Par-POPINS : A Timing-Driven Parallel Placement Method with the Elmore Delay Model for Row Based VLSIs

Tetsushi Koide Mitsuhiro Ono Shin'ichi Wakabayashi Yutaka Nishimaru
Faculty of Engineering, Hiroshima University
4-1, Kagamiyama 1 chome, Higashi-Hiroshima, 739 JAPAN
e-mail: {koide,wakaba}@ecs.hiroshima-u.ac.jp

Abstract— In this paper, we present a parallel algorithm running on a shared memory multi-processor workstation for timing driven standard cell layout. The proposed algorithm is based on POPINS2.0 [13] and consists of three phases. First, we get an initial placement by a hierarchical timing-driven mincut placement algorithm. At the top level of partitioning hierarchy, we perform one step of bi-partitioning by several processors, and in the lower levels of partitioning hierarchy, partitionings of each region in a level are performed in parallel. Next, in phase 2, iterative improvement of the sub-circuit which contains critical paths is performed by nonlinear programming. Parallel processing is realized by performing the nonlinear programming method to each sub-circuit in parallel. Finally, in phase 3, the placement is transformed to a row based layout style by a timing-driven row assignment method. We have implemented the proposed method on a 4CPU multi-processor workstation and showed that the proposed method is promising through experimental results.

I Introduction

In recent years, performance of integrated circuits becomes higher and operation speed of logic circuits becomes faster. Hence, for high performance VLSI chips, the interconnection delay is much longer than the gate delay, and it is a major part of whole signal delay of chips. Therefore *timing-driven layout methods* which take interconnection delay into account explicitly, have been urged to be developed.

There have been many studies about timing-driven placement, and they can be classified into the following four approaches, (1) the net weighting approach [1, 3, 15, 17, 20], (2) the net delay bounds approach [6, 9, 10, 16, 21], (3) the path weighting approach [8, 22], and (4) the path delay bounds approach [2, 7, 11, 12, 19]. However, many of them have a difficulty of trade-off between the quality of the layout and the computation time. Especially for interconnection delay, the estimation of the interconnection delay is inaccurate because of some simplified assumptions of the delay model. Then, we have proposed a timing-driven placement method, called POPINS2.0 [13], which is based on the path delay bounds approach and adopted the delay estimation model based on Elmore's delay model. Experimental results have showed the effectiveness of POPINS2.0. However, in the case of VLSIs which have so much cells in one chip, the computation time of this placement method increased, because of the iterative improvement phase based on nonlinear programming and the row assignment phase based on linear assignment.

In this paper, we propose a parallel algorithm for timing-driven placement, which is an extension of POPINS2.0. To obtain high efficiency of parallel processing, we restrict the amount of communication among processors as much as possible by partitioning the placement problem into some subproblems which can be executed independently. The proposed algorithm consists of three phases. First, we get an initial placement by a hierarchical timing-driven mincut placement algorithm. At the top level of partitioning hierarchy, we perform bi-partitioning by several processors, and in the lower levels of partitioning hierarchy, partitionings of each region in a level are performed in parallel. Next, in phase 2, we select a sub-circuit, which contains critical paths violating the given timing constraints, and improve the placement of the sub-circuit by nonlinear programming. Parallel processing is realized by performing the nonlinear programming method to each sub-circuit in parallel. Finally, in phase 3, the placement is formed to a row based layout style by a timing-driven row assignment method. From the experimental results comparing with POPINS2.0 and RITUAL [19], the proposed parallel placement algorithm is very promising.

The remainder of this paper is organized as follows. In Section II, we describe the interconnection delay model and the timing constraint treated in this paper, and formulate the timing-driven placement problem. In Section III, we propose a parallel algorithm for timing driven placement. Experimental results and the evaluation of the proposed algorithm are shown in Section IV. Finally, in Section V, we describe the conclusions and future works.

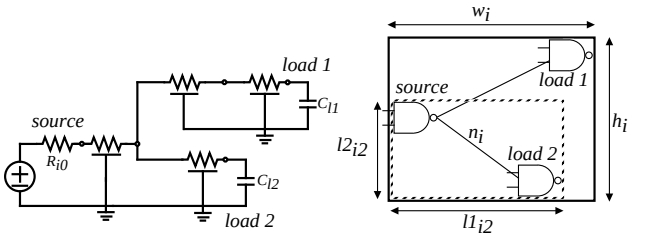
II Preliminaries

A. Layout and Delay Models

In this paper, the row based design such as the poly-cell type standard cell or the gate array models is assumed. We assume that the interconnections are realized by using two layers, the first metal layer (M1) and the second metal layer (M2). The M1 layer is mainly used for horizontal wiring and the M2 layer is mainly for vertical wiring.

An equivalent circuit of an interconnection is originally

ASP-DAC '97
0-89791-851-7$5.00



(a) Interconnection delay model *(b) Estimation of the wire length of a net n_i*

Fig. 1. Delay model.

modeled as a distributed RC circuit, and the *Elmore's delay equation* [4] is often used to represent the interconnection delay(Fig.1(a)). When a multi-terminal net n_i is implemented by a Steiner tree, Kuh and Shih give an upper bound of the Elmore delay from the source pin to the load pin j of the net n_i in Ref. [14]. Since, in our model, the wire capacitances are different between M1 and M2, we compute the delay as the sum of delay of M1 and M2. Furthermore, it is not practical to construct Steiner trees during placement from the point of computation time. We hence estimate the wire length of a net n_i by the half perimeter length of a bounding box of enclosing the pins of the net and the wire length from the source to a load j of the net n_i by the half perimeter length of a bounding box enclosing the source pin and the load pin j(Fig.1(b)). The delay from the source pin to the load pin j of the net is thus defined as,

$$d_{ij}(w_i, h_i, l1_{ij}, l2_{ij}) = (c1 \cdot w_i + c2 \cdot h_i + \sum_{k_i} C_{lk_i}) \times (R_{i0} + r1 \cdot l1_{ij} + r2 \cdot l2_{ij}), \quad (1)$$

where w_i and h_i are the width and height of the bounding box of the net n_i, $l1_{ij}$ and $l2_{ij}$ are the width and height of the bounding box enclosing the source pin and load pin j of net n_i, $c1$ and $c2$ are the capacitance of M1 and M2 per unit length, and $r1$ and $r2$ are the resistance of M1 and M2 per unit length, respectively.

B. *Timing-Driven Placement Problem*

In this paper, we consider the long path problem. As there are many paths from a primary input(PI) or an output of flip-flops(FFs) to a primary output(PO) or inputs of FFs, they can be specified by pairs of pins, source ones and sink ones. Thus we specify a timing constraint as $t_\tau = (s_\tau, e_\tau, D_{allow_\tau})$, where s_τ is a source pin, e_τ is a sink pin, and D_{allow_τ} is the maximum allowable delay from the source to the sink. We have to get the layout satisfying all elements of the set of timing constraints $\mathcal{T}$.

We define some terminologies and symbols. Let $\mathcal{L} = (\mathcal{M}, \mathcal{N})$ be a logic circuit, where $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$ is a set of cells and $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$ is a set of nets. A set $\mathcal{N}_i$ is a set of nets connecting to a cell m_i, and a set $\mathcal{M}_j$ is a set of cells connecting to a net n_j. For every timing constraint $t_\tau \in \mathcal{T}$, we define *a constrained path*, denoted by $p_\pi = (\mathcal{M}_\pi, \mathcal{N}_\pi)$, as any path whose source is s_τ and sink is e_τ, where $\mathcal{M}_\pi$ is a set of cells which are on the constrained path and $\mathcal{N}_\pi$ is a set of nets which have connection to some cell on the constrained path. Let $\mathcal{P}$ be a set of constrained paths and let $\mathcal{P}_\tau \subset \mathcal{P}$ is a set of constrained paths specified by a timing constraint $t_\tau \in \mathcal{T}_\tau$. Let $D_{act\pi}$ be the actual propagation delay of $p_\pi \in \mathcal{P}_\tau$.

For every timing constraint $t_\tau \in \mathcal{T}$, let $\mathcal{L}_\tau = (\mathcal{M}_\tau, \mathcal{N}_\tau)$ be a *constrained circuit*, in which a set of cells and nets are defined as $\mathcal{M}_\tau = \bigcup_{\forall p_\pi \in \mathcal{P}_\tau} \mathcal{M}_\pi$, and $\mathcal{N}_\tau = \bigcup_{\forall p_\pi \in \mathcal{P}_\tau} \mathcal{N}_\pi$. If $\mathcal{M}_\tau$ are regarded as vertices and $\mathcal{N}_\tau$ as edges whose directions are given by corresponding signal flows, a constrained circuit c_τ is represented as a directed acyclic graph [13], in which the source is s_τ and the destination is t_τ. Let $\mathcal{C}$ be a set of constrained circuits, and let $D_{act\tau} = \max_{\forall p_\pi \in \mathcal{P}_\tau} D_{act\pi}$ be the actual propagation delay time from s_τ to e_τ.

Now, we formulate the performance driven placement problem. Given a logic circuit $\mathcal{L} = (\mathcal{M}, \mathcal{N})$, timing constraints $\mathcal{T}$, and physical parameters of equation (1) of the Elmore delay model, the problem is to determine positions of $\mathcal{M}$ which minimize the estimated total wire length of nets under the layout and the timing constraints. The estimated total wire length is the sum of estimated wire length of all nets, and we estimate the wire length of a net by the half perimeter of bounding box of the pins of the net.

III The Proposed Parallel Placement Algorithm

In this section, to reduce the computation time furthermore, we extend the algorithm to a parallel algorithm proposed in [13] running on a shared memory multi-processor workstation. To obtain high efficiency of parallel processing, we restrict the amount of communication among processors as much as possible by dividing the placement problem into some number of sub-problems which can be treated independently. The proposed parallel algorithm for timing-driven placement is based on the sequential algorithm for timing-driven placement, POPINS2.0 [13], which we have proposed previously. In the following, first, we will describe the outline of POPINS2.0 and then we will propose the parallel algorithm running on a shared memory multiprocessor workstations. The details of POPINS2.0 are given in [13].

A. *The Outline of the Sequential Algorithm POPINS2.0*

POPINS2.0 consists of three phases. In phase 1, it generates an initial placement so as to minimize the number of cuts and the total wire length by a hierarchical timing-driven mincut placement algorithm in a comparatively short computation time. Next, it iteratively improves the initial placement obtained in phase 1 by the algorithm based on nonlinear programming in phase 2. In each iteration, a placement of a sub-circuit which contains critical paths violating their timing constraints is improved. Finally, it assigns the cells in rows with linear assignment considering the timing constraints.

In the initial placement of phase1, in order to minimize the total wire length, to distribute cells uniformly in the placement region, and to reduce the violations of timing constraints, we employ an extended version of the timing-driven mincut placement algorithm, which we proposed in [22] as the algorithm

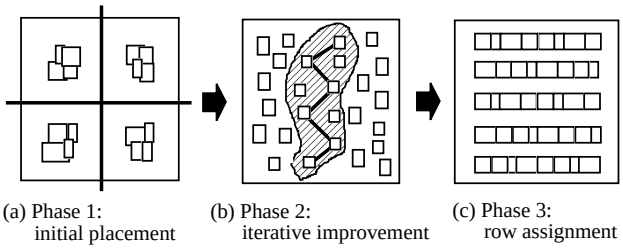


Fig. 2. The outline of POPINS2.0.

of this phase. This algorithm is based on ordinary hierarchical quadratic partitioning. The quadratic partitioning is realized by applying the well-known Fiduccia and Mattheyses' bi-partitioning method, called the FM method [5], in three times (Fig.2(a)). We extended the FM method so as to consider timing constraints. In addition to the original gain for minimizing the cut size, called *cut gain*, we introduce the gain to handle timing constraints, *slack gain* [13], the gain to consider terminal positions of nets, *terminal gain* [13], and the gain to consider the wire length of nets, *wire gain* [13]. Consequently, the gain of a cell is defined as the sum of above four gains, that is, *gain* = $\alpha\times$ *(cut gain)* + $\beta\times$ *(slack gain)* + $\gamma\times$ *(terminal gain)*+ $\delta\times$ *(wire gain)*, and we can obtain an initial placement that minimizes the cut size and the total wire length, distributes cells uniformly in the placement region, and reduces the violations of timing constraints as much as possible.

In phase 2, we iteratively improve the initial placement obtained by phase 1. The objective of phase 2 is to eliminate all the violations of timing constraints and to minimize the total wire length of the placement. To achieve this, we transform the placement problem under the timing constraints to a set of nonlinear programming(NLP) problems and solve them. NLP tends to require much computation time and memory space. We hence apply NLP to sub-circuits, for which the formulated problem can be solved in a practical computation time and with practical size of memory space. The sub-circuit to be improved, called *target sub-circuit*, is selected in such a way that the target sub-circuit includes *a critical path* for which the violation of the timing constraint is large (Fig.2(b)). This improvement based on NLP are applied iteratively until all the timing constraints are satisfied or iteration count reaches to some preset upper bound value.

In phase 3, the cells, which are distributed on the chip in phase 2, are assigned to cell rows(Fig.2(c)). In the proposed row assignment algorithm, first, the cells between each two consecutive cell rows are grouped, and next for each group, the cells in it are assigned to slots in the two consecutive cell rows by linear assignment considering the wire length and the timing constraints (*row assignment of y-direction*). Next, cell groups are constructed based on x-coordinates of cells from left to right of the chip, and the cells in each group are reassigned to slots of the improved region by linear assignment in a similar way of y-direction (*row assignment of x-direction*). The above operations are iteratively carried out while the placement is improved.

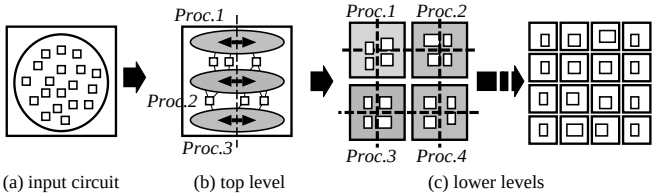


Fig. 3. The parallelization of phase 1.

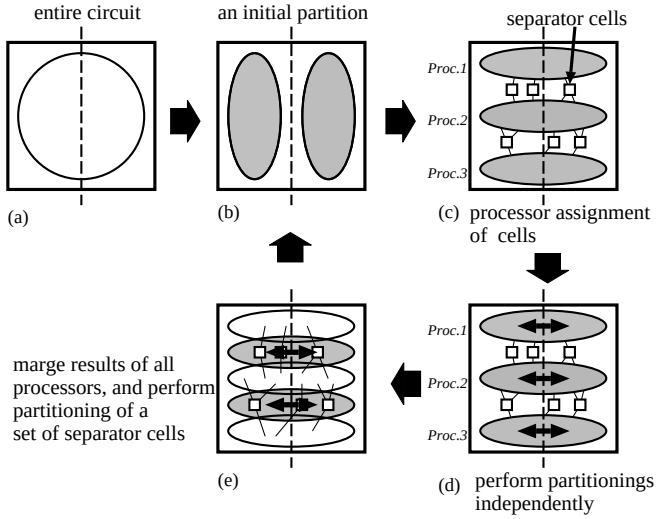


Fig. 4. The parallel algorithm of a bi-partitioning at top level.

B. Parallelization of Phase 1

In the parallelization of phase 1, at the top level of hierarchy, we perform bi-partitioning by several processors, and at the lower levels, carry out partitionings of each region at a level in parallel(Fig. 3). Since the bi-partitioning algorithm used in the parallel algorithm is the same as POPINS2.0, in the following, we will explain the detail of parallelization of top level and lower levels, respectively.

1. Partitioning at Top Level of Hierarchy

At the top level of hierarchy, since we have to handle the entire circuit as well as to reduce the computation time, we perform bi-partitioning by using several processors in parallel. In general, it is difficult to parallelize the FM method directly because it consists of a set of sequential operations intrinsically. In the proposed algorithm, we hence divide the entire set of cells into the sets of cells and perform partitioning of each set of cells independently. As shown in Fig. 4, firstly, we assign every cell to each processor based on an initial partition(Fig.4(c)). Next, each processor performs the partitioning of cells assigned to it with the bi-partitioning method independently(Fig.4(d)). To obtain a bi-partition of whole cells, we merge the partitioning results of each processors(Fig.4(e)) and then we reassign cells to each processor based on the current bi-partition. These operations are iteratively performed while the bi-partition is improved.

However, if the cells which are connected to each other are assigned to the different processors, the gain of those cells

would not be calculated correctly. Thus, in the proposed algorithm, we introduce a set of *separator cells* that divide the entire set of cells into the sets of cells which have no connection. Since the separator cells are not assigned to any processors, the positions of separator cells are fixed while each processor is performing the partitioning(Fig.4(d)). In order to improve the partition, after each processor's partitioning, the partitioning of the set of separator cells are done(Fig.4(e)). However, the movement of cells which improves the whole partitioning are hard to realize because of other fixed cells. We hence add some cells to the set of the separator cells in the breadth-first-search manner from the set of the separator cells, then the partitioning method is applied to the set of both the separator cells and added cells. Introducing separator cells makes it possible to calculate the cut, terminal, and wire gains correctly. However, it is impossible to calculate the slack gain correctly, because the calculation of slack gain needs to calculate a delay of a path, and it is difficult to separate cells in each path with no connection. Moreover, the terminal and wire gains are not useful for the top level partition, because there is only one partitioning region, i.e., whole chip area, and these gains are used to consider the outside of partitioning region. In the top level partitioning, we therefore perform the partitioning considering mainly the cut gain, and other gains are only used as auxiliary gains.

In the assignment of cells to processors(Fig.4(c)), we need to minimize the number of separator cells and to equalize the number of cells assigned to each processor as much as possible. We therefore perform a cell assignment to processors in the following way. Firstly, a pair of seed cells for all processors which has a connection and crosses the cut line of the current partition is randomly selected and is assigned to a processor. Next, expand each set of cells assigned to a processor by adding cells one by one considering the balance of each set of cells. The added cell should have large connectivity, which is the number of connections to the present set of cells assigned to each processor. If the added cell is connected to a cell which has already been assigned to other processor then the cell becomes a separator cell and is added to the set of separator cells. This operation is repeated until all cells have been added to some sets of cells assigned to a processor or a set of separator cells.

2. Partitioning at Lower Levels of Hierarchy

At the lower levels of hierarchy, there are many partitioning regions, and we have to perform the partitioning of many small sized sets of circuits(Fig.3(c)). So, in the proposed algorithm, we apply the partitioning algorithm independently to perform partitioning of each region at a level.

Figure 5 shows an example of processing flow which obtains the 4 by 4 partitioning from the 2 by 2 partitioning. In order to obtain high efficiency in parallel processing, updating the cell positions to calculate slack, terminal and wire gains is performed asynchronously by each processor during cell moving in the partitioning algorithm. Moreover, in order to reduce the inconsistency of cell positions caused by moving cells by other processors, we perform the partitioning, in which the directions of the cut lines of the partitioning regions abutted on

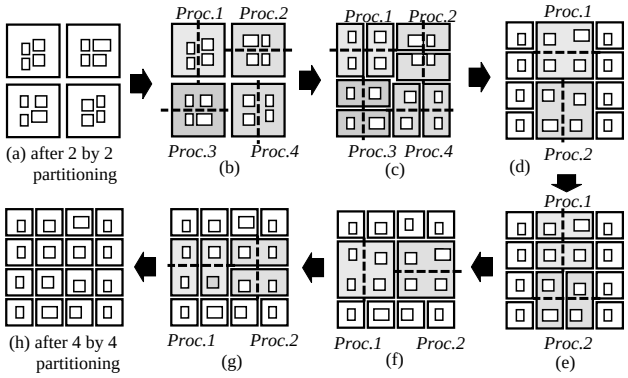


Fig. 5. Partitioning at lower levels.

each other are different from each other as shown in Fig. 5 (b)(c). The effect of the inconsistency of cell positions is decreased in the lower levels of partitioning hierarchy because the regions to be partitioned are apart from each other. We hence change the parameters (α, β, γ, and δ) of gain dynamically during the hierarchy of partitioning so that firstly, the cut gain is mainly considered, and the effect of other gains are increasing as the hierarchy of partitioning goes down to lower levels. In addition to above consideration, in the proposed algorithm, the bi-partitioning method is hierarchically applied by shifting the partitioned region as shown in Fig. 5 (d) $\sim$ (g). Since the precise cell positions are already determined after the 4×4 partitioning as shown in Fig. 5 (c), the terminal and wire gains can be accurately calculated and a good initial placement can be obtained.

C. Parallelization of Phase 2

1. Selection of a Target Sub-circuit

As mentioned in Section A., in phase 2, iterative improvement based on NLP is performed to improve the placement obtained by phase 1. The parallelization of phase 2 is achieved by solving the NLP problems [13] for each target sub-circuit by several processors in parallel (Fig. 6). However, if each target sub-circuit has connections with the other target sub-circuit and is improved concurrently, errors of wire length estimation and path delay estimation have occurred. We therefore construct a target sub-circuit which has no connection with other target sub-circuit which is improved concurrently by introducing *separator cells* as shown in Fig. 6. The paths which are included in several target sub-circuits are also divided into several paths by the separator cells and considered independently.

Now, let a $\mathcal{L}_{mov} = (\mathcal{M}_{mov}, \mathcal{N}_{mov})$ be *target sub-circuit*, where $\mathcal{M}_{mov}$ is the set of cells, called *movable cells*, of the target sub-circuit and $\mathcal{N}_{mov}$ is the set of nets, called *movable nets*, connecting to at least one movable cells, respectively. The cells other than movable cells are called *fixed cells* and denoted as $\mathcal{M}_{fix}$. The nets other than movable nets are called *fixed nets* and denoted as $\mathcal{N}_{fix}$. And let $\mathcal{M}'_{mov}$ represent a set of cells which belongs to other target sub-circuit to be improved by other processors.

The construction of a set of target sub-circuits is as follows.

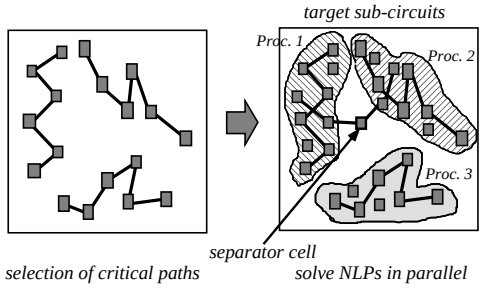


Fig. 6. The parallelization of phase 2.

First, we find one of constrained paths with a large violation ratio, and let the constrained path be the initial sub-circuit. *Violation ratio* is the value of actual delay time of a constrained path (or a constrained circuit) divided by the maximum allowable delay time of it, i.e., $D_{act_\pi} / D_{allow_\pi}$. The candidates for the constrained path are selected from constrained circuits which are the largest $10 \sim 20$ percent in all constrained circuits in terms of the violation ratio. But, to improve the placement in small number of iteration, we don't select any constrained circuits which have been selected in the last $k(> 0)$ iterations[1] of phase 2. Moreover, to perform the improvement to the sets of the target sub-circuits in parallel, we select the constrained path so as to have no connection with any other sub-circuits improved by other processors. As increasing the number of cells, it may become easy, because the placement area is very large, and we can select a path in which cells are placed at the position apart from any other sub-circuits improved by other processors. Next, expand the sub-circuit by adding cells one by one. The added cell should have large *connectivity*, which is the number of connections to the present sub-circuit. In order to avoid repeatedly selecting the same cell to be added to the target sub-circuit in each iteration of phase 2, we introduce a randomness in the selecting step and determine whether the cell is included. Furthermore, to solve the problem in a practical computation time, we must limit the number of variables in the NLP and hence the growing process of the target sub-circuit continues until the number of variables of the NLP problem reaches to a given constant.

2. *The Algorithm of Phase 2*

The algorithm of phase 2 consists of one master process and some slave processes, and improves a placement iteratively. In each iteration, firstly, the master process performs the timing verification and calculates violation ratios for all constrained circuits. Next, the master process constructs a target sub-circuit, and forks a slave process. This loop is repeated until the maximum violation ratio is less than a pre-determined permissible violation ratio or the loop count reaches to some preset value. The slave process receives a target sub-circuit from the master process, formulates a NLP problem and solves it. And it sends the result of NLP problem to the master process. The outline of the algorithm of phase 2 is shown below.

[1] In the current implementation, we set k to three.

[Iterative Improvement Based on NLP]
Master process :

Step 1: Perform timing verification to all constrained circuits, and
LoopNumber = 1.

Step 2: If the maximum violation ratio is less than a pre-determined permissible violation ratio or *LoopNumber* > (*preset value*), then wait for the termination of all slave processes, receive results from the slave processes, and exit.

Step 3: Select a target sub-circuit $\mathcal{L}_{mov}$ to be improved, and start a slave process.

Step 4: If there is a slave process which have terminated then receive a result from the slave process, and perform timing verification to the constrained circuits which has $\mathcal{M}_{mov}$. If it cannot increase a slave process any more, then wait for ending a slave process.

Step 5: *LoopNumber* = *LoopNumber* + 1, go to **Step 2**

Slave process :

Step 1: Find all constrained paths which have the cells of $\mathcal{M}_{mov}$.

Step 2: Formulate the nonlinear programming problem and solve it.

Step 3: Send the result of the nonlinear programming problem to the master process.

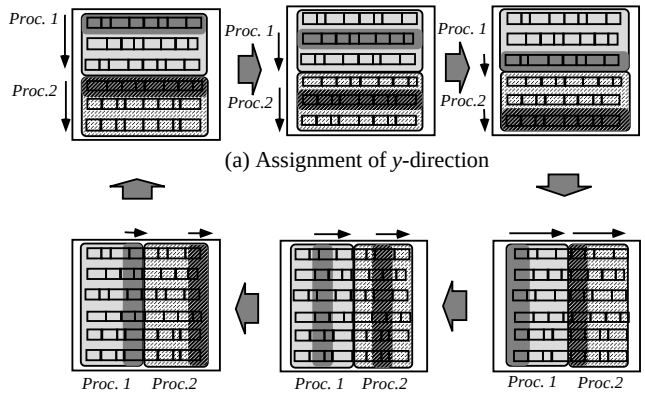
D. *Parallelization of Phase 3*

In phase 3, we introduce parallel processing to the assignments of some consecutive groups of cells to each processor, and perform each linear assignment independently by each processor as shown in Fig. 7. Now, let R be the number of cell rows. All cell rows are numbered from the top of the chip and let $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_R$ be the set of cells assigned to each cell row. Area is given to each cell, and let $a(\mathcal{M})$ be the sum of area of the cells in $\mathcal{M}$. The width of the chip is determined by the width of the longest cell row, and if the width of all rows are same, then the width of the chip is minimized. Thus we give the same capacity, denoted A, to all cell rows, and the sum of area of cells assigned to each cell row $a(\mathcal{R}_i)$ must be satisfied $a(\mathcal{R}_i) \leq A,\ i = 1, \cdots, R$.

First, in order to solve a set of linear assignment problems independently by each processor, we equally assign each processor $P_j,\ j = 1, 2, \cdots, P$ to a set of consecutive cell rows $\{\mathcal{R}_{\frac{R}{P}\times k+1}, \mathcal{R}_{\frac{R}{P}\times k+2}, \cdots, \mathcal{R}_{\frac{R}{P}\times(k+1)}\}$, $(k = 0, 1, \cdots, P-1)$, which is processed by the processor P_j, as shown in Fig.8.

Next, in order to split a set of linear assignment problems, a set of pairs $(\mathcal{G}_i, \mathcal{S}_i)$ are constructed where $\mathcal{S}_i, i = 1, 2, \cdots, R$ is a set of slots in which all cells in $\mathcal{G}_i$ are assigned and $\mathcal{G}_i, i = 1, 2, \cdots, R$ is a set of groups of cells, respectively. For each $\mathcal{G}_i,\ i = 1, 2, \cdots, R$, they have capacities $a(\mathcal{G}_i)$, which are,

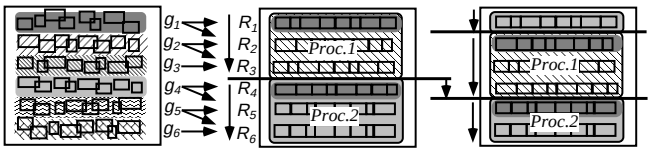
$$a(\mathcal{G}_i) = \begin{cases} \frac{3}{2}A & (\text{if } i = \frac{R}{P} \times k + 1, (k = 0, 1, \cdots, P-1)\ ) \\ \frac{1}{2}A & (\text{else if } i = \frac{R}{P} \times k, (k = 1, 2, \cdots, P)\ ) \\ A & (\text{otherwise}) \end{cases}$$



(a) Assignment of *y*-direction

(b) Assignment of *x*-direction

Fig. 7. Parallelization of phase 3.



(a) Grouping (b) First loop (c) Next loop

Fig. 8. Shifting of the group boundaries.

where P and R are the number of processors and cell rows, respectively. We construct the each cell group as follows. All cells are sorted by their y-coordinates, and the first cells, of which the sum of the area is equal to the capacity of the first group $a(\mathcal{G}_1)$, are assigned to the first group. Similarly, the remaining cells are divided into the groups. Then, each cell group is assigned to the corresponding processor as follows. Groups $\mathcal{G}_1, \cdots, \mathcal{G}_{\frac{R}{P}}$ are assigned to processor P_1, groups $\mathcal{G}_{\frac{R}{P}+1}, \cdots, \mathcal{G}_{2\frac{R}{P}}$ are assigned to processor P_2, and so on (Fig. 8).

Next, the cells of each cell group $\mathcal{G}_i$ are assigned to slots in each cell row $\mathcal{R}_i$ by the timing-driven linear assignment method proposed in [13] in parallel, that is, the cells of groups $\mathcal{G}_1, \mathcal{G}_{\frac{R}{P}+1}, \cdots, \mathcal{G}_{\frac{R}{P}(P-1)+1}$ are assigned to each slots concurrently. In the large circuits, these groups assigned concurrently are apart from each other, then we can reduce the effect of other processors in the cost matrix calculation [2]. Because $a(\mathcal{G}_i) \geq A$, if the cells in $\mathcal{G}_i$ which are not assigned to slots of the cell row i but slots of the cell row $i+1$, then the cells in $\mathcal{R}_{i+1}$ are added to the next group $\mathcal{G}_{i+1}$ and reassigned in the next linear assignment problem.

After the row assignment of y-direction, the row assignment of x-direction is performed in the similar way of y-direction. These assignments of x and y-directions are iteratively performed until there is no more improvement of placement or *LoopNumber* is reached to some preset value, and we obtain a final placement. In each iteration of outer loop, we shift sets of cell groups for processor P_i as shown in Fig. 8, because, in each iteration, if we use same sets of cell groups, then the cells in the last cell group of the set of cell group for processor P_i will be only placed at the same cell row in the further loops, resulting that these cells cannot move to the next cell row and this degrades the quality of the placement. The timing-driven row assignment algorithm is as follows.

[The Timing-Driven Row Assignment Algorithm]

Step 1: *LoopNumber* = 0.

/* Row assignment of y-direction */

Step 2: Construct cell groups $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_R$

Step 3: Assign cell groups to processors.

Step 4: For each processor P_i in parallel

Step 4.1: j = the index of the first cell group assigned to processor P_i.

Step 4.2: For all cells $m_k \in \mathcal{G}_j$, $s_l \in \mathcal{S}_j$, compute c_{kl} and solve the linear assignment problem LAP(j) [3].

Step 4.3: For all cells $m_k \in \mathcal{R}_j$, update their coordinates.

Step 4.4: $\mathcal{G}_{j+1} = \mathcal{G}_{j+1} \cup \mathcal{R}_{j+1}$.

Step 4.5: If $j <$ the index of the last cell group assigned to processor P_i, **then** $j = j + 1$ **go to Step 4.2**.

/* Row assignment of x-direction */

Step 5: Construct cell groups $\mathcal{G}'_1, \mathcal{G}'_2, \cdots, \mathcal{G}'_R$

Step 6: Assign cell groups to processors.

Step 7: For each processor P_i in parallel

Step 7.1 j = the index of the first cell group assigned to processor P_i.

Step 7.2 For all cells $m_k \in \mathcal{G}'_j$, $s_l \in \mathcal{S}'_j$, compute c_{kl} and solve the linear assignment problem LAP(j).

Step 7.3 For all cells $m_k \in \mathcal{R}'_j$, update their coordinates.

Step 7.4 $\mathcal{G}'_{j+1} = \mathcal{G}'_{j+1} \cup \mathcal{R}'_{j+1}$.

Step 7.5 If $j <$ the index of the last cell group assigned to processor P_i, **then** $j = j + 1$ **go to Step 7.2**.

Step 8: If there is no improvement or *LoopNumber = (preset number)*, **then** terminate, **else** Shift the cell groups of the both directions and *LoopNumber* = *LoopNumber* + 1 **go to Step 2**.

IV Experimental Results

We have implemented the proposed algorithm called ParPOPINS on a SPARC server 1000 of Sun Microsystems Inc. (135MIPS × 4CPU, 256MByte main memory) in C language with SunOS5.4 multi-thread library, and performed some experiments. In Table I, we show the example data of the experiments. Data "primary2", "avqs" and "avql" are the benchmark data distributed from MCNC and others are ISCAS benchmark data. For ISCAS benchmarks, logic synthesis and technology mapping were performed by SIS1.2 [18]. In this table, "#cons" is the number of timing constraints. As the timing constraints, we gave a clock cycle time for each data, which was determined by that of the placement produced without timing constraints multiplied by $0.8 \sim 0.9$.

[2] This cost matrix calculation is performed before solving linear assignment problems.

[3] LAP(j) is a linear assignment problem of the $\mathcal{G}_j$ which is to determine the slot assignment of all cells in $\mathcal{G}_j$ which minimizes $\sum_{m_j \in \mathcal{G}_i} \sum_{s_k \in \mathcal{S}_i} c_{jk} z_{jk}$ subject to $\sum_{s_k \in \mathcal{S}_i} z_{jk} = 1 (\forall m_j \in \mathcal{G}_i)$, $\sum_{m_j \in \mathcal{G}_i} z_{jk} = 1 (\forall s_k \in \mathcal{S}_i)$, and $z_{jk} \geq 0, \forall m_j \in \mathcal{G}_i, \forall s_k \in \mathcal{S}_i$ where $\mathcal{S}_i$ is a set of slots in which all cells in $\mathcal{G}_i$ are assigned, and c_{jk} is a cost with which the cell m_j is assigned to the slot s_k.

TABLE I
CHARACTERISTICS OF EXPERIMENTAL DATA.

Data	*#cells*	*#nets*	*#I/O*	*#rows*	*#cons*
C5	1081	1560	301	13	334
C6	1037	1516	301	14	334
C7	2150	2678	315	18	405
s15850	3228	3332	227	22	1105
s38417	7572	7734	134	37	2619
s38584	8964	9344	342	46	2729
avqs	21854	22124	64	80	6064
avql	25114	25384	64	86	6064

#cons : the number of timing constraints.

TABLE II
THE RESULTS OF PHASE1 FOR PAR-POPINS AND POPINS.

Data	*Method*	*#vio*	D_{max}	*length*[λ]	*time*[*sec*]
C6	Par-POP	4	1.02	1311166(0.98)	15(1.36)
	POPINS	2	1.03	1338810(1.00)	11(1.00)
C7	Par-POP	31	1.28	2068333(1.23)	46(1.53)
	POPINS	1	1.00	1679436(1.00)	30(1.00)
s38584	Par-POP	22	1.08	8073146(0.98)	790(0.72)
	POPINS	16	1.10	8245132(1.00)	1102(1.00)
avqs	Par-POP	42	1.26	25493801(0.98)	6934(0.41)
	POPINS	33	1.17	26014083(1.00)	17069(1.00)
avql	Par-POP	30	1.21	26180577(0.92)	8491(0.45)
	POPINS	1	1.00	28457150(1.00)	18954(1.00)

In order to evaluate the parallelization of each phase, we compare the results of each phase of Par-POPINS with that of POPINS. First, we compared the results of phase 1 of Par-POPINS with that of POPINS. Table II shows the results for typical data. In Table II, "*#vio*" is the number of violated timing constraints. "D_{max}", defined as $D_{max} = \max_{\forall t_{tau} \in \mathcal{T}} D_{act_\tau} / D_{allow_\tau}$, is called *the maximal violation ratio*, and if it is less than or equal to 1.0, the placement satisfies all timing constraints. "*length*" is the total wire length estimated by the Manhattern distance (λ). "*time*" is the running time by SPARCserver1000 (seconds). From Table II, the average total wire length of Par-POPINS is comparable to that of POPINS, but the number of timing violations of Par-POPINS is larger than that of POPINS. For computation time, Par-POPINS was 1.2 times faster on average and 2.4 times faster in maximum than POPINS. We can therefore show that the parallelization of phase 1 is suitable for the large scale circuits.

Next, Table III shows the results of phase 2 of Par-POPINS and POPINS for typical data. In these experiments, we used the results of phase 1 of POPINS as the inputs of phase 2 of Par-POPINS and POPINS. From Table III, Par-POPINS produces the comparable results to the results of POPINS for the total wire length. For the computation time, Par-POPINS takes longer than that of POPINS for "avqs". Since Par-POPINS were not able to remove 16 timing violations in "avqs" efficiently, Par-POPINS tried to improve the placement repeatedly and the computation time was hence longer than POPINS. For all data, Par-POPINS is 1.7 times faster on average and 4.7 times faster in maximum than POPINS, but Par-POPINS is 2.6 times faster on average except for "avqs". We can therefore show that the parallelization of phase 2, which solves the nonlinear programming problems for each target sub-circuit by several processors in parallel, is effective to reduce the computation time without degradation of results.

TABLE III
THE RESULTS OF PHASE2 FOR PAR-POPINS AND POPINS.

Data	*Method*	*#vio*	D_{max}	*length*[λ]	*time*[*sec*]
C6	Par-POP	0	0.96	1271435(0.97)	109(0.37)
	POPINS	0	0.96	1313270(1.00)	298(1.00)
C7	Par-POP	0	0.95	1665541(0.99)	1147(0.29)
	POPINS	0	0.97	1680488(1.00)	3969(1.00)
s38584	Par-POP	0	0.98	8101144(0.99)	624(0.21)
	POPINS	0	0.82	8143268(1.00)	2979(1.00)
avqs	Par-POP	16	1.10	25936044(1.01)	1250(2.47)
	POPINS	0	0.98	25737577(1.00)	506(1.00)
avql	Par-POP	1	1.00	28461112(1.00)	753(0.55)
	POPINS	1	1.00	28457150(1.00)	1368(1.00)

TABLE IV
THE RESULTS OF PHASE3 FOR PAR-POPINS AND POPINS.

Data	*Method*	*#vio*	D_{max}	*length*[λ]	*time*[*sec*]
C6	Par-POP	0	0.99	1023204(1.02)	59(0.30)
	POPINS	0	0.90	1003291(1.00)	196(1.00)
C7	Par-POP	0	0.94	1406502(0.98)	189(0.32)
	POPINS	0	0.91	1428595(1.00)	601(1.00)
s38584	Par-POP	0	0.82	7439664(1.04)	1869(0.39)
	POPINS	0	0.82	7168711(1.00)	4820(1.00)
avq-small	Par-POP	0	0.84	22812721(0.94)	28139(0.20)
	POPINS	0	0.77	24221879(1.00)	138702(1.00)
avq-large	Par-POP	0	0.94	27856918(1.23)	103992(0.41)
	POPINS	0	0.83	22551201(1.00)	247999(1.00)

Next, we show the results of phase 3 of Par-POPINS and POPINS in Table IV. We also used the results of phase 2 of POPINS as the inputs of phase 3 of Par-POPINS and POPINS. From Table IV, Par-POPINS can satisfy all timing constraints for all benchmark data. For computation time, Par-POPINS is 3.4 times faster on average and 5.0 times faster in maximum than POPINS. For the total wire length, the results of Par-POPINS is 4.0% longer on average than that of POPINS. For "avql", Par-POPINS were not able to produce the comparable result to POPINS. We have not yet completely analyzed this case, but the reason may be as follows. Although the groups assigned concurrently are apart from each other, the case may occur that the cells have connection with other cells that belongs to other groups assigned concurrently. In this case, since the errors of estimating the wire length of nets have occurred, this degrades the quality of the placement. We believe that this can be improved by decreasing the number of cells which are assigned in a row simultaneously.

Finally, we compared Par-POPINS with POPINS and RITUAL [19]. RITUAL is one of the most powerful performance driven placement algorithms which can satisfy a given clock cycle. The interconnection delay model is similar to ours, except that the wire resistance is not assumed. But to compare with our results, we evaluated the results of RITUAL by our model. Table V shows the results of Par-POPINS, POPINS, and RITUAL. For "s15850" and "s38584", RITUAL was not able to produce the placement due to errors. Moreover, since RITUAL which we used can not permit a cell which has more than one output pins, we could not test "avqs" and "avql". In these experiments, the results of phases 1 and 2 of Par-POPINS were used for the inputs of phases 2 and 3 of Par-POPINS, respectively. From Table V, Par-POPINS is 2.2 times faster on average and 3.3 times faster in maximum than POPINS within 13.9% degradation of the total wire length on average. More-

TABLE V
THE PLACEMENT RESULTS OF PAR-POPINS, POPINS2.0, AND RITUAL.

Data	*Method*	*#vio*	D_{max}	*length*[λ]	*time*[*sec*]
C5	Par-POP	0	0.94	1212371(1.13)	178(0.30)
	POPINS	0	0.89	1074034(1.00)	601(1.00)
	RITUAL	6	1.03	1450583(1.35)	1255(2.09)
C6	Par-POP	0	0.95	1213599(1.21)	191(0.38)
	POPINS	0	0.90	1003291(1.00)	505(1.00)
	RITUAL	0	0.95	1303082(1.30)	1362(2.70)
C7	Par-POP	0	0.96	1890354(1.32)	1612(0.35)
	POPINS	0	0.91	1428595(1.00)	4600(1.00)
	RITUAL	2	1.01	1957200(1.37)	1755(0.38)
s15850	Par-POP	0	0.89	2602893(1.17)	3114(0.38)
	POPINS	0	0.88	2216362(1.00)	8260(1.00)
	RITUAL†	-	- - -	- - - - (- - -)	- - - - (- - -)
s38417	Par-POP	0	0.96	7165758(1.02)	2405(0.47)
	POPINS	0	0.86	7058243(1.00)	5094(1.00)
	RITUAL	16	1.53	8002986(1.13)	14426(2.83)
s38584	Par-POP	0	0.85	9209736(1.28)	6115(0.69)
	POPINS	0	0.82	7168711(1.00)	8901(1.00)
	RITUAL†	-	- - -	- - - - (- - -)	- - - - (- - -)
avq-small	Par-POP	0	0.77	22983119(0.95)	64296(0.41)
	POPINS	0	0.77	24221879(1.00)	156277(1.00)
	RITUAL†	-	- - -	- - - - (- - -)	- - - - (- - -)
avq-large	Par-POP	0	0.88	28375840(1.26)	107222(0.39)
	POPINS	0	0.83	22551201(1.00)	268321(1.00)
	RITUAL†	-	- - -	- - - - (- - -)	- - - - (- - -)

†No results obtained.

over, Par-POPINS improved the total wire length by a 9.3% on average and a 16.4% in maximum compared with RITUAL. For computation time, Par-POPINS is 2.9 times faster on average than RITUAL.

V CONCLUSIONS

In this paper, we proposed a parallel algorithm for timing-driven standard cell placement based on POPINS2.0 [13]. The proposed algorithm can satisfy the performance requirement of the circuit by satisfying the timing constraints. And in the proposed method, to estimate the interconnection delay accurately we adopted the delay estimation model based on Elmore's delay model. To obtain high efficiency of parallel processing, we restrict communication among processors as much as possible by partitioning the placement problem to some sub-problems which can be executed independently. From the experimental results comparing with POPINS2.0, the proposed parallel algorithm, Par-POPINS, produces a placement 2.2 times faster on average and 3.3 times faster in maximum on 4 processors within small degradation of the total wire length. Future research includes the further reduction of computation time and the improvement of quality of placement results. Development of parallel placement algorithms for other parallel architectures is another interesting topic.

REFERENCES

[1] M. Burstein and M. N. Youssef: "Timing influenced layout design," Proc. of 22nd Design Automation Conference, pp. 124–130 (1985).

[2] W. E. Donath, R. J. Norman, B. K. Agrawal, S. E. Bello, S. Y. Han, J. M. Kurtzberg, P. Lowy and R. I. MeMillan: "Timing driven placement using complete path delays," Proc. of 27th Design Automation Conference, pp. 84–89 (1990).

[3] A. E. Dunlop, V. D. Agrawal, D. N. Deutsh, M. F. Jukl, P. Kozak and M. Wiesel: "Chip layout optimization using critical path weighting," Proc. of 21st Design Automation Conference, pp. 133–136 (1984).

[4] W. C. Elmore: "The transient response of damped linear networks with particular regard to wideband amplifiers," J. Appl. Phys., Vol.19, pp. 55–63 (1948).

[5] C. M. Fiduccia and R. M. Mattheyses: "A linear-time heuristic for improving network partitions," Proc. of 19th Design Automation Conference, pp. 175–181 (1982).

[6] T. Gao, P. M. Vaidya and C. L. Liu: "A new performance driven placement algorithm," Proc. of International Conference on Computer-Aided Design, pp. 44–47 (1991).

[7] T. Hamada, C.-. K. Cheng and P. M. Chau: "Prime: A timing-driven placement tool using a piecewise linear resistive network approach," Proc. of 30th Design Automation Conference, pp. 531–536 (1993).

[8] T. Hasegawa: "A new placement algorithm minimizing path delay," Proc. of International Symposium on Circuits and Systems, pp. 2052–2055 (1991).

[9] P. S. Hauge, R. Nair and E. J. Yoffa: "Circuit placement for predictable performance," Proc. of International Conference on Computer-Aided Design, pp. 88–91 (1987).

[10] M. Igusa, M. Beardslee and S. Sangiovanni-Vincentelli: "ORCA: A sea-of-gates place and route system," Proc. of 26th Design Automation Conference, pp. 122–127 (1989).

[11] M. A. B. Jackson and E. S. Kuh: "Performance-driven placement of cell based IC's," Proc. of 27th Design Automation Conference, pp. 370–375 (1989).

[12] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich: "GORDIAN: VLSI placement by quadratic programming and slicing optimization," IEEE Trans. Comput.-Aided Design of Integrated Circuits & Syst., Vol. 10, No. 3, pp. 356–365 (1991).

[13] T. Koide, M. Ono, S. Wakabayashi, Y. Nishimaru and N. Yoshida: "A new performance driven placement method with the Elmore delay model for row based VLSIs," Proc. of Asia and South Pacific Design Automation Conference, pp. 405–412 (1995).

[14] E. S. Kuh and M. Shih: "Recent advances in timing-driven physical design," Proc. of Asia-Pacific Conference on Circuits and Systems, pp. 23–28 (1992).

[15] M. Marek-Sadowska and S. P. Lin: "Timing driven placement," Proc. of International Conference on Computer-Aided Design, pp. 94–97 (1989).

[16] Y. Ogawa, M. Pedram and E. S. Kuh: "Timing-driven placement for general cell layout," Proc. of International Symposium on Circuits and Systems, pp. 872–875 (1990).

[17] B. M. Riess and G. G. Ettelt: "SPEED: Fast and efficient timing driven placement," Proc. of International Conference on Computer-Aided Design, pp. 377–380 (1995).

[18] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldabha, H. Savoj, P. R. Stephan, R. K. Brayton and A. Sangiovanni-Vincentelli: "SIS: A system for sequential circuit synthesis," Technical Report No. UCB/ERL M92/41, University of California, Berkeley (1992).

[19] A. Srinivasan, K. Chaudhary and E. S. Kuh: "RITUAL: A performance-driven placement algorithm," IEEE Trans. on Circuits and Systems II, Vol. 39, No. 11, pp. 825–839 (1992).

[20] S. Sutanthavibul and E. Shragowitz: "An adaptive timing-driven placement for high performance VLSI's," IEEE Trans. Comput.-Aided Design of Integrated Circuits & Syst., Vol. 12, No. 10, pp. 1488–1498 (1993).

[21] M. Terai, K. Takahashi and K. Sato: "A new min-cut placement algorithm for timing assurance layout design meeting net length constraint," Proc. of 27th Design Automation Conference, pp. 96–102 (1990).

[22] S. Wakabayashi, H. Kusumoto, H. Mishima, T. Koide and N. Yoshida: "Gate array placement based on mincut partitioning with path delay constraints," Proc. of International Symposium on Circuits and Systems, pp. 2059–2062 (1993).