A Unified Algorithm for Gate Sizing and Clock Skew Optimization to Minimize Sequential Circuit Area [*]

Weitong Chuang[†] Sachin S. Sapatnekar[‡] Ibrahim N. Hajj[†]

[†]Coordinated Science Laboratory and
Dept. of Electrical & Computer Engineering
University of Illinois at Urbana-Champaign

[‡]Department of Electrical Engineering
and Computer Engineering
Iowa State University

Abstract

This paper examines the problem of minimizing the area of a synchronous sequential circuit for a given clock period specification under the standard-cell paradigm. This is effected by appropriately selecting a size for each gate in the circuit from a standard-cell library, and by adjusting the delays between the central clock distribution node and individual flip-flops. Traditional methods treat these two problems separately, which may lead to very sub-optimal solutions in some cases. Experimental results show that by considering the two problems together, it is not only possible to reduce the optimized circuit area, but also to achieve faster clocking frequencies. We also address the problem of making this work applicable to very large synchronous sequential circuits by partitioning these circuits to reduce the computational complexity.

1 Introduction

The well-known sizing problem attempts to resolve the conflicting requirements of decreasing both the circuit area and the circuit delay. It is most often formulated as follows:

$$\begin{array}{ll} \text{minimize} & Area \\ \text{subject to} & Delay \leq P_{spec}. \end{array} \quad (1)$$

The discrete sizing problem [1–4], allows only a limited number of choices for each gate. These correspond to the gate configurations available in a standard cell library. This problem has been shown to be NP-complete [1].

The actual synchronous sequential circuit optimization problem is more complex than the simple sizing scenario. An additional degree of freedom is available to the designer in that one can set the time at which clock signals arrive at various flip-flops (FF's) in the circuit by controlling interconnect delays in the clock signal distribution network. With such adjustments, it is possible to change the delay specifications for the combinational stages of a synchronous sequential circuit to allow for better sizing.

In this paper, we examine the problem of minimizing the area of a synchronous sequential circuit, given a clock period specification, by appropriately selecting gate size for each gate in the circuit from a standard-cell library, and by adjusting the delays between the central clock and individual FF's. For simplicity, the analysis will use positive-edge-triggered D-flip-flops. In this paper, the words *flip-flop (FF)* and *latch* will be used interchangeably. We assume that all primary inputs (PI) and primary-outputs (PO) are connected to FF's outside the system, and are clocked with zero (or constant) skew.

2 Problem Formulation

2.1 Formulation of Constraints

Under the Elmore delay assumption, the delay of a gate can be represented by a piecewise linear function of its own size and that of its fanout gates. Due to limitations of space, we skip the details and refer the reader to [5].

[*]This research was supported by the Joint Services Electronics Program under contract N00014-90-J-1270.

In general, given a combinational circuit segment that lies between two FF's i and j, if s_i and s_j are the clock arrival times at the two FF's, we have the following relations:

$$s_i + Mindelay(i,j) \geq s_j + T_{hold} \quad (2)$$
$$s_i + T_{setup} + Maxdelay(i,j) \leq s_j + P_{spec} \quad (3)$$

where $Maxdelay(i,j)$ and $Mindelay(i,j)$ are, respectively, the maximum and the minimum combinational delays between the two FF's, T_{hold} and T_{setup} are, respectively, the set-up time and hold time of the FF, and P_{spec} is the clock period. Fishburn [6] studied the clock skew problem, under the assumption that the delays of the combinational segments are fixed, and formulated the problem of finding the optimal clock period and the optimal skews as a linear program. The objective was to minimize P_{spec}, with the constraints given by the inequalities in (2) and (3). In real design situations, however, P_{spec} is dictated by system requirements, and the real problem is to reduce circuit area.

2.2 Formulation of Linear Program

The delay specification requires that all path delays must be bounded by P_{spec}. To reduce the number of constraints, we introduce additional variables, m_i, $i = 1 \cdots \mathcal{N}$ (where $\mathcal{N}$ is the number of gates), corresponding to the longest-path delay from the primary inputs to each gate. For the shortest-path delay, we introduce p_i, $i = 1 \cdots \mathcal{N}$, corresponding to the shortest delay from PI's up to the output of G_i. Using these variables, we have

$$m_j + d_i \leq m_i, \quad \forall\, j \in Fanin(i). \quad (4)$$
$$p_j + d_i \geq p_i, \quad \forall\, j \in Fanin(i). \quad (5)$$

where d_i is the delay of gate i.

Our problem requires us to represent path delay constraints between *every* pair of PI's and PO's of a combinational subcircuit [6]. (Notice that, in a combinational subcircuit, the inputs of FF's are considered as pseudo PO's and the outputs of FF's are considered as pseudo PI's.) To represent path delays, we need intermediate variables m_k^i (p_k^i) to represent the longest (shortest) delay from the ith PI to the kth gate. We now formulate the linear program for a general synchronous sequential circuit as follows. The value of s_i is set to a constant (such as 0) if latch i is a PI latch or a PO latch.

$$\begin{array}{ll}
\textit{minimize} & \sum_{k=1}^{\mathcal{N}} \gamma_k \cdot x_k \\
\textit{subject to} & \\
d_k \geq \hat{D}(x_k, x_{k,1}, \ldots x_{k,fo(k)}), & 1 \leq k \leq \mathcal{N} \\
Minsize(k) \leq x_k \leq Maxsize(k), & 1 \leq k \leq \mathcal{N} \\
\text{For all FF } i, & 1 \leq i \leq \mathcal{L} \\
\quad s_i + p^i_{fanin(j)} \geq s_j + T_{hold} & 1 \leq j \leq \mathcal{L}, \\
\quad s_i + T_{setup} + m^i_{fanin(j)} \leq s_j + P_{spec} & 1 \leq j \leq \mathcal{L}, \\
\quad \text{For all gates } \; k = 1, \cdots, \mathcal{N} & \\
\quad\quad m_l^i + d_k \leq m_k^i, & \forall\, l \in Fanin(k) \\
\quad\quad p_l^i + d_k \geq p_k^i, & \forall\, l \in Fanin(k)
\end{array} \quad (6)$$

where γ_k is the *area coefficient* (a constant). The area of gate k is $\gamma_k \cdot x_k$ if gate k has size x_k. $x_{k,1}, \ldots x_{k,fo(k)}$ are the sizes of the gates to which gate k fans out, and $\hat{D}(\cdot)$ is a piecewise linear delay function. The above is a linear program in the variables x_i, d_i, m_i, p_i and s_i.

2.3 Symbolic Propagation of Constraints

For large circuits, the number of LP constraints could be tremendous. In this section, we propose a symbolic propagation method to prune the number of constraints by a judicious choice of the intermediate variables m and p, without sacrificing accuracy.

The combinational subcircuit is first levelized. Two string variables, $mstring(i)$ and $pstring(i)$, are used to store the long-path delay and short-path delay constraints associated with gate i, respectively. For each gate and each primary input (including pseudo PI's), an integer variable $w_i \in \{0,1\}$ is introduced to indicate its status. w_i has the value 1 whenever $mstring(i)$ and $pstring(i)$ are non-empty, i.e., when the constraints stored in $mstring(i)$ and $pstring(i)$ must be propagated; otherwise, $w_i = 0$.

The algorithm for propagating delay constraints symbolically is given in Figure 1. In the following discussion of the algorithm, we elaborate on the formation of $mstring$; the formation of $pstring$ proceeds analogously. At line 2, for each gate j, w_j and $mstring(j)$ are initialized by setting $w_j = 0$, and $mstring(j)$ to the null string. At line 5, we check if $w_l = 0$ for all $l \in fanin(k)$, i.e., if all of gate k's input gates have a null $mstring$. If so, no constraints need to be propagated. Next, at line 6, we check whether exactly one of all of gate k's input gates, say gate l', has a non-empty $mstring$, while all others have null $mstring$'s. If so, we continue to propagate the constraint. This is implemented by concatenating $mstring(l')$ and "d_k", and storing the resulting string in $mstring(k)$. Also w_k is set to 1 to indicate that further propagation is required at this gate. Finally, if more than one of gate k's input gates have non-empty $mstring$, we add a new intermediate variable, m_k^i, and the string "m_k^i" is stored at $mstring(k)$ (line 9). For each input gate whose $mstring$ is non-empty ($w_l = 1$), we need a delay constraint (line 11).

3 Mapping Phase

The solution of the LP would, in general, provide a gate size, x_k that does not belong to the permissible set, $\mathcal{S}_k = \{x_{k,1} \cdots x_{k,q_k}\}$. If so, we consider the two permissible gate sizes that are closest to x_k; we denote the nearest larger (smaller) size by x_{k+} (x_{k-}). Since the LP solution is likely to be close to the combinatorial problem solution, we formulate the following smaller problem:

For all $k = 1 \cdots \mathcal{N}$:
  Select $x_k = x_{k+}$ or x_{k-}, such that
  for all FF's $1 \le i, j \le \mathcal{L}$
    $s_i + Maxdelay(i,j) + T_{setup} \le s_j + P_{spec}$
    $s_i + Mindelay(i,j) \ge s_j + T_{hold}$

We have developed an algorithm, based on a breadth-first implicit enumeration approach, for mapping the gate sizes obtained using linear programming onto permissible gate sizes. Due to space limitations, the details of the algorithm are omitted, and are given in [5].

4 Satisfying Unresolved Delay Constraints

After the mapping phase, if some of the delay constraints cannot be satisfied, we have to fine-tune some gate sizes in the circuit. For each PO j (including pseudo PO's at the inputs of FF's), the required maximum (minimum) signal arrival times, $req_l(j)$ ($req_s(j)$), can be expressed as

$$\begin{aligned} req_l(j) &= s_j + P_{spec} - T_{setup} \\ req_s(j) &= s_j + T_{hold} \end{aligned} \quad (7)$$

4.1 Long Path Constraints

For every PO that violates long-path delay constraints, we identify the longest path to that PO. For example, if gate n at the PO, with the longest path signal arrival time m_n, violates $req_l(n)$, we first find the longest path, $P_l(n)$, to G_n. The *path slack* of $P_l(n)$ is calculated as

$$Pslack(P_l(n)) = req_l(n) - m_n \quad (8)$$

For each gate along that longest path, we calculate the *local delay difference* for each of the gates along path $P_l(n)$. Assume that G_{k-1}, G_k, G_{k+1} are consecutive gates, in order of precedence, on path $P_l(n)$. The local delay and local delay difference of G_k are defined, respectively, as

$$delay(G_k) = R_{out}^{k+1} \cdot C_{out}^{k+1} + R_{out}^k \cdot C_{out}^k \quad (9)$$
$$\Delta delay(G_k) = R_{out}^{k+1} \cdot \Delta C_{out}^{k+1} + \Delta R_{out}^k \cdot C_{out}^k \quad (10)$$

where R_{out}^k and C_{out}^k are, respectively, the equivalent driving resistance of gate k, and the capacitive load driven by gate k. Therefore, $\Delta delay(G_k)$ is the difference between the the original local delay of G_k and the new local delay of G_k after we replace it with a different gate size that has a different value of R_{out}^k and C_{out}^{k+1}.

The size of the gate G_l that satisfies

$$\Delta delay(G_l) < Pslack(P_l(n)) \quad (11)$$

is changed. If none of the local delay difference satisfies (11), we simply select the most negative one and replace the gate with a new realization. This process continues until the long-path delay constraints are all satisfied.

4.2 Short Path Constraints

Violations of short-path delay constraints can be resolved by inserting delay buffers. However, buffer insertion cannot be carried out arbitrarily, since one must simultaneously ensure that the changes in the circuit do not violate any long path constraints.

For every gate i, we define the *gate slack*, $Gslack(i)$, as

$$Gslack(i) = \quad (12)$$
$$\begin{cases} \min\{\min_{j \in FO(i)}[m_j + Gslack(j) - (d_j + m_i)], (req_l(i) - m_i)\}, & \text{if gate } i \text{ is at a PO.} \\ \min_{j \in FO(i)} \{m_j + Gslack(j) - (d_j + m_i)\}, & \text{otherwise} \end{cases}$$

At the beginning of this phase, we first back-propagate gate slacks from PO's and all FF's. The algorithm for inserting buffers is shown in Figure 3. In line (4) of the algorithm, beginning from the smallest buffer in the library, we try to insert a buffer at the output of gate G_{ni}. The delay of the buffer is denoted by $delay(bf)$. Since the output capacitance of G_{ni} is changed during this process, we have to recalculate its delay, which is denoted by $delay'(G_{ni})$.

5 Partitioning Large Circuits

Ideally for a given synchronous sequential circuit, all variables and constraints should be considered together to obtain an optimal solution. However, for large circuits, the size of the LP could be prohibitively large. Therefore, it is desirable to partition large synchronous sequential circuits into smaller subcircuits, so that we can apply the algorithm described earlier to each subcircuit.

Traditional partitioning problems usually have explicit objective functions. Our synchronous sequential circuit partitioning problem, however, is made harder by the absence of a well-defined objective function; since our ultimate goal is to minimize total area of the circuit, there is no direct physical measure that could serve as an objective function for partitioning. We develop a heuristic measure that will be shown to be an effective objective function for our partitioning problem.

We introduce the following terminology. An *internal latch* is a latch whose fanin and fanout gates belong to the same combinational block. A *sequential block* consists of a combinational subcircuit and its associated internal latches. *Boundary latches* are latches whose fanin and fanout gates belong to different combinational blocks.

For a given sequential block $\mathbf{B}$, let $L_{\mathbf{B}}$ denote the set of boundary latches incident on $\mathbf{B}$, and for a given boundary latch L, $\mathbf{B}_L$ denotes the set of sequential blocks on the latch L. For each boundary latch L, we define *tightness ratio* τ as

$$\begin{aligned}\tau_{in}(L) &= \text{maximum delay from any boundary latch to } L \text{ in the unsized circuit,}\\ \tau_{out}(L) &= \text{maximum delay from } L \text{ to any boundary latch in the unsized circuit,}\\ \tau(L) &= \begin{cases}\tau_{in}/\tau_{out} & \text{if } \tau_{in} \geq \tau_{out}\\ \tau_{out}/\tau_{in} & \text{if } \tau_{in} < \tau_{out}\end{cases} \qquad (13)\end{aligned}$$

where the adjective "unsized" implies that all gates in the subcircuit are at the minimum size.

For each pair of blocks $(\mathbf{B}_i, \mathbf{B}_j)$, define merit μ_{ij} as

$$\mu_{ij} = \sum_{\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j} \tau(L_k) \qquad (14)$$

where $\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j$ means latch L_k lies between $\mathbf{B}_i$ and $\mathbf{B}_j$. μ_{ij} is defined to be 0 if $\mathbf{B}_i$ and $\mathbf{B}_j$ are disjoint.

The cost associated with each block, $\mathbf{B}_i$, is c_i, the number of linear programming constraints required for solving $\mathbf{B}_i$. This number can be calculated very efficiently. Assume that group $\mathbf{G}_k$ consists of blocks $\mathbf{B}_{ki}, i = 1, \ldots |\mathbf{G}_k|$. Then we define the cost of $\mathbf{G}_k$, $C(\mathbf{G}_k) = \sum_{i=1}^{|\mathbf{G}_k|} c_{ki}$, and the merit of $\mathbf{G}_k$, $M(\mathbf{G}_k) = \sum_{i=1}^{|\mathbf{G}_k|} \sum_{j=i+1}^{|\mathbf{G}_k|} \mu_{ij}$. We now formulate the following optimization problem:

$$\begin{aligned} &max && \textstyle\sum_{k=1}^{N} M(\mathbf{G}_k)\\ &subject\ to && C(\mathbf{G}_k) < MaxCnstrts. \end{aligned} \qquad (15)$$

where N is the number of groups, $MaxCnstrts$ is the maximum number of constraints that one wishes to feed to the LP. Now that the partitioning problem has been explicitly defined, we develop a multiple-way synchronous sequential circuit partitioning algorithm based on the algorithm proposed by Sanchis [7].

For each group $\mathbf{G}_k$, and each boundary latch L, define the connection number, $\mathbf{\Phi}$, as:

$$\mathbf{\Phi}_{G_k}(L) = |\{\mathbf{B} | \mathbf{B} \in \mathbf{G}_k \ and\ \mathbf{B} \in \mathbf{B}_L\}|. \qquad (16)$$

The *gain* associated with moving $\mathbf{B}$ from $\mathbf{G}_i$ to $\mathbf{G}_j$ is

$$\begin{aligned}\mathbf{\Gamma}_{ij}(\mathbf{B}) = &\sum(\tau(L_l) | L_l \in L_{\mathbf{B}} \ and\ \mathbf{\Phi}_{G_j}(L_l) = 1)\\ &- \sum(\tau(L_n) | L_n \in L_{\mathbf{B}} \ and\ \mathbf{\Phi}_{G_i}(L_n) = 2) \quad (17)\end{aligned}$$

Given the initial partition, the algorithm improves it by iteratively moving one block from one group to another in a series of passes. A block is labeled *free* if it has not been moved during that pass. Each pass consists of a series of iterations during each of which the free block with the largest gain is moved. During each move, we ensure that the number of constraints in a group does not violate the limit given by (15). The gain number is updated constantly. At the end of each pass, the partitions generated during that pass are examined and the one with the maximum objective value is chosen as the starting partition for the next pass. Passes are performed until no improvement can be obtained.

6 Experimental Results

The algorithm above was implemented in C on a Sun Sparc 10 station. The test circuits includes many ISCAS89 benchmark circuits. Each cell in the standard-cell library has five different sizes of realization with different driving capabilities.

In Table 1, the experimental results of five ISCAS89 circuits are listed. For each circuit, the number of longest-path delay constraints without using symbolic constraint propagation algorithm and the number of constraints pruned by the algorithm are given. It is clear that our pruning algorithm is very efficient. The number of delay constraints is reduced by more than 95% on the average. For a given desired clock period (P_{spec}), the optimized results for both with and without clock skew optimization are shown. Depending on the structure of the circuits, the improvement over total area of the circuit ranges from 1.7% to almost 20%.

Table 2 provides some more experiments of s1423. In this experiment, we try to minimize the area using different specified clock periods. For s1423, the minimum clock period without clock skew optimization is about 32.5. On the other hand, using clock skew optimization, the minimum period can be as small as 22, which gives an almost 33% improvement in terms of clock speed.

Table 3 gives the experimental results for the partitioning procedure. Since most of the ISCAS89 circuits consist of only one combinational block, we generated some synchronous sequential random logic circuits. The number of gates and FF's in those circuits are shown in Table 3. For each circuit, we conduct three experiments. (1) Minimize the area using clock skew optimization, but without partitioning. (2) Minimize the circuit area using both clock skew optimization and partitioning. (3) Minimize the circuit with neither clock skew optimization nor partitioning.

It can be seen that the first approach is able to obtain the best result as expected. However, the runtime is large. Compared to the first approach, the second approach runs much faster, at a very slight area penalty. Not surprisingly, the third approach gets the worst solution. We also note that the introduction of clock skew provides a significantly faster clock speed for circuit m1337.

REFERENCES

[1] P. K. Chan, "Algorithms for library-specific sizing of combinational logic," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 353–356, 1990.

[2] S. Lin, M. Marek-Sadowska, and E. S. Kuh, "Delay and area optimization in standard-cell design," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 349–352, 1990.

[3] W. Li, A. Lim, P. Agrawal, and S. Sahni, "On the circuit implementation problem," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 478–483, 1992.

[4] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "Delay and area optimization for discrete gate sizes under double-sided timing constraints," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 9.4.1–9.4.4, 1993.

[5] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "Timing and area optimization for standard-cell VLSI circuit design," Tech. Rep. UIUC-ENG-93-2228, Coordinated Science Lab., University of Illinois, 1993.

[6] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Computers*, vol. 39, pp. 945–951, July 1990.

[7] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Computers*, vol. 38, pp. 62–81, Jan. 1989.

```
ALGORITHM Symbolic_propagation()
 1. for i = 1 to $\mathcal{L}$
 2.   $w_j \leftarrow 0$, $mstring(j) \leftarrow$ "", $ptring(j) \leftarrow$ "",  $\forall$ gates;
 3.   for j = 1 to max_level
 4.    for each gate $k$ at level $j$
 5.     if ( $w_l = 0 \; \forall \, l \in fanin(k)$ ); /* do nothing */
 6.     if ( among all $l \in fanin(k)$, exactly one $w(l) = 1$, others equal 0 )
 7.      $mstring(k) \leftarrow mstring(l') +$ "$d_k$", $pstring(k) \leftarrow pstring(l') +$ "$d_k$", $w_k \leftarrow 1$; /* $w_{l'} = 1$, $l' \in fanin(k)$ */
 8.     else
 9.      $mstring(k) \leftarrow$ "$m_k^i$", $pstring(k) \leftarrow$ "$p_k^i$", $w_k \leftarrow 1$;
10.      for all $w_l = 1$, $l \in fanin(k)$
11.       write down the constraints, $mstring(l) + d_k \leq m_k^i$,  $pstring(l) + d_k \geq p_k^i$;
```

Figure 1: Symbolic constraints propagation algorithm.

```
ALGORITHM Insert_buffer($n1$)
 1.   Let $P_s(n1)$ be the shortest path to gate $n1$, and $G_{n1}, G_{n2}, \cdots, G_{nk}$ be on path $P_s(n1)$
      ($G_{ni}$ fans out to $G_{n(i-1)}$, $2 \leq i \leq k$, $k$ = # of gates along $P_s(n1)$.);
 2.   $i \leftarrow 1$;
 3.   while ( $p_{n1} < req_s(n1)$ )
 4.      if ( $\exists$ a (smallest) buffer, $bf$, in the library such that:
         $delay(G_{ni}) < delay'(G_{ni}) + delay(bf) \leq delay(G_{ni}) + slack(G_{ni})$ )
 5.         insert $bf$ at the output of $G_{ni}$;
 6.         incrementally update $slack(j)$, $m_j$, $p_j$ for each gate $j$ in the circuit;
 7.         if ( $p_{n1} \geq req_s(n1)$ ) stop;
 8.         else goto 1.
 9.      $i \leftarrow i + 1$;
```

Figure 2: The buffer insertion algorithm.

Table 1: Performance comparison with and without clock skew optimization.

Circuit	longest-path constraints			P_{spec}	with clock skew opt.		w/o clock skew opt.		$\frac{A_1}{A_2}$
	original	pruned	%		Area (A_1)	Runtime	Area (A_2)	Runtime	
s208	3276	214	6.5%	6.8	1404.00	3.32s	1745.25	3.06s	0.805
s420	11830	544	4.6%	12.0	2522.00	9.06s	2952.63	8.94s	0.854
s838	55948	2670	4.8%	10.5	6162.00	100.67s	7324.42	43.77s	0.841
s953	34470	1788	5.2%	10.5	5516.87	243.93s	5898.75	67.69s	0.935
s5378	911854	6593	0.7%	10.0	29219.12	2633.78s	29717.53	1414.49s	0.983

Table 2: Improving possible clocking speeds using clock skew optimization.

Circuit	# of PI's	# of PO's	# of FF's	# of gates	P_{spec}	with clock skew opt.		w/o clock skew opt.		$\frac{A_1}{A_2}$
						Area (A_1)	Runtime	Area (A_2)	runtime	
s1423	17	5	74	657	32.5	9998.63	1130.89s	10545.71	84.05s	0.948
					30.0	10154.08	1450.03s	-	-	-
					22.0	12178.83	1605.43s	-	-	-
					20.0	-	-	-	-	-

Table 3: Performance comparison of the partitioning procedure.

Circuit	# of PI's	# of PO's	# of FF's	# of gates	# of blocks
m1337	51	53	97	1337	42
m1783	90	54	124	1783	43

Circuit	P_{spec}	with clock skew optimization						w/o clock skew opt.	
		w/o partitioning		with partitioning					
		Area	Runtime	MaxCnstrts	N	Area	Runtime	Area	Runtime
m1337	9.5	12364	135.35s	1500	6	12370	58.96s	13055	47.54s
	9.25	12353	151.34s	1500	6	12356	57.91s	-	-
	6.75	13049	186.61s	1500	6	13112	60.94s	-	-
m1783	9.5	18564	427.14s	300	16	18743	155.07s	21074	140.23s
				1000	8	18708	156.55s		
				2000	6	18572	159.93s		