A Loosely Coupled Parallel Algorithm for Standard Cell Placement

Wern-Jieh Sun and Carl Sechen
Department of Electrical Engineering
University of Washington
Seattle, Washington 98195

Abstract

We present a loosely coupled parallel algorithm for the placement of standard cell integrated circuits. Our algorithm is a derivative of simulated annealing. The implementation of our algorithm is targeted toward networks of UNIX workstations. This is the very first reported parallel algorithm for standard cell placement which yields as good or better placement results than its serial version. In addition, it is the first parallel placement algorithm reported which offers nearly linear speedup, in terms of the number of processors (workstations) used, over the serial version. Despite using the rather slow local area network as the only means of interprocessor communication, the processor utilization is quite high, up to 98% for 2 processors and 90% for 6 processors. The new parallel algorithm has yielded the best overall results ever reported for the set of MCNC standard cell benchmark circuits.

1. Introduction

Nearly ten years ago, an early implementation of simulated annealing for standard cell placement yielded particularly promising results[23]. However, it was many times slower than previously known methods. Therefore, this spawned considerable research efforts into developing a parallel implementation of simulated annealing for row-based placement. These previous methods can be categorized by the type of parallel hardware employed as well as the interprocessor communication schemes used. These methods were based on: 1) a shared-memory architecture [2][3][14], 2) both a shared-memory and dedicated communication channels architecture[18][19], 3) a hypercube machine [1][8][21], 4) a massively parallel machine[4][30], and 5) a network of workstations[9][10].

In every case, these previous methods failed to yield results as good as those produced by the state-of-the-art serial standard cell placement algorithm available at the time[23][24][26][27]. In fact, the results yielded by these prior methods diverged appreciably as the size of the placement problem instances grew. Worse yet, these methods generally had difficulty even handling the very large problem instances. And it's exactly those large problems for which one would want to have a parallel implementation available. Another problem which diminished the utility of the prior methods is that they achieved substantially less than linear speedup as a function of the number of processors employed. Worse still, most of these methods were only applicable to very expensive hardware.

As a consequence of these problems, no parallel implementation of simulated annealing has ever been used in industry. It would therefore be of considerable interest if there existed a coarse-grained, parallel standard cell placement algorithm which ran on a standard network of low-cost workstations and which yielded results at least equivalent to the serial version of the same algorithm, and in addition, offered nearly linear speedup with the number of processors used. Furthermore, the new parallel method would have to produce results at least as good as those ever reported for the widely adopted set of benchmark circuits available from MCNC.

In fact, we report precisely such a parallel placement algorithm in this paper, which is organized as follows. In Section 2 we first review the previous parallel approaches developed for standard cell placement. In Section 3, we very briefly touch on the key aspects of the serial placement algorithm, derived from simulated annealing, from which we began the development of our new parallel algorithm. Our new parallel placement algorithm is the subject of Section 4. We analyze the performance of the algorithm in Section 5. Finally, in Section 6 we present our results.

2. Previous Work

Among the first reported parallel algorithms for standard cell placement were those by Rutenbar and Kravitz[14][20]. They proposed two shared memory multiprocessor simulated annealing algorithms. One was based on move decomposition and the other on parallel moves. A move could be either the displacement of a single cell or the exchange of two cells. In the first approach, a move was decomposed into several sub-tasks and parallelism was exploited in these sub-tasks. This scheme was only able to exploit limited parallelism. In addition, the synchronization of these sub-tasks had to be performed very carefully. In [14], a speedup of 2 was reported by using 3 processors. The speedup increased only slightly with the introduction of additional processors.

In their parallel moves approach, a *serializable* subset is defined as a subset of all non-interacting moves that can be carried out serially in any order. Thus, all moves in this subset can be carried out in parallel and the outcome would be the same as if the moves had been executed and evaluated serially. An advantage of this approach is that the convergence behavior of simulated annealing is the same as that for the serial version[16]. However, it turns out that finding serializable sets is extremely difficult. In fact, the authors resorted to the simplest form of this approach which is based on attempting and evaluating a group of moves in parallel, and then actually performing the single move that is accepted first (if any) and then aborting all the other attempts. As noted in [3], this approach unfortunately puts a bias in favor of moves that required less computation time. It turned out that this method showed a linear speedup with the number of processors for the lowest temperatures in the annealing schedule, where few moves are accepted, but performed poorly at higher temperatures.

Rose, *et al.* [18][19] proposed a method called heuristic spanning to replace the high temperature regime in simulated annealing. Parallelism was obtained by executing a mincut algorithm with different starting partitions on different processors. After collecting the results generated by the various processors, the one with the lowest cost is chosen to enter the low temperature annealing phase. This phase is accelerated by mapping different regions of the chip onto different processors. Every processor performed moves in parallel and then broadcast the new cell locations to the other processors. To avoid idle processors, asynchronous moves were exploited at the cost of introducing error in the wire length calculations since processors often had different views on the current position of some portion of the cells. The processor utilization efficiency was estimated to be 70%.

In [4] and [30] there were attempts to implement simulated

© 1994 ACM 0-89791-690-5/94/0011/0137 $3.50

annealing on a massively parallel machine, such as the Connection Machine. The data structures for cells, nets and pins are distributed over many processors. The maximum circuit size that can be processed is then limited by the number of processors available. In [4], it was reported than even on a 65,536 processor Connection Machine, a maximum of only 8000 cells could be handled. The processor utilization was also very low. There were no experiments showing that this approach can achieve the same high quality results produced by the sequential simulated annealing algorithm.

Banerjee, *et al.* [1][8][21] presented parallel algorithms for standard cell placement on a hypercube multiprocessor. The cells to be placed are mapped onto all of the processors in the hypercube. The mapping was either done in grid-like fashion[1][8] or in row-based fashion [21]. Due to the hypercube structure, only processors with a particular address pattern are allowed to perform cell interchanges. This rather restricts cell movement. To avoid error accumulation[21], sets of non-interacting cells are identified and only cells in the same sets are allowed to be exchanged. This further impeded cell movement. This algorithm performed and evaluated moves in parallel on all processors. After each move, the new cell position was broadcast to all processors. The global synchronization scheme was rather expensive. In [21], the expected speedup was 11-21 using 64 processors.

Of all the parallel implementations of simulated annealing, those which perform moves in parallel attract the most attention because they have the most potential for appreciable speedups. Parallelism can be highly exploited by performing moves on all of the processors at the same time. All of the previous methods required frequent application of some form of global synchronization. They either broadcast new information after each move or after fewer than 10 moves. This rather high communication cost restricted its application to only special parallel architectures that can provide high communication throughput in the form of either shared memory, a hypercube interconnection network or other dedicated hardware.

There has been a small amount of previous work on developing standard cell placement implementations for loosely coupled parallel processing environments, in particular, consisting of networks of low-cost workstations. Mohan, *et al*., [17] presented a placement approach based on the genetic algorithm. Also, Kling, *et al*., [12] presented a method based on simulated evolution. However, neither of these approaches were shown to be comparable in terms of placement quality with the state-of-the-art serial implementation of simulated annealing. Finally, Banerjee, *et al.*, have developed a parallel version of simulated annealing for this loosely coupled environment[9][10]. They based their approach, which they called ProperPLACE, on TimberWolfSC version 6.0. Unfortunately, the results produced by ProperPLACE diverged badly as circuit sizes increased in comparison to TimberWolfSC 6.0. In addition, the speedup as a function of the number of processors was far from linear.

As a consequence of the limitations of the previous approaches to parallel placement, particularly parallel implementations of simulated annealing, none of these methods has ever been used in industry. However, by no means does this imply that faster placement isn't badly needed. Although the state-of-the-art serial version of simulated annealing for standard cells (TimberWolf 7.0 [26][27]) was reported to be dramatically faster than the previous version (6.0), in fact, seven to ten times faster for larger circuits, the computation time on a state-of-the-art workstation can still approach 24 hours for a 200,000-cell gate array. No other serial algorithm has been reported which can yield comparable results in less computation time.

It would therefore be of considerable interest if there existed a coarse-grained, parallel standard cell placement algorithm which ran on a standard network of low-cost workstations and which yielded results at least equivalent to TimberWolfSC version 7.0, and in addition, offered nearly linear speedup with the number of processors used. Furthermore, the new parallel method would have to produce results at least as good as those ever reported for the widely adopted set of benchmark circuits available from MCNC.

3. Serial Version of Simulated Annealing for Standard Cell Placement

We felt that we could develop the most effective loosely coupled parallel algorithm for standard cell placement by basing it on the most effective serial algorithm available, name TimberWolfSC 7.0. In this section we review the key aspects of this algorithm.

In [26][27] a new approach to simulated annealing and a new hierarchical algorithm for row-based placement was described. This implementation obtained the best results ever reported for the set of MCNC benchmark circuits. Chip area reductions up to 15% were achieved compared with TimberWolfSC v6.0. In addition, chip area reductions up to 21% were achieved while consuming up to 7.5 times less CPU time in comparison to TimberWolfSC v6.0. The new approach produced lower total wire length by an average of 8% than Gordian/Domino[5][6][11][25], 11% lower wire length than Ritual/Tiger [22], while using comparable run time.

```
 1. randomly select cell $a$
 2. randomly select row $r$ and location $x$ in $r$
 3. /* $x$ in $r$ is within the range limiter window span for $a$ [24] */
 4. if (adding $a$ to $r$ doesn't exceed length limit for $r$) then
 5.    compute $\Delta C$ for moving $a$ to location $x$ in $r$
 6. else
 7.    /* now consider an exchange of $a$ and $b$ */
 8.    note cell $b$ covering location $x$ in $r$
 9.    if (length limit of neither row is exceeded) then
10.       compute $\Delta C$ for exchanging $a$ and $b$
11.    else
12.       go to line 1
13. if ($accept(\Delta C)$) then
14.    eliminate overlaps for the row(s) of $a$ and $b$
15. update estimation model for $a$ and $b$
```

Figure 1 New State Generator

Figure 1 shows the algorithm used by TimberWolfSC 7.0 for generating new placement configurations. First, cell a is randomly selected. A single cell move is attempted if the target row's length limit is not exceeded. Otherwise, a cell b which covers the target location is noted and an interchange of a and b is attempted if no row length limits are violated. The length limit is set to be the smaller of one percent of the average row length or one average standard cell width. If a limit was violated, the new state generator begins anew. If a single cell or interchange move is feasible, then the change in cost is computed. The probability of accepting the new configuration is one if $\Delta C \leq 0$; otherwise, it is equal to $e^{(-\Delta C/T)}$, where T is the temperature. If the new configuration is accepted, the cells in the affected rows are shifted to avoid any cell overlapping. Every new configuration thus generated is legal and physically feasible.

$$\Delta C = \Delta W + \Delta W_S \qquad (1)$$

The incremental cost function used by TimberWolfSC 7.0 is given in (1). ΔW is the change in the net lengths for those nets connected to the cell (or two cells) participating in the single cell (or exchange) move. ΔW_S represents the change in the net lengths for those nets connected to the cells in the affected rows which must be shifted to avoid the creation of cell overlapping. When a new cell is inserted into a row, other cells in this row generally need to be shifted to accommodate the new cell. An effective and efficient method for estimating ΔW_S, the change in the net lengths for those nets connected to the shifted cells are given in [26][27].

When a cell moves from one row to another, the overall move distance will consist of two components, the *x* displacement and the *y* displacement. When a cell moves within the same row, there will only be an *x* displacement. The value of a *y* displacement will be at least the distance between the rows. On the other hand, the *x* displacement can be as small as one grid size. In a typical circuit, the distance between the rows is usually tens or hundreds of grid sizes. Hence the *y* displacement is generally several times larger than the *x* displacement. This phenomenon causes $|\Delta C|$ for an *inter-row* move, which has both *x* and *y* components, to generally be several times larger than for an *intra-row* move, which has only an *x* component.

When the simulated annealing acceptance criterion is used to evaluate moves, those moves that generate higher cost changes will generally have a much higher chance of being rejected. Thus, inter-row moves, which generate higher changes in cost, are much more likely to be rejected than intra-row moves at a given temperature. This was found to be especially severe when the temperature is low; primarily only intra-row moves were being accepted. This greatly hampered the search ability of simulated annealing since inter-row moves were not adequately exploited. When only inter-row moves were included in calculating the acceptance rate, the annealing schedule described in [26][27] yielded much better results.

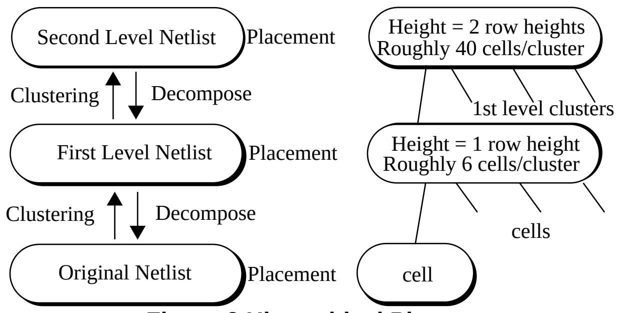

Figure 2 Hierarchical Placement

Figure 2 shows the hierarchical placement methodology described in [26][27], which combined a new clustering technique with the new approach to simulated annealing. The original netlist is hierarchically clustered into various levels of netlists. Then the new approach to simulated annealing is used to place those various levels of netlists.

In the clustering stages, the original netlist is condensed into the first and then the second level netlists. The produced clusters in the higher level netlists have similar size, which greatly aids the annealing placement stages. In the placement stages, the condensed second level netlist is placed using simulated annealing at the higher temperatures. Then the second level netlist is decomposed back to the first level netlist. Cells of the lower level netlist are randomly placed within the range of the cluster to which they belong in the higher level netlist. At the new lower level, these cells may then move outside the bounds of the higher level cluster. The first level netlist is then placed at the middle temperatures. Next, the first level netlist is decomposed back to the original netlist, and the original netlist is placed at the lower temperatures. There are two clustering stages and three placement stages. This combined clustering and simulated annealing methodology can be viewed as a combined bottom-up and top-down approach, where the two clustering stages provide bottom-up perspectives and the three placement stages provides a top-down view, from the course grain of the higher levels to the fine grain of the lower levels. Timing requirements are satisfied in all stages[29]. The right hand side of figure 2 shows some typical values.

The clustering technique is based on graph connectivity. It produces clusters with about the same size, and emphasizes nets with small fan-out. The technique was shown to yield good results using a linear (in terms of the number of cells) time implementation.

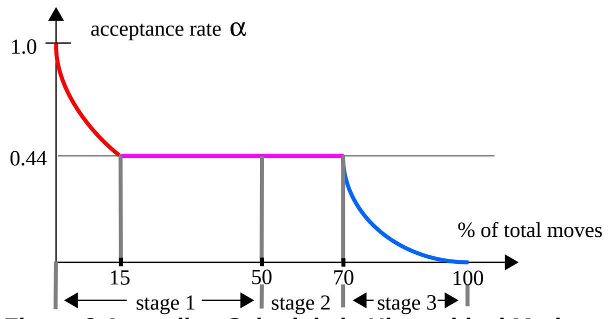

Figure 3 Annealing Schedule in Hierarchical Mode

As stated above, the first level netlist is placed in the higher temperature regime. This first stage comprises the first 50% of the total annealing schedule as shown in figure 3. The second placement stage starts at 50% and ends at 70% of the total annealing schedule as shown in figure 3. The constituent cells belonging to a cluster are randomly placed within the confines of the location of the bounding rectangle of the cluster as determined in the previous stage. The third stage starts at 70% of the total annealing schedule. The restarting temperature *T* of the second (and third) placement stage is given by (2), where $\overline{\Delta W}$ is the average net length and α is the target acceptance rate.

$$T = -\frac{\overline{W}}{\log \alpha} \tag{2}$$

The total number of moves (the *x* axis in figure 3) is divided into 150 *iterations*. Each *iteration* consists of a number of moves equal to the total number divided by 150. The total number of moves, *m*, is given by (3), where *n* is the number of cells in the circuit.

$$m = \begin{cases} 12500n & n \le 500 \\ 25n\left(\frac{n}{500}\right)^{\frac{1}{3}} & n > 500 \end{cases} \tag{3}$$

4. The New Parallel Placement Algorithm

4.1 Introduction

Given that our parallel processing hardware consists of a network of workstations, our objective was to achieve near linear speedup with the number of processors (workstations). There are two main obstacles: 1) how to absolutely minimize interprocessor communication since the local area network is quite slow, and 2) given that moves will be generated and evaluated in parallel, how to execute (generate and evaluate) these moves effectively in the presence of erroneous information on the location of some portion of the cells.

Communication between the processors can be reduced to zero by dividing the chip into *n* non-overlapping regions and assigning each of the *n* processors to a unique region. Each processor then optimizes the locations of the cells which were initially placed in its region. There are several critical problems with this approach. First, it is virtually impossible to initially assign all of the cells into their proper region. Partitioning algorithms which have been proposed to date have not been shown to be effective for this problem. Second, given that some cells will therefore not be in their optimal regions, there is no way for the cells to move between regions and to therefore improve their region assignment. Third, when optimizing the placement of the cells within its region, the processor has almost no idea of where the cells are in other regions. Given that there can be more than 10,000 cells in a region and that there can be a similar number of inter-region nets connecting these cells, this will be a very serious source of error when computing total wire lengths (the basic cost function in annealing-based placement algorithms).

In summary, a parallel implementation based on no interprocessor communication can be characterized as having virtually linear speedup with the number of processors, but will not yield state-of-the-art results because of the confinement of the cells to the original regions and because of the uncertainty in the positions of the cells outside of a given processor's region. In our new parallel algorithm, we retain the main advantage of the no-interprocessor-communication technique (virtually linear speedup) while avoiding its key disadvantages: 1) by permitting a small (almost negligible) amount of inter-processor communication and 2) by using a new dynamic region generation scheme.

4.2 Interprocessor Communication

In our approach, each processor maintains its own copy of the complete data structures for the entire placement problem. At the start of each iteration, the entire chip area is divided into a union of non-overlapping regions. Each processor is assigned to a unique region and this region will be termed the *active* region for that processor. All other regions are termed *inactive* with respect to that processor. A processor will only move those cells that are initially (*i.e.* at the start of the iteration) in its active region. It will never move a cell outside of its active region. Furthermore, a processor assumes that the positions of the cells in its inactive regions remain as they were at the beginning of the iteration. In the next section we will address the impact of this error.

Interprocessor communication only takes place at the end of each iteration (that is, 150 times during a placement run). Each processor broadcasts the positions of only those cells in its active region which have changed since the end of the previous iteration. Each processor also receives new position information for only those cells in its *inactive* regions which have moved since the last iteration.

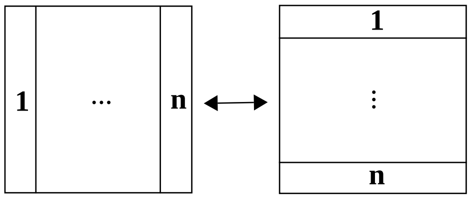

Figure 4 Dividing Chip Area

4.3 Dynamic Region Generation

During the course of an iteration, the cells in a processor's active region are not permitted to move outside that region. Therefore, badly misplaced cells (cells assigned to non-optimal regions) can seriously degrade the quality of the final placement. We found that how the chip area is divided into regions has a profound effect on the placement quality.

We discovered that the most effective approach is to dynamically generate the regions. In addition, we found that in order to get good placement results, it is very important that each processor be capable of generating and evaluating both long distance and short distance moves. We were able to satisfy these criteria with the following scheme. At the beginning of each odd numbered iteration, the chip area is divided into equal-size regions which consist of *n* vertical slices as illustrated on the left-hand side of figure 4. At the start of each even numbered iteration, division of the chip area into non-overlapping equal-size regions (now consisting of horizontal slices) takes place as shown on the right-hand side of figure 4. In either case, region *i* is the active region for processor *i*.

Note that this scheme permits a cell to move from its current position to anywhere else on the chip in at most two iterations. This is illustrated in figure 5, where cell *A* is currently in the upper left corner of the chip but should move to the lower right corner to maximally reduce the total wire length. During the current iteration (left-hand side of figure 5) *A* can move to the bottom of its region and then during the next iteration (right-hand side of figure 5) it can move to the extreme right of its region.

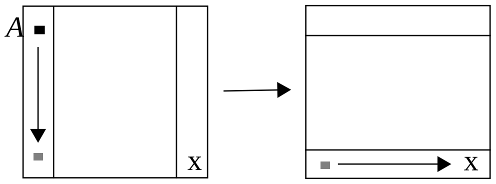

Figure 5 Cell Movement

4.4 New Parallel Placement Algorithm

```
 1. determine initial temperature
 2. set range limiter dimensions equal to chip dimensions
 3. start with a random initial placement of the cells
 4. **for** each iteration **do**
 5.   /* given $n$ processors */
 6.   divide chip into $n$ distinct regions
 7.   /* vertical slices on odd iterations */
 8.   /* horizontal slices on even iterations */
 9.   /* a processor can only move cells in its region */
10.   for each processor in its active region do
11.     generate a move
12.     determine whether to accept/reject
13.     /* see figure 1 for details */
14.     periodically adjust the temperature
15.     /* for each processor, force the actual acceptance
16.     /* rate to track the target rate shown in figure 3 */
17.   until the iteration is complete
18.   /* an iteration consists of a fixed no. of moves */
19.   /* an iteration is terminated for all processors as
20.   /* soon as one processor completes the iteration */
21.   each processor broadcasts its cell position changes
22.   reduce range limiter window dimensions
23.   set temperature to average temperature over all regions
24. until all iterations have been completed
```

Figure 6 Parallel Placement Algorithm

Figure 6 shows the overall pseudo code for our new parallel standard cell placement algorithm. Line 14 implies that each processor manipulates its own value of the temperature so that its actual acceptance rate follows the target rate shown in figure 3. The temperature is updated about every ten moves; it can be raised or lowered as necessary. Since each processors maintain its own temperature, the temperature will be a little different on the various processors. Because the regions change aspect ratios, each processor will generally get a largely different set of cells in its active region from one iteration to the next. Therefore at line 23 we set the initial temperature value for the next iteration equal to the average value over all regions at the end of the previous iteration. The range limiter window [23][28] in line 22 is used to generate moves more efficiently. In the early iterations, the window dimensions are very large, permitting cells to move any distance across the chip. As the iterations proceed, the window size is reduced so that we preferentially generate moves that have a non-negligible chance of being accepted.

$$P(x) = \frac{1}{2r} exp\left(\frac{-|x|}{r}\right) \quad (4)$$

The functional form of the range limiter that we use is given in (4), where r is the dimension of the window (either horizontally or vertically), x is the proposed move distance, and $P(x)$ is the probability that such an x will be generated. The window size r is initially set to the dimension of the entire chip and it is then decreased exponentially (with respect to the iteration number). It reaches its minimum dimension, about twice the average cell width, slightly more than half way through the annealing schedule. The range limiter plays an important role in our parallel algorithm.

5. Performance Evaluation and Error Analysis

5.1 Cell Mobility Analysis

In our approach, the chip area is divided into regions and the cells are not allowed to move between regions. A key concern is the impact this has on cell mobility. For example, if a long-range, out-of-a-region cell move was proposed (*e.g.* by a serial algorithm operating with a single region), it might be accepted by the serial algorithm but prevented by the parallel algorithm. If this happens frequently, it is quite apparent that the serial version will outperform the parallel algorithm.

The probability of a certain move distance is controlled solely by r in (4). The distances of more than 95% of the moves will lie within the interval $\pm 3r$. Therefore, if r is much greater than the size of the region, the cell mobility will be affected. On the other hand, if r is smaller than the size of the region, we contend that the cell mobility will not be affected.

To assess the parallel algorithm's impact on cell mobility, we noted how far each cell traveled during an iteration. We compiled the empirical data shown in figures 7, 8, 9 and 10 for one iteration consisting of one million moves, for a very large industrial circuit. The horizontal axis is the net distance that a cell moved during the iteration and the vertical axis is the percentage of the cells that moved that net distance (normalized in terms of average cell size). The range limiter dimension r (normalized the same way) is shown along the horizontal axis, as is the region size w. Note that in figure 7 (the net move distance in the x direction) and 8 (the net move distance in the y direction), $w < r$. As expected, the cell mobility profile for the parallel algorithm is noticeably different than that of the serial algorithm (using only a single region equal to the entire chip) caused by dividing the chip area into regions. Nonetheless, an important observation is that although there is distortion, there are still a substantial number of long-range moves (net cell moves that exceed the size of the region). This is attributed to the way we divide the chip area into regions (figure 4). In particular, there is always either a vertical or horizontal direction in which a cell can travel a long distance.

Figure 9 (net move distance in the x direction) and 10 (net move distance in the y direction) are for a later iteration when $w > r$. It is apparent that the cell mobility profiles are virtually identical in this case. This is an important result for the parallel algorithm. As long as $w > r$, the statistics of the net cell moves will not be affected, and therefore the quality of the optimization during these iterations should not be affected.

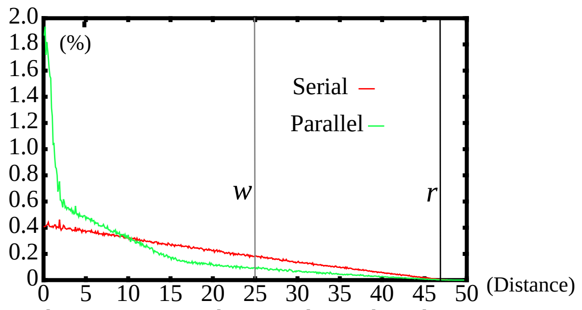

Figure 7 Move Distance in x Direction

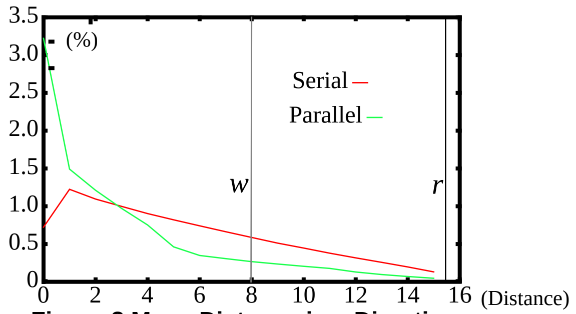

Figure 8 Move Distance in y Direction

As mentioned earlier, r is initially set to the chip dimensions and it is exponentially decreased (as a function of iteration) down to a minimum size of about two average cell widths just over half way through the annealing schedule. Two average cell widths is a very small distance, relative to the chip dimension, given the size of standard cell circuits today, which include up to 100,000 cells. Since r is exponentially decreased, this implies that $w > r$ for most of the iterations. This provides a rich opportunity to explore parallelism.

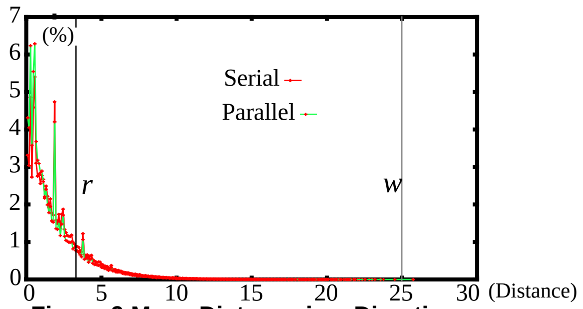

Figure 9 Move Distance in x Direction

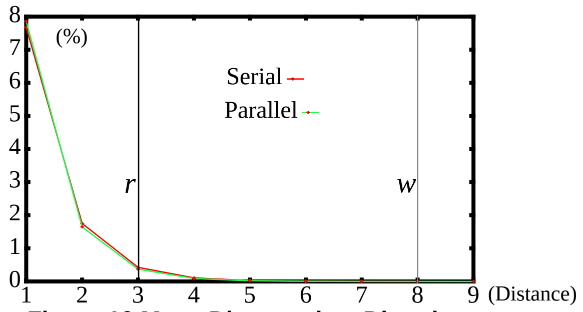

Figure 10 Move Distance in y Direction

5.2 Error Analysis

In the parallel algorithm, each processor evaluates cell moves within its active region assuming that the cells in the other regions do not change during the course of the iteration. This leads to error when a processor computes the change in wire length due to a move in its region. The errors are not eliminated until the global synchronization step of line 21 in figure 6 where each processor broadcasts (to all other processors) the positions of cells that have changed since the last update (iteration).

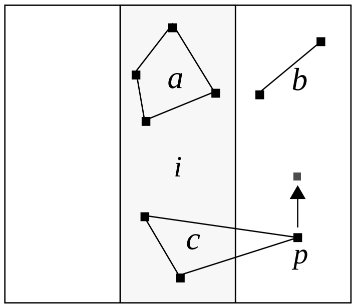

Figure 11 Internal and Crossover Nets

Figure 11 shows a scenario a processor might face in our parallel algorithm. The shaded area is the active region assigned to processor *i*. The processor can only move cells within its active region. Three nets are shown: net *a* connecting 4 cells, net *b* connecting 2 cells, and net *c* connecting 3 cells. We classify all of the nets into one of two categories: *internal* nets or *crossover* nets. A net whose span is entirely within one region is defined as an internal net; otherwise, it is a crossover net. In figure 11, nets *a* and *b* are internal nets while net *c* is a crossover net.

For all internal nets, a processor either knows exactly the span of the net, or it does not care about the net. In either case, no error will result. For example, net *a* in figure 11 is an internal net. All of its four cells are in the active region of processor *i*. This processor knows the *exact* locations of these four cells at all times because only it is allowed to move them. Therefore there will never be any error when processor *i* computes the change in length of net *a*. On the other hand, processor *i* does not care about net *b* since this net has no connections to cells in region *i*. Thus processor *i* will never be called upon to evaluate the change in *b*'s length and therefore no error will ever result.

Crossover nets, however, may lead to errors when a processor is computing the changes in wire length resulting from a cell move in its region. For example, net *c* in figure 11 is a crossover net. Cell *p* on net *c* is not in the active region of processor *i* and hence this processor does not know the exact location of this cell. Processor *i* assumes that cell *p* remains at the position it was at the beginning of the iteration even if it was moved upward by some other processor as indicated in figure 11. This source of inaccurate cell information causes error when processor *i* evaluates its moves. It is straightforward to experimentally ascertain the magnitude of this error. At the end of iteration *k*, just before each of the processors broadcast the updated cell positions to all of the other processors, we note C_k, a processor's view of the total length of all of the crossover nets and from this we subtract $\hat{C}_k$, the (exact) value after the broadcast. The percentage error E_k is then determined by dividing by W_k, the overall total wire length (*i.e.* the sum of the lengths of the internal and crossover nets after the broadcast), as shown in (5).

$$E_k = \frac{C_k - \hat{C}_k}{W_k} \qquad (5)$$

For the MCNC benchmark circuit *Primary1*, in figure 12 we plot E_k as a function of *k* for two different runs: one using two processors and the other using four processors. In each case, the total number of moves was approximately one million. Two observations are apparent: First, E_k for four processors has greater absolute value than for two processors. This is not surprising since dividing the chip area into four regions as opposed to two is certain to yield more crossover nets. Second, in the very beginning when virtually all moves are accepted, it is equally likely that a processor will over estimate as under estimate a crossover net's length. Thus the net percentage error is around zero. Since the number of long nets is large, leading to many crossover nets, the percentage error increases as the acceptance rate falls. It reaches its maximum value when the acceptance rate falls to around 50 percent (or about iteration 30). From then on the error decreases toward zero as the annealing schedule proceeds.

This decreasing trend is due to several factors: 1) Net lengths continually get smaller and therefore fewer and fewer crossover nets will exist as the iterations go by. 2) As the range limiter window dimensions decrease, the distance that cells move during an iteration continually decreases implying that a processor's view of the positions of cells in its inactive regions will be closer to reality as the iterations go by.

In any case, the percentage error at any iteration is under about two percent. And even this maximum error occurs only when the acceptance of *uphill* moves is prevalent. When the temperature gets low enough that primarily only *downhill* moves are accepted, then the percentage error is virtually zero. Hence we would expect the parallel algorithm to yield comparable results to the serial algorithm.

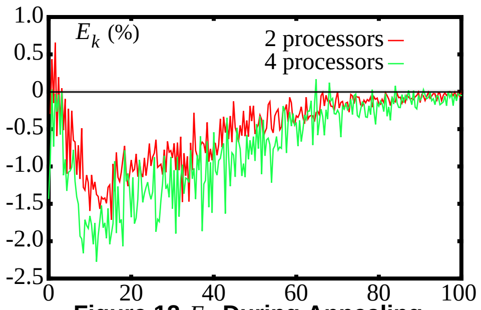

Figure 12 E_k **During Annealing**

Figures 13, 14 and 15 show results for E_k versus *k* for our parallel hierarchical placement approach. Figure 13 shows the result of E_k versus *k* for circuit *Biomed* using 4 processors. Figures 14 and 15 show the results of E_k versus *k* for circuit *Biomed* and *Avqsmall*, respectively, by using 6 processors. The results consistently show that there is some error in the first half of the annealing process and the error decreases toward zero in the second half. The maximum absolute value of E_k is less than 2%.

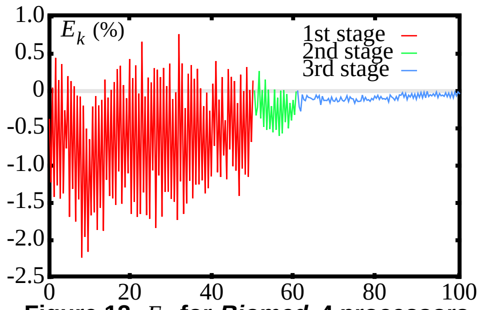

Figure 13 E_k **for *Biomed*, 4 processors**

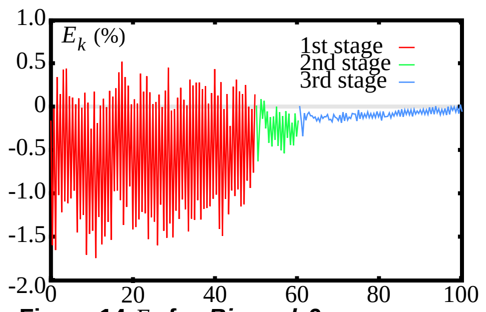

Figure 14 E_k **for *Biomed*, 6 processors**

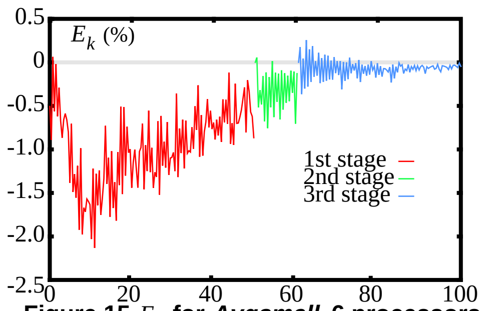

Figure 15 E_k **for *Avqsmall*, 6 processors**

6. Results

Table 1 shows the parameters of our test circuits. All circuits are from the 1993 MCNC layout benchmark set[13].

Circuit	# cells	# nets	# pins	# rows
Biomed	6417	5742	26947	46
Industry 2	12142	13419	125555	72
Industry 3	15059	21940	176584	54
Avqsmall	21854	22124	82601	80
Avqlarge	25114	25384	82751	86

Table 1: Test Circuits

6.1 Wire Length Results

Table 2, 3, and 4 compare the wire length results produced by our parallel algorithm with those generated by the serial program *TimberWolf v7*. The second column shows the wire length produced by *TimberWolf v7* for the five MCNC benchmark circuits. These wire lengths were the best among all previously published results. The 3rd, 5th and 7th columns of table 2 show the wire length produced by our parallel algorithm using 2, 4, and 6 processors. The results range from 1% worse to 5% better than the serial program *TimberWolf v7*. It is interesting to note that our parallel algorithm tends to produce even *better* results than the serial counterpart. This can be attributed to E_k, the errors introduced by our parallel algorithm (Section 5.2). This small random error actually allows our parallel algorithm to climb out of local minima and find better solutions.

Circuit	Wire *TW v7.0*	Wire *2 proc's*	Difference
Biomed	3.24	3.28	-1%
Industry 2	13.53	13.64	-1%
Industry 3	42.84	42.92	-0%
Avqsmall	5.41	5.32	2%
Avqlarge	5.86	5.57	5%
average	-	-	1%

Table 2: Wire Length Comparison
TimberWolf v7* vs. *Parallel 2 processors

Circuit	Wire *TW v7.0*	Wire *4 proc's*	Difference
Biomed	3.24	3.23	-0%
Industry 2	13.53	13.35	1%
Industry 3	42.84	43.08	-0%
Avqsmall	5.41	5.38	1%
Avqlarge	5.86	5.91	-1%
average	-	-	0%

Table 3: Wire Length Comparison
TimberWolf v7* vs. *Parallel 4 processors

Circuit	Wire *TW v7.0*	Wire *6 proc's*	Difference
Biomed	3.24	3.29	-1%
Industry 2	13.53	13.71	-1%
Industry 3	42.84	42.71	0%
Avqsmall	5.41	5.31	2%
Avqlarge	5.86	5.84	0%
average	-	-	0%

Table 4: Wire Length Comparison
TimberWolf v7* vs. *Parallel 6 processors

6.2 Run Time Speed Up

Table 5, 6, and 7 show the run time speed up gained by our parallel algorithm. All reported times are elapsed times (including CPU times and interprocessor communication times) in seconds on DEC alpha workstations 3000/400. The results show that our parallel algorithm achieves near linear speed up.

Circuit	Time *TW v7.0*	Time *2 proc's*	Speed Up
Biomed	1327	668	1.98
Industry 2	4928	2514	1.96
Industry 3	6756	3385	1.99
Avqsmall	6881	3578	1.92
Avqlarge	8252	4194	1.97
average	-	-	1.96

Table 5: Speed Up
TimberWolf v7* vs. *Parallel 2 processors

Circuit	Time *TW v7.0*	Time *4 proc's*	Speed Up
Biomed	1327	380	3.49
Industry 2	4928	1306	3.77
Industry 3	6756	1765	3.83
Avqsmall	6881	1755	3.92
Avqlarge	8252	2127	3.88
average	-	-	3.78

Table 6: Speed Up
TimberWolf v7* vs. *Parallel 4 processors

Circuit	Time *TW v7.0*	Time *6 proc's*	Speed Up
Biomed	1327	305	4.35
Industry 2	4928	847	5.81
Industry 3	6756	1188	5.68
Avqsmall	6881	1279	5.38
Avqlarge	8252	1559	5.29
average	-	-	5.30

Table 7: Speed Up
TimberWolf v7* vs. *Parallel 6 processors

Figure 16 plots the speed up achieved by our parallel algorithm. Although our target parallel environment (a network of workstations) only provides limited communication bandwidth, our parallel algorithm still achieves near linear speedup, especially for larger circuits. And it's exactly those large problems for which one would want to have a parallel implementation available.

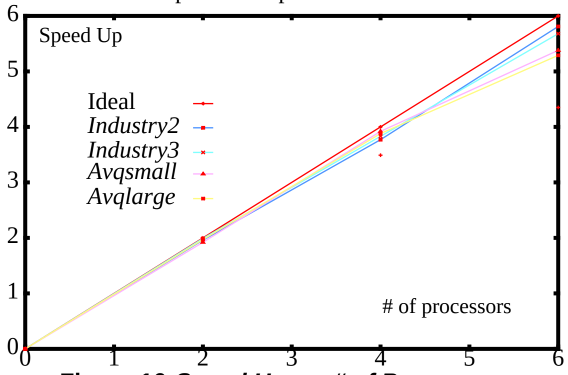

Figure 16 ***Speed Up*** **vs.** ***# of Processors***

6.3 Processor Utilization

Table 8 shows the processor utilization in our parallel environment. They are obtained by dividing the CPU time by the elapsed time. The utilization is on average 98% when 2 processors are used. The average utilization is 87% when 6 processors are used.

Circuit	2 Processors	4 Processors	6 Processors
Industry 2	98%	89%	87%
Industry 3	98%	91%	90%
Avqsmall	98%	94%	86%
Avqlarge	97%	96%	86%
average	98%	93%	87%

Table 8: Processor Utilization

7. Conclusion

We presented a loosely coupled parallel algorithm for the placement of standard cell integrated circuits. Our algorithm is a derivative of simulated annealing. The implementation of our algorithm is targeted toward networks of UNIX workstations. This is the very first reported parallel algorithm for standard cell placement which yields as good or better placement results than its serial version. In addition, it is the first parallel placement algorithm reported which offers nearly linear speedup, in terms of the number of processors (workstations) used, over the serial version. Despite using the rather slow local area network as the only means of interprocessor communication, the processor utilization is quite high, up to 98% for 2 processors and 90% for 6 processors. The new parallel algorithm has yielded the best overall results ever reported for the set of MCNC standard cell benchmark circuits.

8. Reference

[1] P. Banerjee and M. Jones, "A Parallel Simulated Annealing for Standard Cell Placement on a Hypercube Computer." *Proc. Intl. Conf. on Computer-Aided Design* (1986): 34-37.

[2] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells." *Proc. Intl. Conf. on Computer-Aided Design* (1986): 30-33.

[3] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells." *IEEE Trans. on CAD*, Volume 6, No. 5, pp 838-847, Sep 1987.

[4] A. Casotto, A. Sangiovanni-Vincentelli, "Placement of Standard Cells Using Simulated Annealing on the Connection Machine." *Proc. Intl. Conf. on Computer-Aided Design* (1987): 350-353.

[5] K. Doll, F. M. Johannes, and G. Sigl, "Accurate Net Models for Placement Improvement by Network Flow Methods." *Proc. Intl. Conf. on Computer-Aided Design*, 1992, pp. 594-597.

[6] K. Doll, F. M. Johannes, and G. Sigl, "Domino: Deterministic Placement Improvement with Hill-climbing Capabilities." *Proc. VLSI*, 1991, pp. 3b.1.1-3b.1.10.

[7] R. Jayaraman and F. Darema, "Error Tolerance in Parallel Simulated Annealing Techniques." *Proc. Intl. Conf. on Computer Design* (1988): 545-548.

[8] M. Jones and P. Banerjee, "Performance of a Parallel Algorithm for Standard Cell Placement on the Intel Hypercube." *Proc. 24th Design Automation Conf.* (1987): 807-813.

[9] S. Kim, "Improved Algorithms for Cell Placement and their Parallel Implementations," Tech. Rep. #CRHC-93-18, UILU-ENG-93-2231, University of Illinois, Urbana, IL, July 1993.

[10] S. Kim, J. Chandy, B. Ramkumar, S. Parkes and P. Banerjee, "ProperPLACE: A Portable, Parallel Algorithm for Standard Cell Placement," *Proc. 8th Int. Parallel Processing Symp.*, Cancun, Mexico, April 1994.

[11] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization." *IEEE Trans. on CAD*, Volume 10, No. 3, 1991, pp 356-365.

[12] R. Kling and P. Banerjee, "Concurrent ESP: A Placement Algorithm for Execution on Distributed Processors." *Proc. Intl. Conf. on Computer-Aided Design* (1987): 354-357.

[13] K. Kozminski, "Benchmarks for Layout Synthesis." *Proc. 28th Design Automation Conf.*, 1991, pp. 265-270.

[14] S. Kravitz and R. Rutenbar, "Placement by Simulated Annealing on a Multiprocessor." *IEEE Trans. on CAD*, Volume 6, No. 4, pp 534-549, Jul 1987.

[15] J. Lam, J. M. Delosme, and C. Sechen, "Performance of a New Annealing Schedule." *Proc. 25th Design Automation Conf.* (1988): 306-311.

[16] D. Mitra, R. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and Finite-Time Behavior of Simulated Annealing." *Advances in Applied Probability*, Vol. 18. No. 3, pp. 747-771, 1986.

[17] S. Mohan, and P. Mazumder, "Wolverines: Standard Cell Placement on a Network of Workstations." *IEEE Trans. on CAD*, Volume 12, No. 9, pp 1312-26, Sep 1993.

[18] J. S. Rose, D. R. Blythe, W. M. Snelgrove, and Z. G. Vranesic, "Fast, High Quality VLSI Placement on an MIMD Multiprocessor." *Proc. Intl. Conf. on Computer-Aided Design* (1986): 42-45.

[19] J. S. Rose, D. R. Blythe, W. M. Snelgrove, and Z. G. Vranesic, "Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing." *IEEE Trans. on CAD*, Volume 7, No. 3, pp 387-396, Mar 1988.

[20] R. Rutenbar and S. Kravitz, "Layout by Annealing in a Parallel Environment." *Proc. Intl. Conf. on Computer Design* (1986): 434-437.

[21] J. S. Sargent and P. Banerjee, "A Parallel Row-Based Algorithm for Standard Cell Placement with Integrated Error Control." *Proc. 26th Design Automation Conf.* (1989): 590-593.

[22] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A Performance Driven Placement Algorithm for Small Cell ICs." *Proc. Intl. Conf. on Computer-Aided Design*, 1991, pp. 48-51.

[23] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package." *IEEE J. of Solid-State Circuits, vol SC-20, no 2*, pp 510-522, Apr 1985.

[24] C. Sechen and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-based Placement." *Proc. Intl. Conf. on Computer-Aided Design* (1987): 478-481.

[25] G. Sigl, K. Doll, and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?" *Proc. Design Automation Conference*, 1991, pp. 427-432.

[26] W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits." *Proc. Intl. Conf. on Computer-Aided Design* (1993): 170-177.

[27] W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits." submitted to *IEEE Trans. on CAD.*

[28] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells," *Proc. Int. Conf. on Computer-Aided Design*, 1990, pp. 336- 339.

[29] W. Swartz, "Automatic Layout of Analog and Digital Mixed Macro/Standard Cell Integrated Circuits.", *Ph. D. Thesis*, Yale University, 1993.

[30] Chi-Pong Wong and Rolf-Dieter Fiebrich, "Simulated Annealing-Based Circuit Placement Algorithm on the Connection Machine System." *Proc. Intl. Conf. on Computer Design* (1987): 78-82.