Performance-driven Global Placement via Adaptive Network Characterization

Mongkol Ekpanyapong and Sung Kyu Lim
School of Electrical and Computer Engineering,
Georgia Institute of Technology
{pop,limsk}@ece.gatech.edu

Abstract - Delay minimization continues to be an important objective in the design of high-performance computing system. In this paper, we present an effective methodology to guide the delay optimization process of the mincut-based global placement via adaptive sequential network characterization. The contribution of this work is the development of a fully automated approach to determine critical parameters related to performance-driven multi-level partitioning-based global placement with retiming. We validate our approach by incorporating this adaptive method into a state-of-the-art global placer GEO. Our A-GEO, the adaptive version of GEO, achieves 67% maximum and 22% average delay improvement over GEO.

1. INTRODUCTION

With tremendously increasing demand in high performance computing, circuit performance improvement during physical design becomes indispensable. During physical planning, gate location is identified and hence can be used to accurately calculate wire delay. Circuit optimization then can employ this knowledge and gain superior performance over same optimizations without such information. One optimization that can employ this advantage is retiming [9]. Retiming is a logic optimization technique, which shifts the position of flip-flops (FFs) for delay minimization or FFs reduction [9]. Recently, retiming has become more attractive in physical design where wire delay is more essential in the context of deeper submicron technology [15,6]. Exploiting geometric information enables us to further enhance retiming techniques with placement—since location information is available, wire delay calculation becomes more accurate. Retiming based placement can be classified into two approaches: iterative approach and simultaneous approach. The iterative approach [16,10,11] first performs placement or floorplanning, followed by retiming. The alternative approach [4,14,5] simultaneously performs placement or floorplanning with retiming by incorporating retiming information during placement. In [13,4,5], the authors suggest that the latter approach is better than the former with respect to retiming delay improvement.

In [4], a state-of-the-art approach for mincut-based placement with retiming, so called GEO, was proposed. The concepts of Sequential Arrival Time (SAT) [13] and Sequential Required Time (SRT) were adopted here. Then the sequential slack value, which is used to identify critical gates/clusters after retiming, is computed as the difference between SRT and SAT. Cong et al. [5] extended [4] by generalizing the model to handle the gates/clusters with multiple outputs. There are two parameters in delay weight based algorithms such as [4,5] that play a critical role: ε determines the size of the sub-network so called ε-network that contains timing critical nodes, whereas α determines the amount of additional weights added to the nets in ε-network. Both [4] and [5] fix ε and α during the entire pass of their algorithm. In [15,8], the authors propose a way to identify critical edges using criticality distribution. However with their method, one has to iteratively search until they find a proper ε-network and net weight constant α. This is time consuming, and for some circuits, it is hard to achieve.

In this paper, we show that while weighted cutsize highly correlates with retiming delay, there is no guarantee that it will result in the best retiming delay among all runs. Then we propose a methodology to automatically update ε and α for more efficient delay optimization. In addition, we study the impact of clustering on delay optimization. We note that we lose accuracy of circuit retiming information since it is calculated based on each gate location. To alleviate this problem, we propose a way to adaptively decide when to perform clustering based on circuit information. The organization of this paper is as follows. Section 2 describes problem formulation. Our observations are discussed in section 3. Section 4 is devoted to our methodology. Section 5 presents our experimental results and final section presents our conclusion and future work.

2. PROBLEM FORMULATION

Given a sequential gate-level netlist $NL(C, N)$, where C is the set of cells representing gates and flip-flops, and N is the set of nets connecting the cells, the purpose of the Physical Planning with Retiming (PPR) problem is to assign cells in NL to a given m x n (=K) slots by preserving area constraints. Given a PPR solution $C\rightarrow B$, let $\omega(B)$ and $\phi(B)$ respectively denote the wirelength (= half-perimeter of the net bounding box) and retiming delay (to be defined later). The formal definitions of PPR problem is as follows:

PPR Problem The Physical Planning with Retiming problem has a solution P: $C\rightarrow B$, when each cell in C is assigned to a unique block. $B = \{B_1(x_1,y_1), B_2(x_2,y_2),..., B_K(x_K,y_K)\}$, where B denotes the set of blocks, and (x_i,y_i) represents the geometric locations of B_i, and area constraints $A(L,U)$, for $1 \le i \le K$. PPR solution has to satisfy the following condition: 1) $B_i \subset C$ and $L \le |B_i| \le U$ 2) $B_1 \cup B_2 \cup \ldots \cup B_k$=C 3) $B_i \cap B_j = \varnothing$. The primary objective of PPR is to minimize $\phi(B)$ and secondary objective is to minimize $\omega(B)$.

For the delay objective, we model NL using a directed graph $G = (V, E)$ where the vertex set V represents cells, and the directed edge set E represents the signal direction in NL. In the *geometric delay model*, each vertex v has delay $d(v)$ and each edge $e=(u,v)$ has delay $d(e)$. Let $s(e)$ denote the *cut-state* of e: $s(e)=1$ if e is cut, and $s(e)=0$ otherwise. In this paper, we assume $d(e)=m(e)\cdot s(e)$, where $m(e) = |x_u-x_v|+|y_u-y_v|$. The delay of a path p, denoted $d(p)$, is the sum of the delay of gates and edges along p. Then, the *normal delay* $\delta(B)$ of global placement solution B is computed as $\max_{p\in G}\{d(p(u,v))|u\in PI \text{ or } FF \ \& \ v\in PO \text{ or } FF\}$.

By employing the concept of a retiming graph [9], we model NL using a directed graph $R = (V, E_R)$, where the edge weight $w(e)$ of $e=(u,v)$ denotes the number of flip-flops between gate u and v. The path weight can be calculated by $w(p)=\sum_{e\in p} w(e)$. Let $w^r(e)$ denote edge weight after retiming r, i.e. number of flip-flops on the edge after retiming. Then, $w^r(p)=\sum_{e\in p} w^r(e)$. A circuit is retimed to a delay ϕ by a retiming r if the following conditions are satisfies; (i) $w^r(e) \geq 0$ for each e, (ii) $w^r(p) \geq 1$ for each path p such that $d(p) > \phi$. We define the edge length of $e=(u,v)$ as $l(e)=-\phi\cdot w(e)+d(v)+d(e)$, and the path length of p as $l(p)= \sum_{e\in p} l(e)$. The *sequential arrival time* of vertex v, denoted $l(v)$, is the maximum path length from PIs or FFs to v. If the sequential arrival time of all POs or FFs are less than or equal to ϕ, the target delay ϕ is called *feasible*. Let $q(e)=\phi\cdot w(e)-d(u)-d(e)$, is the required edge length of e. The required path length $q(p)= \sum_{e\in p} q(e)$. The *sequential required time* of vertex v, denote $q(v)$ is the minimum required path length from v to POs or FFs, when $q(\text{PO})$ or $q(\text{FF}) = \phi$. Then slack of v is given by $q(v)-l(v)$. Let $D_g=\max\{d(v)|v\in V\}$. Then, the *retiming delay* $\phi(B)$ of a placement solution B is the minimum feasible $\phi + D_g$.

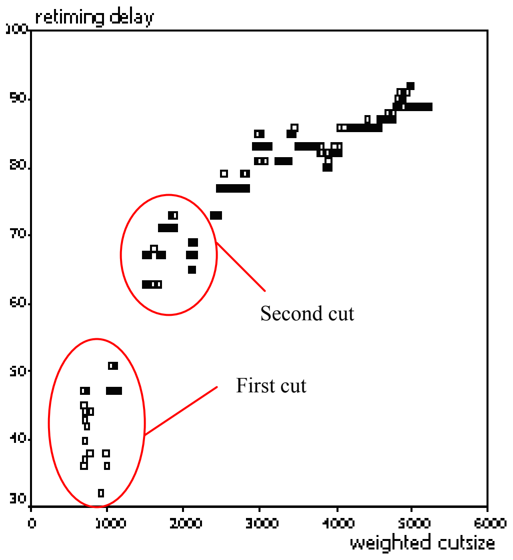

Figure 3.1 Correlations between retiming delay and weighted cutsize on circuit s1238.

3. OBSERVATIONS

In this section we provide our observation on retiming based on timing analysis (RTA) [9]. In addition, we utilize such information to dynamically guide the mincut-based global placement for performance improvement. We perform this study on benchmark circuits from ISCAS89 [18] and ITC99 [17] suites. Throughout the paper, our studies are based on 8×8 global placement with 5 runs, α = 20, T filter, and ε = top 5% nodes with small slacks unless explicitly specified (all to be explained later).

3.1. Correlation between Weighted Cutsize and Retiming

Most simultaneous placement or floorplanning with retiming [4,5] algorithms employ weighted cutsize during partitioning, where a retiming based timing analysis is performed for net weight computation. For example, in GEO [4], Net Weight = Cutsize Weight + α·Delay Weight is used. The α parameter determines how important delay weight is compared to cutsize. We studied the correlation between weighted cutsize and retiming delay and discovered that the correlation factor is about 0.9 on the average. This implies that weighted cutsize is highly related to retiming delay. In Figure 3.1 we plotted the retiming delay versus weighted cutsize for circuit s1238. When the first bipartition has a weighted cutsize of about 800, the variation on retiming delay is from 36 to 47. The cutsize of the next partition is between 1200 and 2,200, and the retiming delay varies from 62 to 75. This implies that using weighted cutsize alone might not be enough to achieve a high reduction in retiming delay even though the weighted cutsize is highly correlated with retiming delay.

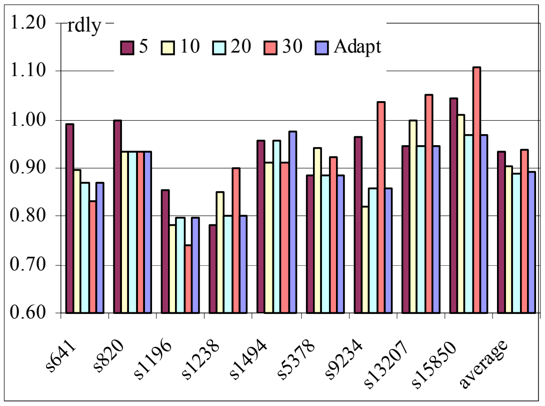

Figure 3.2 Impact on different α value

3.2. Delay Weight (α Parameter)

As shown in Figure 3.1, the highest impact partition based on mincut-based approach is the first partition. As we perform more partitions, the retiming delay improvement becomes less visible. Hence we assign higher α value and then reduce it as we perform more partitioning. First we find the best α as shown in Figure 3.2. The results are normalized to the case where α = 1. Among several fixed α values we tried, we found that α = 20 provides the best retiming delay on average. However, our new scheme, where we gradually decrease α value from 20 to 0 as we perform more partitions, outperforms other cases where we use fixed α value as evident from Figure 3.2.

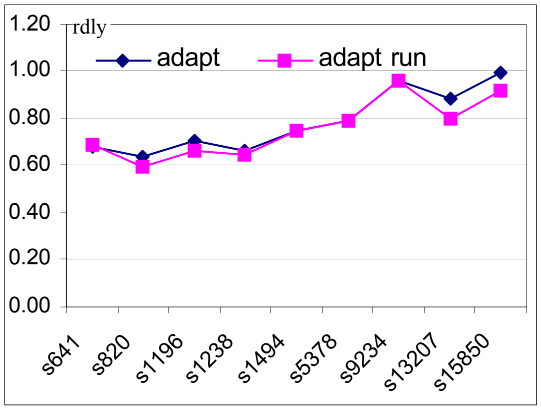

Figure 3.3 impact of adaptive number of run

3.3. Number of Runs

During each *run* in an iterative improvement based partitioning, we start from a random solution and perform refinement steps. After we finish all the runs, we pick the best run for the current partitioning. We observed that there exists more room for retiming delay improvement during early partitioning. Therefore, in our adaptive scheme we change the number of runs dynamically, where we perform more runs during the early partitioning and gradually decrease it as we perform more partitioning. For example, consider 8×8 global placement. Instead of using fixed 5 runs throughout the program, we use 20 runs in the first 3 levels (first 7 partitioning) and 3 runs for the rest. Hence we have 7×20+3×56, which is smaller than 63×5. The obvious advantage is on runtime saving. However, we also noted from Figure 3.3 that our adaptive scheme generated a slightly but consistently better retiming delay results. This is due to the fact that we now afford more number of runs for the early cuts in our adaptive scheme compared to the conventional scheme where the number of runs is fixed during the entire top-down partitioning process.

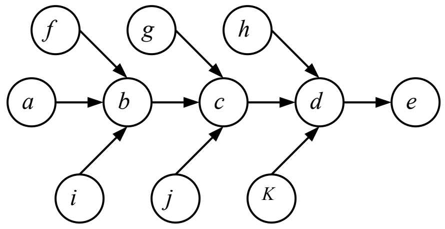

Figure 3.4 Example of net filtering

3.4. Nets Filtering and Cells Selection (ε Parameter)

During circuit partitioning, a hypergraph model is employed to represent the netlist. In [4], the net weight is assigned based on criticality among cells/clusters. The equation used in [4] to assign net weight is as follows:

$$dwgt(n) = 1 - \frac{\min\{slack(v) \mid v \in n\}}{\max\{slack(w) \mid w \in NL\}}$$

However we note from the above equation that non-critical cells might be included in the weight computation as illustrated in Figure 3.4. Suppose cells *a* through *e* are on the critical path. By selecting nets that contain critical cells, we include other cells such as *f,g,h,i,j*, and *k* that are not timing critical. To remedy this problem, [5] employs PATH [8] weight function. However it is expensive since PATH requires the computation of exponential function. Here we propose two net filtering methods. The first one is *T filter*. In T filter, a net gets new delay weight only when there are at least two critical cells/clusters in it. This scheme now solves the problem illustrated in Figure 3.4. The second filter is *A filter*, where a net gets new delay weight only when all cells/clusters in the net are critical.

Figure 3.5 shows the impact of different filtering methods, where O represents the original GEO, T represents T filter method, and A represents A filter method. The results are normalized relative to GEO with no cut threshold (i.e. ε = 100%), whereas the rest use cut threshold ε = 5% of cells/clusters (i.e. top 5% minimum slack value as critical cells/clusters). Results show that on average, the T filter method is better than GEO and A filter by about 11% and 9% respectively. However, if we look at big circuits such as s9234, s13207, and s15850, the A filter yields better result. This is because it is harder to group large critical cells/clusters into the same partition when the number of cells/clusters is large. The A filter then can be used to consider only "highly" critical nets and hence reduce the number of cells/clusters. In our adaptive scheme, we use the A filter when the number of cells in current partitioned circuits is higher than 5,000 cells (based on results from Figure 3.5).

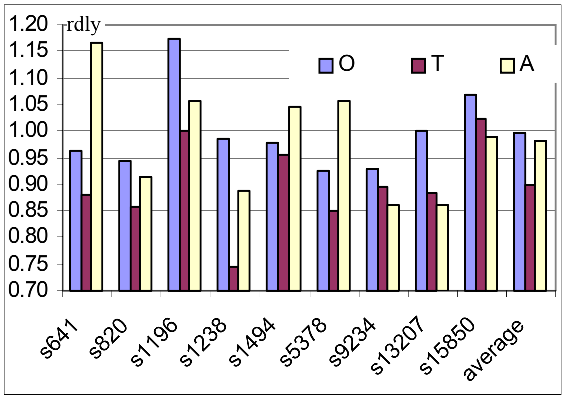

Figure 3.5 Impact on different net filtering

Next we study a way to dynamically update *ε* based on circuit characteristic. The *ε* parameter is used to decide how many cells are critical, for example first top 5% cells having minimum slack value. In [4,1,3,5], it is assigned as a fixed value. In our adaptive scheme, however, we first classify the cumulative slack values into three groups: *slow start*, *medium start*, and *fast start* based on the number of cells/clusters falling into top minimum slack value as shown in Figure 3.6. Here we plot slack distribution using initial clock cycle so that minimum slack will not have zero value. The boundary lines we use here are 15% for medium and 35% for fast start. From our observation, most circuits' initial partitions fall into the slow start category. We found that for slow start distribution, choosing the best ε value is difficult; especially where randomness is involved.

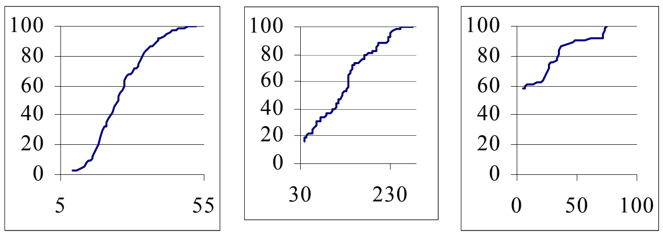

Figure 3.6 slack distribution classifications when x-axis represents slack value and y-axis represents cumulative frequency percentage normalized to 100%

The result in Figure 3.7 shows that the higher the number of runs the lower the variance on different ε values. In Figure 3.7, 5r and 20r represents 5 and 20 runs respectively, and the percentage represents the ε value. The graph is normalized relative to GEO with ε=100%. We found that the variance reduces from 0.0021 to 0.0008 when the number of runs increases from 5 to 20 runs. We also found that when the slack distribution of current partition falls into medium start category, selecting the first critical slack value is sufficient since it already contains a substantial number of cells/clusters for consideration. For the fast start distributions, assigning α to be zero (i.e. consider only minimizing wire length) is adequate, since there are too many critical cells/clusters, and it is hard to group these cells/clusters into the same partition without violating area constraints. Figure 3.8 shows the impact on the medium and the fast start case on bipartition with 5 runs on 2×1 slots normalized to a global placement with $\alpha = 0$. The slack distribution of s641 is shown in Figure 3.6 (b), and that of s35932 is shown in Figure 3.6 (c). For the medium start, once ε is higher than starting threshold, the retiming delay drops as can be seen when ε = 17%. On the other hand, for the fast start case, no matter what value of ε is, retiming delay is higher than partitioning targeting cutsize.

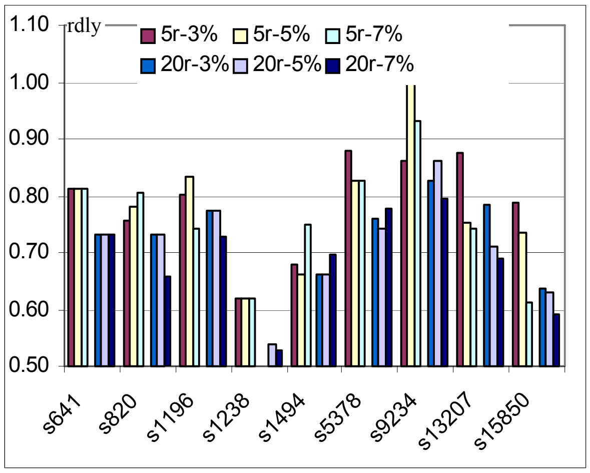

Figure 3.7 Impact on different ε value

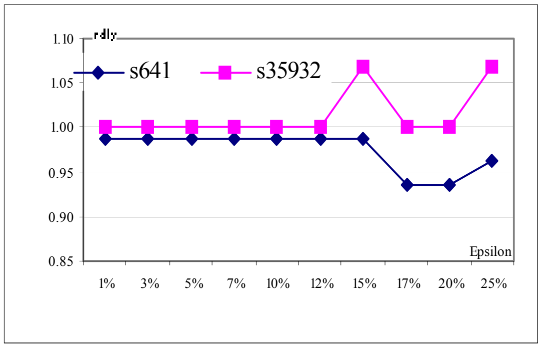

Figure 3.8 Examples of medium and fast start

3.5. Impact of Clustering

Circuit clustering is important for cutsize reduction, especially when considering a large number of cells. This also holds for weighted cutsize. As shown earlier in section 3.1, weighted cutsize has positive correlation with retiming delay. Hence for large circuit without clustering, the delay results are usually worse compared to the clustering case. However, we may loose accuracy of retiming based timing analysis (RTA) [4] if we perform clustering. This is because of the fact that the criticality of clusters needs to be computed based on the criticality of individual gates in the cluster. In fact, it is difficult to assign a single number to represent the timing criticality of all nodes in a cluster. In addition, clustering requires more runtime and more memory space. Therefore, there is a tradeoff in performing clustering in terms of solution quality versus runtime. Figure 3.9 shows this observation, i.e. when number of cells is large, clustering starts to outperform non-clustering approach. Here we propose a way to adaptively decide when to perform the clustering based on number of cells. We use 7,000 cells (based on results from the graph) as threshold to decide whether to perform clustering or not in this current partition. If the number of cells is higher than threshold, we employ clustering to reduce weighted cutsize.

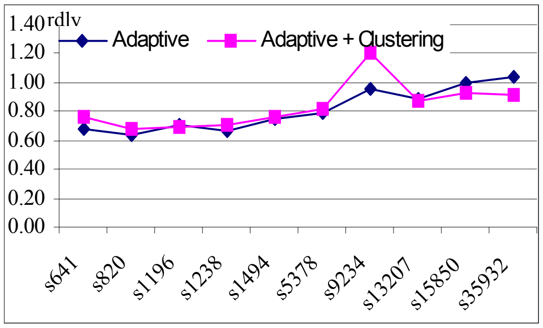

Figure 3.9 Impact on clustering

4. METHODOLOGY

We modify GEO [4] algorithm and call it A-GEO, the adaptive GEO. An overview of A-GEO algorithm is shown in Figure 4.1. Any part that is updated from GEO is underlined. Note that our adaptive scheme can not only be used for GEO but also any timing-driven mincut-based global placement. A-GEO produces a global placement solution for the PPR problem. Based on mincut-based global placement, we recursively bipartition the netlist *NL* until $m \times n$ tiles are generated. After all bipartitionings are finished for the current level, then we perform multiway refinement [19] on the entire netlist. Initially, the partitioning tree *T* has only root node *R*. Then all cells in *NL* are inserted into *R*. The FIFO queue *Q* is used to support the recursive breadth-first cut sequence.

```
A-GEO(NL,K,run)
1. insert all cells in NL to root node R in T (= part tree)
2. insert R into Q (= FIFO queue)
3. while (leaf nodes in T < K)
4.     N = remove front element in Q
5.     GEO-2way(N,run) (= bipartitioning on N)
6.     split cells in N into N1 and N2
7.    insert N1 and N2 into Q and T
8.    refine
9. return T
------------------------------------------------
 A-GEO-2way(N,run)
10. NL' = sub-netlist containing cells in N
11  if  #cells > Threshold T1
12.     ESC(NL') (= multi-level clustering on NL')
13. h = height of the cluster hierarchy
14. B = random partitioning among clusters at level h
15. for (i = h downto 0)
16.     compute adaptive #run
17.     NL'(i) = coarsened NL' at level I
18.     for (j=1 to run)
19.          while (gain)
20.          if not (bottommost level & node id > K/2)
21.         DELAY-WEIGHT(NL'(i))
22.         total net weight = 1 + α delay weight
23.             while (gain)
24.                move cells in NL'(i) to min. weighted cutsize
25.               retrieve max gain moves and update B
26.     project best retiming B to level i-1
27. return B
------------------------------------------------
 DELAY-WEIGHT(NL')
28. set delay of edges in R (= retiming G)
29. perform RTA(R) (= timing analysis)
30. compute sequential slack for nodes in R
31. for each cluster C in NL'
32.   C(R) = all cells in R grouped into C
33.   slack(C) = min among cells in C(R)
34.   X = ⌈top x% small slack⌉ if |X| ≤ 25%
35.   if #cells < threshold T2 use T filter
36.   else A filter
37. for each net N in NL'
38.   if original GEO filter
39.      compute delay-weight(N) using Eqn1
40.   elif (T filter and at least two cells in N are in X)
41.       compute delay-weight(N) using Eqn1
42.   elif (A filter and all clusters in N are in X)
43.      compute delay-weight(N) using Eqn1
```

Figure 4.1. Overview of the A-GEO algorithm

A-GEO-2way first generates a sub-netlist from the given partition tree node and performs multi-level clustering on it. ESC clustering algorithm [3] is used for this purpose. Then we obtain a random initial partitioning *B* among the clusters at the top level of the hierarchy. The subsequent top-down multi-level refinement is used to improve *B* in terms of delay. For timing driven global placement, RTA [9] is performed to identify timing critical cells/clusters. Then we compute the delay weights for the nets in the sub-netlist for delay optimization. The subsequent iterative improvement through a cluster move tries to minimize the weighted cutsize. Finally the current solution is projected to the next level coarser netlist for multi-level optimization. At the end of A-GEO-2way, two new children nodes are inserted into *T* based on *B*.

Table 5.1 Comparison among GEO, GEO+200r and A-GEO

ckt	GEO		GEO + 200r		A-GEO	
	Dr	wire	Dr	wire	Dr	wire
s641	143	409	137	383	97	442
s820	47	599	42	631	28	582
s1196	74	1,032	74	1,047	49	1,025
s1238	77	1,128	70	1,019	50	1,095
s1494	55	997	51	1,024	41	1,055
s5378	57	1,453	51	1,371	45	1,907
s9234	50	1,459	48	1,408	48	2,132
s13207	86	1,689	72	1,491	69	2,091
s15850	90	1,824	88	1,708	83	2,025
s35932	45	2,113	47	1,903	41	2,536
s38417	41	2,394	39	2,054	41	2,610
s38584	81	3,184	75	2,371	59	4,450
b14o	67	3,658	64	3,704	64	4,114
b15o	79	5,786	72	5,306	79	5,773
b20o	74	6,087	73	5,990	64	7,158
b21o	79	6,149	67	5,775	70	6,941
b22o	80	7,620	63	7,229	63	8,774
Avg.	1.00	1.00	0.93	0.91	0.82	1.14
Time	1,517		51,233		14,232	

We summarize the modifications made on GEO as follows. First, we know that selecting from best weighted cutsize does not guarantee the best retiming delay among all runs. Instead of returning *B* as the best weighted cutsize among all runs, we choose *B* based on real retiming results as shown in line 26. Second, we consider adaptive α in line 20, where we begin with α=20 and gradually decrease it. Third, we adapt the number of run in line 16 by starting with a high number of runs during earlier partitions and decreasing it gradually as described in Section 3.3. Here we use 20 runs for the first 3 levels and 3 runs for the rest. Fourth, we employ adaptive filtering based on number of cells in the current partition as in lines 35,36,40 and 42. Fifth, we consider the ε characteristic while selecting the ε value. If circuit characteristic is in slow start mode, we use $\varepsilon = 5\%$. On the other hand, If the circuit characteristic is in medium start mode, we use the first value in the medium range as ε. If it falls in the fast start mode, we set $\alpha = 0$ and consider only cutsize instead as shown in line 34. Finally, we decide to perform clustering based on the number of cells in line 11.

Since we project the best retiming delay to the next level, it requires calculating retiming delay after each run. However, each RTA requires O($n\log n$). To be more precise, our algorithm requires $run \times K \times n\log n$, where *run* represents number of runs, K represents the number of partitions, and n represents the number of cells.

5. EXPERIMENTAL RESULTS

Our algorithms are implemented in C++/STL, compiled with gcc v2.96 with –O3, and run on Pentium III 746 MHz machine. The benchmark set consists of twelve circuits from ISCAS89 [18] and five circuits from ITC99 [17] suites. We report our result in Table 5.1 on 8×8 tiles. GEO represents a state-of-the-art timing driven mincut-based global placement proposed in [4] with five runs. A-GEO represents the modified GEO algorithm with our adaptive methods with about 4.88 runs (using adaptive number of runs). We also report GEO+200r with is GEO with 200 runs to be fair since our A-GEO has a higher running time than original GEO. The average ratio and running time are also reported and measured in seconds. Results from Table 5.1 shows that the A-GEO is better than the GEO by about 21.9%, and better than the GEO+200r by 13.1%. Note that the GEO+200r requires more running time that the A-GEO by about four times. Hence by increasing number of run alone is not as good as using our adaptive method.

6. CONCLUSION AND FUTURE WORK

We propose an adaptive methodology to improve timing driven placement using adaptive parameters. Our method can improve a state-of-the-art timing driven placement GEO [4] by as much as 67% and 22% on average for performance improvement. We are working to employ c-timing [12] instead of retiming to reduce the running time.

7. REFERENCES

[1] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis, “Multi-objective Circuit Partitioning for Cut size and Path-Based Delay Minimization,” *IEEE International Conference in Computer Aided Design*, page 181-185, 2002.

[2] G. Beraudo and J. Lillis. Timing Optimization of FPGA Placements by Logic Replication. *ACM Design Automation Conf.* page 196-201, 2003.

[3] J. Cong and S. K. Lim, “Edge separability based circuit clustering with application to circuit partitioning,” *to appear in IEEE Trans on Computer-Aided Design*, 2003.

[4] J. Cong and S. K. Lim, “Physical Planning with Retiming,” *IEEE International Conference in Computer Aided Design*, page 2-7, 2000.

[5] J. Cong and X. Yuan. Multilevel Global Placement with Retiming. *ACM Design Automation Conf.* page 208-213, 2003.

[6] R. Ho, K. W. Mai and M. A. Horowitz. “The Future of Wires.“ In *Proceedings of IEEE*, 2001.

[7] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, “Multilevel hypergraph partitioning: Application in VLSI domain,” *ACM Design Automation Conf.*, page 526-529, 1997.

[8] T. Kong. A novel net weighting algorithm for timing-driven placement. *In Proc. Int. Conf. on Computer Aided Design*, pages 172-176, 2002.

[9] C. E. Leiserson and J. B. Saxe, Retiming synchronous circuitry. *Algorithmica*, page 5-35, 1991.

[10] I. Neumann and W. Kunz. Placement driven retiming with a coupled edge timing model. *In Proc. Int. Conf. On Computer Aided Design*, pages 95-102, 2001

[11] I. Neumann and W. Kunz. Tight coupling of timing-driven placement and retiming. In Proc. *IEEE Int. Symp. On Circuits and Systems*, pages 351-354, 2001.

[12] P. Pan. Continuous retiming: Algorithms and application. *In Proc. IEEE Int. Conf. on Computer Design*, pages 116-121, 1997.

[13] P. Pan, A. K. Karandikar, and C. L. Liu, “Optimal clock period clustering for sequential circuits with retiming,” *IEEE Trans on Computer-Aided Design, pages 489-498,1998.*

[14] SIA, National Technology Roadmap for Semiconductors, 2001.

[15] D. P. Singh and S. D. Brown. Integrated retiming and placement for field programmable gate arrays. *In Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, pages 67-76, 2002.

[16] T.C. Tien, H. P. Su, and Y.W. Tsay. Integrating logic retiming and register placement. *In Proc. Int. Conf. On Computer Aided Design*, pages 136-139, 1998.

[17] http://www.cad.polito.it/tools/itc99.html

[18] http://www.cbl.ncsu.edu

[19] Jason Cong and Sung Kyu Lim, "Multiway Partitioning With Pairwise Movement", IEEE International Conference on Computer Aided Design, p512-516, 1998.