Performance-Optimal Clustering with Retiming for Sequential Circuits

Abstract

We propose an exact *clustering with retiming* algorithm to minimize the clock period for sequential circuits. Without moving flip-flops (FF's) by retiming, conventional clustering algorithms can only handle combinational parts and therefore cannot achieve the best cycle time. Pan et al. [2] have proposed an optimal algorithm under the unit gate delay model. We propose a more powerful and faster algorithm that produces optimal results even under the more realistic general gate delay model. Experimental results show that the average ratio of the run time used by Pan's algorithm to ours is 2 : 1.

1 Introduction

Circuit clustering groups cells in a design into macros to satisfy some constraints such as area or pin limitations [3][4][5]. But this often induces large interconnect delays between macros. Therefore, avoiding performance degradation is a major objective when we perform circuit clustering.

Retiming, which repositions flip-flops (FF's) while preserving circuit functionality, can be used to shorten the clock period [1]. Traditional clustering techniques, which do not consider retiming, cannot achieve the optimal performance for sequential circuits [4][5]. These clustering methods often treat a sequential circuit as combinational parts by dropping all FF's and then clustering each combinational part independently. If we can appropriately relocate FF's by retiming when clustering a circuit, we can achieve better performance.

Pan et al. have proposed an approach to combine retiming and clustering[2]. Under the unit gate delay model[4], their algorithm can achieve the optimal clock period. For the general gate delay model, it can get near-optimal clock period within the maximum delay of any gates in the circuit. They use a labeling technique to achieve the retiming effect and to integrate it with clustering.

Pan's algorithm cannot produce the optimal results under the general delay model because it does not find the best labeling. If the relocated position for an FF, computed during labeling, is occupied by a gate, the labeling value of this gate has to be modified.

In this paper, we propose a new algorithm to cluster circuits with retiming using a new labeling method. This algorithm not only can achieve the optimal clock period under the general delay model but also uses less time than Pan's. Experimental results show that the average ratio of the run time used by Pan's algorithm to ours is 2 : 1.

The rest of this paper is organized as follows. Preliminaries are described in Section 2. How to enhance Pan's labeling method is presented in Section 3. For the convenience of illustrating our clustering algorithm, we first review Pan's algorithm in Section 4. Our algorithm is introduced in Section 5. Section 6 presents some experimental results. Finally, Section 7 draws some concluding remarks.

2 Preliminaries

A sequential circuit N is modeled as a directed (multi-)graph $G(V, E, d, w)$. Each vertex $v \in V$ corresponds to a combinational gate, a primary input (PI), or a primary output (PO) with propagation delay $d(v)$. Each directed edge $e \in E$ connects two vertices u and v if the corresponding gate of u drives, across zero or more registers, an input of the corresponding gate of v. The number of registers between u and v is denoted as edge weight $w(e)$ or $w(u, v)$. A path p consists of a sequence of successive vertices and edges such as $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$ and is denoted as $v_0 \overset{p}{\leadsto} v_k$. The path delay is the sum of the delay of each vertex of p: $d(p) = \sum_{i=0}^{k} d(v_i)$. The path weight is the sum of the edge weight of p: $w(p) = \sum_{i=0}^{k-1} w(e_i)$. The critical path of the circuit has the maximum zero-weight path delay denoted as $\phi(G) = max\{d(p) : w(p) = 0\}$.

A retiming of a circuit is an integer-valued vertex-labeling function $r : V \to Z$. Labeling vertex v with $r(v)$ means that $r(v)$ registers will be moved from the output to the input of v as shown in Fig. 1. The edge weight of the retimed graph $G_r(V, E, d, w_r)$ can be obtained by

$$w_r(e) = w(e) + r(v) - r(u).$$

A clustered circuit is composed of clusters and is functionally equivalent to the original one. The area of each cluster must be less than M, the given area constraint. The interconnect between clusters incurs large

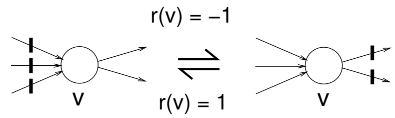

Fig. 1: Basic retiming operation.

delay, D, which is also a given parameter. We define the function $\lambda(u)$ to denote the interconnect delay from u to a cluster. Because we assume there is no inter-cluster delay for PI's and PO's, $\lambda(u)$ is zero if u is a PI or the cluster is a PO; otherwise it is D. A node can be duplicated without changing functionality for optimizing the clock period. The cycle time may differ from the original because of retiming. The clustering problem in this paper is as follows:

Problem 1 *Given a sequential circuit G, and a target clock period c, find a clustered circuit G_r with $\phi(G_r)$ less than or equal to c, if such a circuiting exists.*

3 The Weakness of Pan's Approach

In this section, we explain why Pan's approach cannot obtain the optimal clock period under the general delay model and give our solution. Pan's approach does not find the best labeling. The algorithm labels each node in the circuit an "*l-value*", defined as the weight of the longest path from the PI's to the node using the "w_1 *weight*". The w_1 weight is defined as $w_1(e) = -c * w(e) + d(v)$ for each edge $e : u \rightarrow v$, where c is the target clock period.

Next we explain the meaning of the l-value. Consider a path p starting from a PI u and ending at a node v. If we want to retime the path to satisfy the time constraint c, there must be at least $\lceil l(p)/c \rceil - 1$ FF's on p in the final circuit. Since there exists $w(p)$ FF's on p, we can set the retiming value $r(v)$ as $(\lceil l(p)/c \rceil - 1) - w(p)$. $(\lceil d(p)/c \rceil - 1) - w(p)$ can be further translated as follows:

$$\begin{aligned} & (\lceil d(p)/c \rceil - 1) - w(p) \\ = & \lceil d(p)/c - w(p) \rceil - 1 \\ = & \lceil (d(p) - c * w(p))/c \rceil - 1. \end{aligned}$$

We want to use a function related to v to represent the value $d(p) - c * w(p)$, so we let

$$l(v) = d(p) - c * w(p).$$

If p is the longest path from the PI's to v using the w_1 weight, $l(v)$ is the l-value. The use of l-value also gives a good failure condition for the retiming algorithm by checking on whether the l-value of the PO is greater than c [2]. If the l-value of a PO is greater than c, there exists no clustered circuit G_r with $\phi(G_r)$ less than or equal to c.

However, this labeling method is not adequate if we want to find the optimal solution under the general delay model. In fact, it attempts to relocate the ith FF to the position at which the propagation delay from PI equals to $i * c$. Unfortunately, this position is very likely to be at the middle of a gate. Since a gate cannot be split, we can only push this FF to the front of the gate to satisfy the timing constraint c. Thus this kind of approach may produce a solution near the optimal within the maximum delay of any gates in the circuit.

In our approach, we follow the method proposed in [6] to modify the l-value. The l-value of a node v can be computed by using the l-value of its fan-in node u by

$$l(v) = l(u) - c * w(u, v) + d(u).$$

If we find that there is an FF with computed position occupied by a gate v, the l-value of v is increased to

$$l(v) = lag(v) * c + d(u),$$

where $lag(v)$ is defined as

$$lag(v) = \begin{cases} 0 & \text{if } v \text{ is a PI} \\ \lceil l(v)/c \rceil - 1 & \text{otherwise.} \end{cases}$$

The final value of $lag(v)$ is just the retiming value $r(v)$. We can detect whether the computed position of an FF is occupied by a gate v by checking on whether $lag(l(v))$ is greater than $lag(l(v) - d(v))$.

Fig. 2 shows a simple example that Pan's approach cannot achieve the optimal clock period but we can. Fig. 2 (a) is the original circuit showing the gate delay on each node. The minimal timing constraint that Pan's labeling can achieve is 5. The l-value and the corresponding retiming value $r(v)$ for each node are listed in Fig. 2 (b). However, the critical path delay of the retimed circuit is 7. On the other hand, the minimal timing constraint that our labeling can achieve is 6. As shown in Fig. 2 (c), the critical path delay of the retimed circuit according to our labeling is 6, which is the optimal clock period.

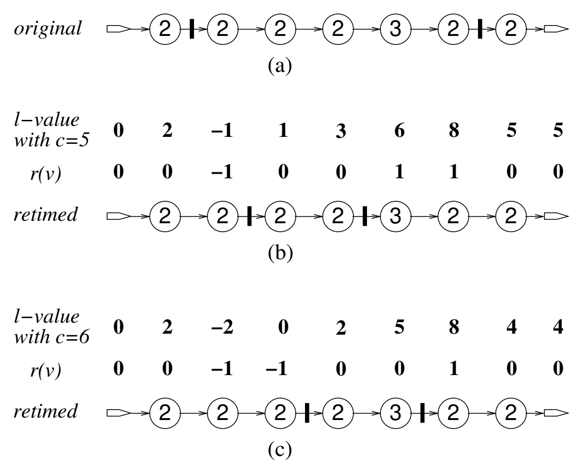

Fig. 2: (a) The original circuit. (b) The final critical path delay is 7 by using Pan's labeling. (c) We can achieve the optimal clock period 6 by using our labeling.

4 Outline of Pan's Algorithm

For illustrating our algorithm, we outline the algorithm proposed by Pan et al.[2] in this section. Their algorithm consists of two phases. The first is the labeling phase to compute the l-value and generate the corresponding cluster for each node. The second phase connects all nodes and clusters, retimes the clustered graph, and then merges clusters to reduce area. Clusters have to be merged because the algorithm generates one cluster for each node during the first phase. We address the labeling phase in this paper; the second phase can be found in [2].

As we have illustrated in the previous section, the l-value of a PO cannot be greater than the target clock period c. Instead of Problem 1, Pan's algorithm tries to solve the "*Strong Clustering Problem*":

Problem 2 *(Strong Clustering Problem) Given a positive integer c, find a clustered circuit such that the l-values of the PO's in the circuit are less than or equal to c.*

To reduce the search space, Pan's algorithm only considers "*simple clustered circuits*". A simple clustered circuit satisfies four conditions:

1. Each cluster has only one node that can output signals to the outside of the cluster. Thus we can name a cluster as C_v if it outputs from node v.
2. The retiming value at each node in the clustered circuits is zero.
3. For each node v in the circuit, there is at most one C_v.
4. If C_u is connected to nodes in C_v, then C_v does not contain a copy of u.

Details about the simple clustered circuit can be found in [2]. We just quote one of its theorems:

Theorem 1 *If a strong clustering problem has a solution, it has a simple solution.*

```
PANSLABEL(N, c)
 for each v in the circuit N do
   if (v is a PI) then l(v) ← 0
   else l(v) ← −∞
 for i ← 1 to B do /* B, number of iterations */
   changed ← FALSE
   for each non-PI node v in N do
     (l_new, C_new) ← PANSTIGHTENBOUND(v)
     if ( l_new > l(v) ) then
       l(v) ← l_new
       changed ← TRUE
     C_v ← C_new
     if (v is a PO and l(v) > c) then
       return FAILURE /* no solution */
   if (changed = FALSE) then
     return SUCCESS /* Bounds have settled */
```

Fig. 3: Pan's labeling procedure.

Pan's algorithm iteratively computes a new l-value for each node. A new cluster C_v which leads to the new l-value for node v is generated at the same time. If the present l-value of v is less than the new computed value, it is updated. If no more l-value can be updated, the algorithm terminates. Otherwise, the failure condition will be met for some PO with its l-value greater than the target clock period c. Fig. 3 gives the pseudo code of Pan's labeling procedure.

The new l-value of a node v and the new cluster C_v are calculated by the routine PANSTIGHTENBOUND as shown in Fig. 4. PANSTIGHTENBOUND first computes the candidate l-value of v for each u by assuming that u fanouts to C_v. $l'(u)$, the candidate l-value of v for each u, is computed by the equation:

$$l'(u) = l(u) + \Delta(u,v) + \lambda(u),$$

where $\Delta(u,v)$ is the w_1 weight of the longest path from u to v. Among these candidates, there exists an $l'(u)$ which may become the new l-value of v. So, a binary search is performed to find the minimum $l'(u)$ and the legal cluster C_v leading to the $l'(u)$. If we want to achieve a desired l-value L for v, there is no choice but putting all nodes with $l'(u)'s$ greater than L into the cluster C_v. Of course, the total area of the formed cluster C_v must be smaller than M, the given cluster size constraint. Finally, PANSTIGHTENBOUND will return a possible l-value for v to be compared with the present one and a legal cluster C_v.

```
PANSTIGHTENBOUND(v)
 for each node u in N do
   l'(u) ← l(u) + Δ(u,v) + λ(u)
 sort all l' values in increasing order L_1,L_2,...,L_t
 low ← 1, high ← t
 while (low ≤ high) do
   mid ← (low + high)/2
   remove every node u with l'(u) ≤ L_mid
   form cluster C_{v,L_low}
   if (area of C_{v,L_low} > M) then low ← mid + 1
   else high ← mid
 return (L_low, C_{L_low})
```

Fig. 4: Pan's procedure for improving the lower bound.

The time used by Pan's algorithm is mainly spent on $\Delta(u,v)$ calculation and the labeling phase. $\Delta(u,v)$ can be calculated in time $O(|V|^2 log|V| + |V||E|)$ by using an all-pair shortest path algorithm [10]. Time for the labeling phase depends on how many iterations will be executed (i.e., B in Fig. 3). Pan et al. give the upper bound on B as $|V|(|V|-1)D$, and PANSTIGHTENBOUND takes time $O((|V| + |E|)log|V|)$. Therefore, the time complexity for the algorithm is $O(|V|^3|E|Dlog|V|)$.

5 Proposed Algorithm

In this section, we present our exact algorithm for the performance-optimal clustering with retiming. We have illustrated our labeling method leading to the optimal clock period in Section 3 with

$$l(v) = \begin{cases} lag(v) * c + d(u) & \text{if } lag(l(v)) > lag(l(v) - d(v)) \\ l(u) - c * w(u,v) + d(u) & \text{otherwise.} \end{cases} \quad (1)$$

In addition, we adopt another circuit traversing method rather than Pan's by using FIFO's to incorporate our labeling technique for run-time efficiency.

The procedure PANSLABEL is a variation of the Bellman-Ford algorithm[1] [10]. We develop an algorithm similar to the retiming algorithm proposed by Chen [6]. Chen's algorithm uses a FIFO to store the nodes for updating rather than iteratively traversing the whole circuit. Experimental results show that Chen's algorithm runs much faster than the Bellman-Ford-like retiming algorithm [6]. Thus in our labeling procedure, we use a queue, called *queue*1, to store the nodes whose new l-values will be calculated. Initially, all PI's are put into *queue*1. Then, nodes are retrieved from *queue*1 one at a time. We update the l-value for a node if the calculated value is greater than the present one. Then we put all nodes reachable from the updated node into *queue*1 if they have a chance to be updated. The procedure stops if *queue*1 is empty or the failure condition is detected. Fig. 5 shows the procedure, LABEL.

LABEL(N, c)
for each u in the circuit N **do**
  for each v in the circuit N **do**
    $labelmatrix[u][v] \leftarrow -\infty$
  if (u is a PI) **then**
    $l(u) \leftarrow 0$
    if (UPADTELABELMATRIX(u,c)=FAILURE) **then**
      return FAILURE /* no solution */
  else $l(u) \leftarrow -\infty$
while ($queue1 \neq \emptyset$) **do**
  $v \leftarrow$ dequeue($queue1$)
  $(l_{new}, C_{new}) \leftarrow$ TIGHTENBOUND(v)
  if ($l_{new} > l(v)$) **then**
    $l(v) \leftarrow l_{new}$
    if (UPDATELABELMATRIX(v,c)=FAILURE) **then**
      return FAILURE /* no solution */
  $C_v \leftarrow C_{new}$
  if (v is a PO and $l(v) > c$) **then**
    return FAILURE /* no solution */
return SUCCESS /* Bounds have settled */

Fig. 5: Our labeling procedure.

Fig. 6 shows routine UPDATELABELMATRIX which decides on whether a node has a chance to be updated. Since a node reachable from the updated vertex v may have a chance to be updated, we use another queue, named *queue*2, to help traverse the sub-circuit starting from v. If the l-value of a node y computed according to the updated l-value of v is greater than the original value, y will be put into *queue*2 for further traversal. The candidate l-value (i.e., $l'(v)$) to a node y computed according to the l-value of node v is stored in $labelmatrix[v][y]$. The l-value is calculated according to Equation 1. Meanwhile, y is also put into *queue*1 if it is put into *queue*2 since its l-value has a chance to be updated. Routine TIGHTENBOUND as shown in Fig. 7 will calculate the possible l-value of y associated with C_y when y is retrieved from *queue*1.

UPDATELABELMATRIX(v,c)
if (lag($l(v)$+$\lambda(v)$) > lag($l(v)$)) **then**
  /* propagation delay exceeds c */
  /* a flip-flop is assumed to be moved here */
  $labelmatrix[v][v] \leftarrow \text{lag}(l(v)+\lambda(v)) \times c + \lambda(v)$
else $labelmatrix[v][v] \leftarrow l(v) + \lambda(v)$
enqueue(v, $queue2$)
while ($queue2 \neq \emptyset$) **do**
  $x \leftarrow$ dequeue($queue2$)
  for each fanout node y of x **do**
    $label0 \leftarrow labelmatrix[v][x] - c \times w(x,y)$
    $label \leftarrow label0 + d(y)$
    if (lag($label$) > lag($label0$)) **then**
      /* propagation delay exceeds c */
      /* a flip-flop is assumed to be moved here */
      $label \leftarrow \text{lag}(label) \times c + d(y)$
    if ($label > labelmatrix[v][y]$) **then**
      if ($label > 2c$) **then**
        return FAILURE /* no solution */
      $labelmatrix[v][y] \leftarrow label$
      if ($y \notin queue2$) **then** enqueue(y, $queue2$)
      if ($y \notin queue1$) **then** enqueue(y, $queue1$)
return SUCCESS

Fig. 6: Our procedure for updating the *labelmatrix*.

TIGHTENBOUND(v)
for each node u in N **do**
  $l'(u) \leftarrow labelmatrix[u][v]$
sort all l' values in increasing order $L_1, L_2, \ldots, L_t$
$low \leftarrow 1$, $high \leftarrow t$
while ($low \leq high$) **do**
  $mid \leftarrow (low + high)/2$
  remove every node u with $l'(u) \leq L_{mid}$
  form cluster $C_{v,L_{low}}$
  if (area of $C_{v,L_{low}} > M$) **then** $low \leftarrow mid + 1$
  else $high \leftarrow mid$
return $(L_{low}, C_{L_{low}})$

Fig. 7: Our procedure for improving the lower bound.

To prove the correctness of our algorithm, we give the following theorem:

Theorem 2 *Given a sequential circuit G and a target clock period c, there exists a retimed clustered circuit*

[1] Strictly speaking, PANSLABEL is not a pure Bellman-Ford algorithm because a failure condition is inspected to enhance the run-time efficiency.

G_r with $\phi(G_r)$ less than or equal to c if, and only if, procedure LABEL *returns SUCCESS.*

The proof for Theorem 2 is omitted due to space limitation.

The storage requirement is $O(|V|^2)$ for the all-pairs matrix, *labelmatrix*, which is the same as that of Pan's algorithm for storing $\Delta(u,v)$ for every node pair (u,v).

To analyze the time complexity, we first examine how many times a node will be visited in procedure LABEL. A node v will be visited only when the l-value of another node u is updated and there is a path from u to v. The increasing amount for an updated l-value is at least c/δ, where δ is the minimal positive difference between any two gate delays. The l-value has a range between $-c*|F|$ and $c*(|F|+1)$ because the largest possible propagation delay is $c*(|F|+1)$, where $|F|$ is the number of FF's in the circuit. As a consequence, *queue*1 is enqueued at most $\frac{c*(2|F|+1)}{\delta}|V|^2$ times. Next we examine how many times a node will be visited in routine UPDATELABELMATRIX. A node y will be visited only when one of its fan-in node, x, has been updated. By the similar analysis, $O(\frac{c*(2|F|+1)}{\delta}|E|)$ nodes will be visited in UPDATELABELMATRIX. Since the time complexity of TIGHTENBOUND is almost the same as PANSTIGHTENBOUND at $O((|V|+|E|)log|V|)$, our approach takes time $O((\frac{c*(2|F|+1)}{\delta}|V|^2)*(\frac{c*(2|F|+1)}{\delta}|E|+(|V|+|E|)log|V|)) = O(\frac{c^2}{\delta^2}|F|^2|V|^2|E| + \frac{c}{\delta}|F||V|^2|E|log|V|)$. Comparing our results with the time complexity of Pan's approach, we are not sure which is faster. Thus we need experiments to prove the efficiency of our algorithm.

6 Experimental Results

We have implemented our algorithm in C language and embedded it in the SIS package [7]. We have also implemented Pan's algorithm for comparison purpose. We run the experiments on an UltraSPARC-2 machine with 2GB of memory. The M is set as a quarter of the total circuit area, and D twice the average gate delay.

Experimental results on the ISCAS89 benchmark suite [9] are listed in Table 1. All the circuits are technology mapped by SIS using a $0.5um$ library from TSMC [8]. The number of gates for each circuit are listed in "$|V|$" Circuits are retimed before clustering and the results are listed in "$\phi(G)$" column. The maximal gate delay of each circuit is listed in column "max d" for reference. The minimal target clock periods that the labeling phase can achieve are listed in "min c" columns. The final clock periods are listed in "$\phi(G_r)$" columns. Results prove that the clock period obtained by Pan's algorithm is bounded by c plus maximal gate delay and may not be the optimal. 11 of totally 26 cases of Pan's results are sub-optimal as bold-faced in the table.

The run time in second is listed in "CPU" columns, and the ratio of the time used by Pan's algorithm over ours is listed in "Pan's/Ours CPU" column. Only for two cases (s382 and s9234) is our algorithm a little bit slower than Pan's. The average ratio of the run time used by Pan's algorithm to ours is 2 : 1.

The experimental results prove our algorithm valuable since it is an exact algorithm and run faster than the previous non-optimal heuristic.

7 Conclusions

We have presented an exact algorithm to *cluster-with-retiming* sequential circuits to get the optimal clock period. We pointed out why a previous work, Pan's approach, could not produce the optimal solution under the general delay model. We modified Pan's labeling methods and proposed an exact algorithm which used two queues to enhance run-time efficiency. Experimental results show that the average ratio of the run time used by Pan's algorithm to ours is 2 : 1.

These techniques can also be used for technology mapping with retiming. We will study this problem in the future.

References

[1] C. E. Leiserson, and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, vol.6, pp.5-35, June, 1991.

[2] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal Clock Period Clustering for Sequential Circuits with Retiming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.17, pp.489-498, June 1998.

[3] H. H. Yang, and D. F. Wong, "Circuit Clustering for Delay Minimization under Area and Pin Constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.16, pp.976-986, September 1997.

[4] R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli, "On Clustering for minimum delay/area," in *Proceedings of International Conference on Computer Design*, pp.6-9, 1991.

[5] E. Lawler, K. Levitt, and J. Turner, "Module Clustering to Minimize Delay in Digital Networks," *IEEE Transactions on Computers*, vol.18, pp.47-57, January 1969.

[6] W.-J. Chen, "A Study on the Relationship Between Retiming and Loop Folding," Master thesis, M312.93, National Tsing-Hua University, Taiwan, August 1994.

[7] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS : A System for Sequential Circuit Synthesis," *Memorandum No. UCB/ERL M92/41,* Electronics Research Laboratory, College of Engineering, University of California, Berkeley, May 1992.

[8] *TCB650 Library, 0.5um Standard Cell Data Book,* TSMC, April 1996.

[9] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profile of sequential benchmark circuits," in *Proceedings of International Symposium of Circuits and Systems,* pp. 1929-1934, May 1989.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms,* New York: McGraw-Hill, 1993.

Table 1: Experimental results.

circuit	Initial			Pan's			Ours			$\frac{\text{Pan's}}{\text{Ours}}$
	$\|V\|$	$\phi(G)$	max d	min c	$\phi(G_r)$	CPU	min c	$\phi(G_r)$	CPU	CPU
s208	83	2.74	0.42	3.08	3.08	2.0	3.08	3.08	1.0	2.0
s298	98	0.99	0.38	0.99	1.05	3.9	1.05	1.05	2.4	1.6
s344	180	2.10	0.42	2.10	**2.13**	11.9	2.10	2.10	8.0	1.5
s349	174	1.99	0.42	1.99	1.99	11.1	1.99	1.99	6.9	1.6
s382	204	1.40	0.38	1.40	**1.52**	21.9	1.45	1.45	*21.2*	*1.0*
s386	175	2.30	0.38	2.66	2.66	7.3	2.66	2.66	4.2	1.7
s400	198	1.69	0.42	1.69	**1.77**	24.3	1.69	1.69	9.7	2.5
s420	179	2.80	0.42	3.15	3.15	6.2	3.15	3.15	2.7	2.3
s444	197	1.54	0.38	1.54	**1.65**	19.0	1.56	1.56	10.6	1.8
s510	288	2.41	0.70	2.58	**2.90**	40.4	2.79	2.79	22.6	1.8
s526	198	1.45	0.38	1.45	**1.50**	11.7	1.45	1.45	8.2	1.4
s526n	185	1.46	0.42	1.46	**1.49**	13.9	1.46	1.46	6.7	2.1
s641	276	2.99	0.56	3.55	3.55	20.1	3.55	3.55	9.9	2.0
s713	225	2.35	0.38	2.69	2.69	10.3	2.69	2.69	4.7	2.2
s820	425	2.33	0.38	2.43	**2.59**	180.2	2.49	2.49	81.7	2.2
s832	428	2.33	0.38	2.50	**2.69**	96.3	2.50	2.50	52.7	1.8
s838	399	4.62	0.56	4.97	4.97	39.5	4.97	4.97	16.4	2.4
s1196	732	5.05	0.70	5.56	5.56	129.4	5.56	5.56	38.6	3.4
s1238	895	2.73	0.38	3.12	3.12	258.5	3.12	3.12	66.1	3.9
s1423	867	5.60	0.42	5.60	5.71	288.5	5.71	5.71	299.7	1.0
s1488	525	3.78	0.70	3.78	3.94	87.8	3.94	3.94	54.0	1.6
s1494	776	2.54	0.41	2.91	**3.05**	209.2	2.91	2.91	160.5	1.3
s5378	1403	3.78	0.70	4.28	4.28	850.8	4.28	4.28	300.0	2.8
s9234	1406	2.20	0.56	2.31	**2.46**	929.1	2.32	2.32	*1123.3*	*0.8*
s35932	11720	2.88	0.43	3.10	3.10	245944.3	3.10	3.10	23563.7	10.4
s38417	9750	6.19	0.70	6.19	6.19	16431.5	6.19	6.19	11675.8	1.4
average					3.03			2.99		2.1