period for K-way partitioning and retiming using the PRIME algorithm [2] as described in Section 3.2. Then, we partition those clusters into K blocks to get the top level partitioning solution. The performance-driven clustering problem can be solved by PRIME [2], while the cutsize-driven partitioning problem can be solved with hMetis [5] or FM [4]. Since the cutsize of this partitioning solution is still quite large, we perform subsequent steps of de-clustering and refinement for cutsize minimization, while preserving the circuit delay. In the following, we first review the performance-driven clustering with retiming algorithm in [2]. Then, we present details on initial partition generation at the top level, performance-driven de-clustering and performance-bounded cutsize driven refinement.

3.2 Review of the Performance-Driven Clustering

The performance-driven clustering is to form a number of clusters for a circuit under a given area bound. Under the general delay model with node delay d_v, intra-block delay d and inter-block delay D, our objective is to minimize the circuit delay after optimal retiming. To achieve the minimal clock period, PRIME [2] performs a binary search in the range $[lb, ub]$, where lb is a lower-bound on the clock period computed by assuming that every edge has delay d and ub is an upper-bound on the clock period computed by assuming that every edge has delay D. For a retiming hypergraph $G(V, E, W, H)$ of a circuit and a target clock period ϕ, the *edge length*, denoted $length(e)$, of an edge $e(u, v)$ is defined to be $-\phi \cdot w(e) + d(e) + d_v$, where $w(e)$ is the edge weight and $d(e)$ is the edge delay. For a given clustering solution, each edge has a well-defined edge length. The l-value $l(v)$ of a node v is defined to be the maximum path length from PIs to v.

It was proven in [11] that:

THEOREM 1. *In a clustered circuit C of a sequential circuit with a target clock period ϕ, if there is a PO whose l-value is greater than ϕ, the clustered circuit cannot be retimed to clock period of ϕ or less. If, on the other hand, the l-values of all POs are less than or equal to ϕ, the clustered circuit can be retimed to a clock period less than $\phi + D$.*

Accordingly, let $l^{opt}(v)$ be the minimum $l(v)$ among all possible clustering solutions, one can check if there exists one clustering solution with a retimed clock period of no more than ϕ by computing $l^{opt}(v)$ for all PO nodes v. Since our problem is circuit partitioning without node duplication, we refer PRIME to the duplication-free heuristic proposed in [2] in the remainder of this paper.

3.3 Top Level Clustering and Partitioning

For a circuit with an area of A and K-way partitioning, we perform the performance-driven clustering algorithm PRIME [2] with an area bound of $\frac{A}{K}$. Since PRIME only groups critical nodes together, i.e., groups nodes if separating them will increase the final delay, PRIME will generate more than K clusters. In practice, most have an area far less than $\frac{A}{K}$, while only a few have an area closing to $\frac{A}{K}$. We further pack those clusters into K partitions with area bounded by $\frac{A}{K}$. Since packing will only reduce edge delay, the minimum clock period computed by PRIME can be preserved in any K-way partitioning solution of those clusters. It is not difficult to prove that:

THEOREM 2. *For a given general delay model with given node delay, intra-block delay, inter-block delay, and a given clustering solution of a circuit, let ϕ be the circuit delay. For any partitioning solution of the clusters, its circuit delay will be less than or equal to ϕ.*

Clearly, with a delay optimal clustering solution, we can focus on cutsize minimization only when computing a K-way partitioning solution of the clusters.

In MLPR, we use hMetis [5] to pack those clusters generated by PRIME [2] into K area-balanced partitions for smaller cutsize of the initial partitioning at the top level.

3.4 Performance-Driven De-clustering and Refinement

The initial partitioning solution generated by PRIME [2] and hMetis [5] at the top level defines the best delay we can achieve. However, the cutsize is usually much larger than what is achievable by hMetis [5]. This is because some of the large clusters generated by PRIME limit the solution space for cutsize reduction. To further reduce the cutsize without increasing the circuit delay, we perform gradual de-clustering and refinement until we reach the bottom level, where each cluster is a single node.

De-clustering will form smaller clusters so that refinement can have greater freedom in moving clusters (or nodes) for cutsize reduction. De-clustering in MLPR is actually a clustering process with a smaller area bound on each cluster. The objective is performance optimization, i.e., it tries to group together nodes in critical paths to reduce the circuit delay. According to Theorem 2, any partitioning solution of the de-clustered netlist will only have the same or smaller delay. Our performance-driven de-clustering provides a guarantee on the circuit delay so that the follow-on step of refinement can be simplified to consider cutsize only.

[2]We ignore the size of FFs is because it is difficult to predict which partition an FF will be in before retiming. For FPGA, in general this will not cause a problem because they are register-rich. For example, both Xilinx Virtex and Altera Flex FPGAs has one FF to use for each 4-LUT. For ASIC, if retiming causes area unbalance problem, a postprocessing of rebalancing area is needed, or we can disable retiming and consider FF area during partitioning.

A big difference between our de-clustering procedure and the uncoarsening procedure in hMetis [5] is that our de-clustering is based on the current partitioning solution, while the uncoarsening solution is determined a priori in the coarsening stage. The major drawback of the uncoarsening approach is that it does not use any information in the current partitioning solution and does not know where the critical paths are.

In MLPR, we perform a partition-bounded clustering for de-clustering. For the partitioning solution defined at a higher level, a partition-bounded clustering solution is a clustering solution in which two nodes are in one cluster only if they are in the same partition. In other words, any cluster will not be moved across the partition boundary. The reason is that by preserving the higher level partition boundary, we guarantee that the higher level partitioning solution is also a partitioning solution of the de-clustered netlist at the current level. In theory, we can guarantee the optimal solution at the current level is always better than the higher level solution. Notice that the refinement procedure can still move clusters across the partition boundary.

Our partition-bounded performance-driven de-clustering is performed as follows. We first mark all cut edges across different partitions. Then, we compute an area bound which is smaller than the area bound used at the higher level. (At the top-most level, the area bound of clusters is the maximum size of each partition.) We then call PRIME to do performance-driven clustering with the following modification. When PRIME wants to pack a fanin node u of node v into the cluster of v based on the performance requirement, we check if the edge from u to v is cut. If the edge is cut, we do not allow u to be packed into the cluster of v even though the delay will increase. We call this algorithm the modified PRIME algorithm.

In the refinement step, we perform FM algorithm [4] for cutsize reduction. At the end of refinement, we perform timing analysis to compare the current solution with the previous solution at a higher level and decide whether to keep the refined solution or roll back to the solution before refinement. For better control on delay, we use the multiple rollback point method proposed in [3]. This method saves several local optimal solutions in one pass of refinement, evaluates the delay of the partition corresponding to each saved solutions and chooses one with the smallest cutsize while preserving the delay of the higher level partition. Because only a few points are considered, we can do timing analysis on the entire circuit without increasing the runtime too much. If none of the points we computed can preserve the delay of the higher level partition, we can pick a solution based on a tradeoff on delay and cutsize.

When we reach the bottom level, every cluster is just a single node. We perform one more refinement on the flat netlist without any clustering constraints. Finally, we perform an optimal retiming on the partitioning solution to get the minimum clock period. Notice that during the entire partitioning, de-clustering and refinement stage, our timing analysis always considers retiming by computing the node labels $l^{opt}(v)$ as defined in Section 3.2. As a result, our partitioning solution already determines the retiming solution which can be computed directly from node labels as [2].

The pseudo code of our MLPR algorithm is shown in Figure 3.

```
MLPR(G(V, E, W, H), K)
1   compute the area A of the entire circuit
    compute the upper and lower bounds on each block
2   for area bound of A/K, compute the minimal clock period φ
      with the PRIME algorithm [2]
3   for φ, construct a (duplication-free) clustering
      solution with PRIME [2]
4   initial partitioning P_0: merge those clusters into K blocks
5   for level i from 1 to h
6     set area bound A_i to be A/(K·2^i)
7     call modified PRIME with A_i under P_{i-1} to
        get a clustering solution C_i
8     refining P_{i-1} to get P_i by moving clusters in C_i
9     stop if A_i = 1
10  return the final solution
```

Figure 3: Multi-level Performance-Driven K-way Partitioning. $G(V, E, W, H)$ is the retiming graph of the original circuit, where V is the node set, E is the edge set, W is the edge weight set and H is the hyperedge set. K is a given constant for K-way partitioning.

4. EXPERIMENTAL RESULTS

We have implemented our algorithm, named MLPR, in C language and incorporated into the SIS [12] package. The experiments were run on a dual Intel PentiumIII 400Mhz PC with 1GB memory. We obtained hMetis V1.5.3, PRIME and HPM from their authors for evaluation. We used 12 MCNC benchmarks. Tech_decomp is performed to decompose the original circuits into 2-bounded circuits, i.e., each gate has only two inputs. The size of the circuits ranges from 500 to 30,000 2-input gates.

In our test, we assume each two-input gate has an area of 1 and a delay of 1. The local interconnect delay within a partition is 0 and the inter-partition delay is 5 as is the one used in [3]. We compared our algorithm with the other algorithms for both 16-way partitioning (as tested in [3]) and bi-partitioning. Since the HPM binary code we obtained can only run for 16-way partitioning, we could not compare with HPM for bi-partitioning. For PRIME, we used the duplication-free heuristic in [2]. For both 16-way and bi-partitioning test, we set the area skew to be 5%. For bi-partitioning, the partition size can be 45% to 55% of the circuit size. For 16-way partitioning, the area bound on each partition is within the range of 95% to 105% of $\frac{A}{16}$, where A is the circuit size. The 16-way partitioning results are shown in Table 1 and the bi-partitioning results are shown in Table 2. For 16-way and bi-partitioning, we run K-way hMetis, denoted khMetis, and hMetis, respectively. In either case, we perform 20 runs with the VCycle option turned on. We then read back the partitioning solutions and perform an optimal retiming. For PRIME, HPM and our MLPR, retiming capability is included in the package.

In Table 1, we can see that MLPR consistently outperforms all other algorithms in terms of the clock period. We can get even smaller clock periods than the (duplication-free heuristic) PRIME algorithm. Our results show that compared with MLPR, PRIME generates results with 3 times larger cutsize and 15% larger clock period. The khMetis algorithm can generate results with 28% smaller cutsize, but a 30% larger clock period on average. The HPM algorithm can generate results with a 1% smaller cutsize and

16-way Partitioning Result										
			PRIME [2]		khMetis [5]		MLPR		HPM [3]	
circuit	#nodes	ϕ_{lb}	cutsize	ϕ	cutsize	ϕ	cutsize	ϕ	cutsize	ϕ
s1423	530	57	162	67	61	120	144	68	88	104
s838.1	376	19	97	29	61	43	85	28	77	40
s5378	1578	13	354	27	157	35	171	25	222	31
s9234.1	1368	21	248	30	110	30	175	29	187	31
s1196	509	24	158	43	96	53	138	43		
s13207.1	3376	32	408	45	126	55	306	41	251	51
s15850.1	4016	37	709	65	218	57	405	51	296	67
s38417	9897	27	1129	39	171	38	186	30	243	38
s38584.1	13551	29	1874	34	278	39	396	29		
sbc	754	16	217	32	128	40	163	28	171	35
bigkey	9058	7	773	17	41	14	49	12	46	11
clma	31020	57	3660	97	388	76	366	76	415	80
average		28.3	815.8	43.8	152.9	50.0	215.3	38.3		
			299%	15%	-28%	30%	1	1	-1%	23%
		1		66%		88%		44%	78%	

Table 1: Comparison of MLPR with PRIME, khMetis and HPM on 16-way partitioning. Area skew is 5%, node delay is 1, intra-block delay is 0, inter-block delay is 5. #nodes is the number of nodes in each circuit. ϕ_{lb} is a lower bound on the clock period for each example computed by PRIME by allowing node duplication. Empty entry means that HPM could not generate a solution for that example. The comparison is based on the average improvement of each example when compared with MLPR or ϕ_{lb}.

2-way Partitioning Result								
			PRIME [2]		hMetis [5]		MLPR	
circuit	#nodes	ϕ_{lb}	cutsize	ϕ	cutsize	ϕ	cutsize	ϕ
s1423	530	50	90	61	12	55	12	55
s838.1	376	15	86	20	5	20	37	20
s5378	1578	11	234	20	42	20	48	17
s9234.1	1368	20	112	22	41	30	34	20
s1196	509	19	114	29	32	38	84	27
s13207.1	3376	30	173	30	33	36	38	30
s15850.1	4016	34	369	38	40	41	51	34
s38417	9897	27	156	27	27	27	27	27
s38584.1	13551	29	694	34	33	32	34	29
sbc	754	14	142	29	24	24	35	18
bigkey	9058	7	657	19	8	12	8	12
clma	31020	52	2648	78	39	54	39	54
average		23.5	489.5	31.5	29.5	30.4	39.5	26.2
			1662%	20%	-17%	16%	1	1
		1		46%		39%		20%

Table 2: Comparison of MLPR with PRIME and hMetis on bi-partitioning. Area skew is 5%, node delay is 1, intra-block delay is 0, inter-block delay is 5. #nodes is the number of nodes in each circuit. ϕ_{lb} is a lower bound on the clock period for each example computed by PRIME by allowing node duplication. The comparison is based on the average improvement for each example when compared with MLPR or ϕ_{lb}.

23% larger clock period. As to the runtime, khMetis consistently outperforms other algorithms significantly because it does not consider any timing issues. For the largest design *clma*, khMetis needs only 113 seconds, while PRIME takes 1044 seconds and MLPR takes 4808 seconds. We did not measure HPM's runtime as it was run on a Sun Solaris machine.[3] Since HPM uses a fixed and very small cluster size for its clustering step, its runtime is usually much shorter than PRIME and MLPR which use $\frac{A}{K}$ as the area bound for clustering. For fixed K, performance-driven clustering algorithm PRIME[2] works in quadratic order of the circuit size. MLPR's runtime will be even longer. Currently, we are looking for faster clustering algorithms to speed up MLPR. Nevertheless, MLPR demonstrates the significant advantage on performance optimization with consideration of cutsize over the state-of-the-art algorithms.

To demonstrate how far away our delays are from the optimal ones, we compare the delay of each algorithm with the lower bound on the minimum clock period by the labeling

[3]The binary code we obtained could only run on Solaris machine.

procedure of the PRIME algorithm,[4] our MLPR algorithm can produce results with delay of 44% larger than the lower-bounds. Notice that the lower-bounds are computed under the assumption that node duplication is allowed. If we compare our results with the optimal *duplication-free* partitioning solutions, the delay increase is most likely to be even smaller.

Similar results hold for bi-partitioning as shown in Table 2.

5. DISCUSSIONS AND FUTURE WORK

In this paper, we present a performance-driven circuit partitioning with retiming algorithm. Our main contribution is a new global clustering-based multi-level optimization scheme. Our results show that our algorithm consistently outperforms other state-of-the-art partitioning algorithms for circuit delay minimization. Retiming is considered seamlessly during partitioning to achieve the best performance. We believe our new multi-level optimization scheme can be adapted to other problems in various stages in logic synthesis and physical delay for performance optimization.

A key step in our algorithm is the performance-driven global clustering. In MLPR, we use the PRIME algorithm to compute clustering and de-clustering solutions in each level. The PRIME algorithm runs in the order of $O(A{\cdot}n{\cdot}log^2 D)$ for a given area bound A, circuit size n and inter-block delay D [2]. In MLPR, we need to set $A = \frac{n}{K}$ which makes PRIME too slow for large designs and small K. To reduce the runtime, we may need to either reduce the area bound or speed up the PRIME algorithm. Reducing the area bound may increase the clock period of the initial partitioning solution at the top level, which may propagate to the final solution. We think speeding up PRIME is a better approach.

In the future, we also want to extend our algorithm to performance-driven floorplan or placement problems to consider node locations.

6. ACKNOWLEDGEMENTS

The authors want to thank Prof. S. Lim for providing the HPM binary code for the test.

7. REFERENCES

[1] Semiconductor Industry Association, National Technology Roadmap for Semiconductors, 1997.

[2] J. Cong, H. Li, and C. Wu, Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization, In *Proc. ACM/IEEE Design Automation Conference*, pages 460–465, 1999.

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

[4] C. Fiduccia and R. Matheyses, A Linear-Time Heuristic for Improving Network Partitions, In *ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

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

[6] E. L. Lawler, K. N. Levitt, and J. Turner, Module Clustering to Minimize Delay in Digital Networks, *IEEE Trans. on Computers*, 18:47–57, 1969.

[7] C. E. Leiserson and J. B. Saxe, Retiming Synchronous Circuitry, *Algorithmica*, 6:5–35, 1991.

[8] L. Liu, M. Kuo, C. K. Cheng, and T. C. Hu, Performance-Driven Partitioning using a Replication Graph Approach, In *Prod. 32th ACM/IEEE Design Automation Conference*, pages 206–210, 1995.

[9] L. Liu, M. Shih, N. Chou, C. K. Cheng, and W. Ku, Performance-Driven Partitioning Using Retiming and Replication, In *IEEE International Conference on CAD*, pages 296–299, 1993.

[10] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, On Clustering for Minimum Delay/Area, In *IEEE International Conference on CAD*, pages 6–9, 1991.

[11] P. Pan, A. K. Karandikar, and C. L. Liu, Optimal Clock Period Clustering for Sequential Circuits with Retiming, *IEEE Trans. on Computer-Aided Design of Integrated Circuits And Systems*, 17(6):489–498, 1998.

[12] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, SIS: A System for Sequential Circuit Synthesis, Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41, 1992.

[13] D. Sylvester and K. Keutzer, Getting to the Bottom of Deep Submicrion, In *IEEE International Conference on CAD*, pages 203–211, 1998.

[14] H. Yang and D. F. Wong, Circuit Clustering for Delay Minimization under Area and Pin Constraints, In *ED&TC*, pages 65–70, 1995.

[4] The lower bound computed by PRIME is the minimum clock period under node duplication. We refer readers to [11, 2] for the details.