International Parallel Processing Symposium (IPPS), April 1994

ProperPLACE: A Portable Parallel Algorithm for Standard Cell Placement

Sungho Kim* John A. Chandy† Steven Parkes†
Balkrishna Ramkumar‡ Prithviraj Banerjee†

* LSI Logic Corporation, Milpitas, CA 95035
† Center for Reliable and High-Performance Computing, University of Illinois, Urbana, IL 61801
‡ Dept. of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242

Abstract

Parallel algorithms developed for CAD problems today suffer from three important drawbacks. First, they are machine specific and tend to perform poorly on architectures other than the one for which they were designed. Second, they cannot use the latest advances in improved versions of the sequential algorithms for solving the problem. Third, the quality of results degrade significantly during parallel execution. In this paper we address these three problems for an important CAD application: standard cell placement. We have developed a new parallel placement algorithm that is portable across a range of MIMD parallel architectures. The algorithm is part of the ProperCAD project which allows the development and implementation of a parallel algorithm such that it can be executed on a wide variety of parallel machines without any change to the source. The parallel placement algorithm is based on an existing implementation of the sequential simulated annealing algorithm, `TimberWolfSC` *6.0 [1].*

1 Introduction

With the rapid improvement in VLSI technology, circuit design is becoming extremely complex and is placing increasing demands on CAD tools. Parallel processing is fast becoming an attractive solution to reduce the inordinate amount of time spent in VLSI circuit design. This has been recognized by several researchers in VLSI CAD as is evident in the recent literature for cell placement, floor planning, circuit extraction, test generation, fault simulation, logic synthesis, etc.

Much of the work reported to date, however, suffers from a major limitation. The parallel algorithms proposed for the CAD applications are designed with a specific underlying architecture in mind. As a result, these programs perform poorly on architectures other than the one for which they were designed. Moreover, incompatibilities in programming environments also make it difficult to port these programs across different parallel architectures.

This limitation has serious consequences. A new parallel algorithm has to be developed for every target MIMD architecture. This is compounded by the length of the software development cycle, which, for parallel programs, is considerably longer than for sequential programs. As a result, parallel programs are considerably costlier to develop than sequential programs.

The major goal of the ProperCAD project [2] is to develop portable parallel algorithms for VLSI CAD applications that will run on a range of parallel machines including shared memory multiprocessors such as the Encore Multimax and Sun 4/670MP, distributed memory multicomputers such as the Intel iPSC/860, and networks of workstations. This was accomplished in two phases. The first phase, ProperCAD I, involved the use of the C-based CHARM language and runtime system [3, 4]. CAD application specific libraries were developed on top of CHARM to support the development of parallel CAD applications. Applications developed using ProperCAD I include circuit extraction [5], test generation [6], and logic synthesis [7, 8]. An earlier version of our placement tool, `ProperPLACE`, was also developed using ProperCAD I [9].

The second phase, ProperCAD II, involves the integration of the CAD libraries and the machine independent runtime support in a truly object–oriented manner [10]. This environment consists of a C++ library which provides a fully object–oriented parallel interface based on the actor model [11] of concurrent object–oriented computing. Applications created using this newer environment include test generation [12] and placement which will be described further in this paper. (See Figure 1)

Parallel execution in the ProperCAD II environment is based on a fundamental object called an actor. An actor object communicates with other actors by sending messages, and all actor actions are in response to these messages. Specific actor methods are invoked to process each type of message, and actors are not allowed to block or ex-

Acknowledgement: This research was supported in part by the Semiconductor Research Corporation under Contract SRC 93-DP-109.

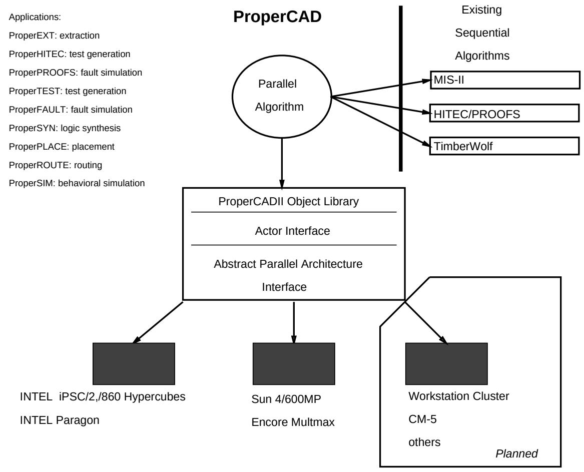

Figure 1: An overview of the ProperCAD project.

plicitly make receive requests from other processors. Also concurrent abstractions known as aggregates [13] are available in ProperCAD II to support a multi-access interface to groups of actors.

In this paper, we present a new parallel algorithm, named `ProperPLACE`, for standard cell placement using simulated annealing. The algorithm has three significant contributions: (1) It is an asynchronous parallel simulated annealing algorithm with an error control mechanism. (2) It is portable across MIMD machines. (3) It is built around an existing sequential placement algorithm using a well-defined sequential algorithm interface, so that it can benefit from future improvements of the sequential algorithm. This last feature ensures that the overheads of parallelization are kept low ($\leq$10%), and the parallel algorithm can automatically benefit from future improvements in the uniprocessor algorithm (such as newer versions of `TimberWolfSC`). One of the major contributions of this work is the fact that we have reused nearly 75% of the sequential source code from `TimberWolfSC` 6.0 in our parallel implementation.

This paper also addresses a significant problem with current approaches to parallel placement. In all the asynchronous schemes proposed to date, maintaining the accuracy of local databases has been the major hurdle. This inaccuracy in the database arises from the fact that accepted moves by each processor are not relayed to the other processors on time. To reduce this inaccuracy, we propose a prioritized message passing technique and a dynamic message size control scheme. We present results on several MCNC and ISCAS benchmark circuits for the Encore Multimax (shared memory), the Intel iPSC/860 hypercube (distributed memory), a Sun 4/670MP (shared memory). Our implementation produces speedups averaging 3.0 on 8 processors while maintaining the quality of placement solution within 20% of `TimberWolfSC` 6.0.

The rest of this paper is organized as follows. In the following section, we review previous work done in parallel simulated annealing for placement. Section 3 describes the `ProperPLACE` algorithm. The keys to the success of our parallel placement algorithm in producing excellent quality and speedup are (1) in using priorities to send messages across processors and (2) in performing message sizing dynamically to control the error in the cost function calculation. These are discussed in detail also in Section 3. Section 4 describes the performance of the `ProperPLACE` algorithm on several parallel machines. We conclude the paper along with a discussion on future work in Section 5.

2 Previous work

2.1 The placement problem

The VLSI cell placement problem involves placing a set of cells on a VLSI layout, given a netlist which provides the connectivity between each cell and a library containing layout information for each type of cell. This layout information includes the width and height of the cell, the location of each pin, the presence of equivalent pins, and the possible presence of feed through paths within the cell. The primary goal of cell placement is to determine the best location of each cell so as to minimize the total area of the layout and the length of the nets connecting the cells together.

Simulated annealing [14] is a powerful but slow algorithm for solving combinatorial optimization problems. It is suitable for problems, such as VLSI cell placement, that are not so well-understood and therefore lack good heuristic algorithms. Simulated annealing has been extremely successful in solving the placement problem, although with long computation times. One of the more popular versions of simulated annealing for placement has been the `TimberWolfSC` set of standard cell tools [1].

2.2 `TimberWolfSC`

In the original version of the `TimberWolfSC` algorithm (version 3.2) for standard cell placement, the cost function consisted of three parts:

- Estimate of wirelength of all nets as the half perimeter of the bounding box.
- Overshoot or undershoot of each row length over the desired row length
- Area overlap between cells in the same row.

Two types of moves are used to generate new configurations. Either a cell is chosen randomly and displaced to a random location on the chip, or two cells are selected randomly and exchanged. A temperature dependent range limiter is used to limit the distance over which a cell can move. Initially, the span of the range limiter is set such that a cell can move anywhere on the chip. Subsequently, the span decreases logarithmically with temperature. The initial temperature is set to a very high fixed temperature, and the final temperature is a very low fixed temperature. Many improvements have been made to the latest version of `TimberWolfSC` (version 6.0) to improve the run-time performance. However, the types of moves and the cost function are the same as the earlier version.

2.3 Parallel algorithms for placement

To reduce the computation time of placement algorithms, numerous researchers have investigated the use of parallelism [15, 16, 17, 18, 19, 20, 21]. There have been two major approaches that researchers have taken to apply parallelism to simulated annealing for placement — move acceleration and parallel moves.

A. Move acceleration In this approach, each move is evaluated faster by dividing up the task of evaluating a move into subtasks and allocating various subtasks to different processors.

B. Parallel moves In this method, each processor generates and evaluates moves independently assuming the other processors are not making any moves. One problem with this approach is that the cost function calculations may be incorrect due to the moves made by the other processors. There are two sub-approaches for handling this inconsistency.

- **B.1** Use parallel evaluation of multiple moves and acceptance of moves that do not interact. The advantage is that the convergence characteristics of the parallel algorithm is identical to the serial algorithm. But, the task of identifying noninteracting moves itself is an difficult problem [15].
- **B.2** Use parallel evaluation and acceptance of multiple interacting moves. In this case, multiple moves are evaluated for acceptance based on inaccurate information, namely, some errors in computations of cost functions are allowed. Although interacting moves may affect the convergence characteristics of the parallel algorithm, this approach has the scope for maximum parallelism and is the approach taken in this research.

Kravitz and Rutenbar [16] have tried two of the approaches (A and B.1.) on a shared memory multiprocessor and obtained a speedup of 2 on 4 processors for the first approach and 3.5 on 4 processors in the second approach. Banerjee, Jones and Sargent [15] implemented a parallel placement algorithm using the parallel move approach on an Intel hypercube multiprocessor and proposed several partitioning strategies for the problem specific to the hypercube topology. Speedups of up to 12 on 16 processors were reported. Casotto et al. [17] worked on speeding up simulated annealing for the placement of macrocells, and have achieved a speed-up of 6 using 8 processors on a shared memory multiprocessor. Rose et al. [18] proposed a parallel algorithm on an experimental distributed memory multiprocessor. In that algorithm, they replaced the high temperature portion of the parallel simulated annealing placer with a placement program based on min-cut algorithm and used a parallel moves (B.2.) strategy for lower temperatures. Speedups of 4 on 5 processors were reported. Jayaraman and Rutenbar [19] have proposed a parallel floor-planning algorithm that runs on an Intel iPSC hypercube multiprocessor. It controls the error by periodic synchronization. Both Casotto and Sangiovanni-Vincentelli [20] and Wong and Fiebrich [21] have presented results on parallel implementation of simulated annealing placement on the Connection Machine.

These and all the other previous work on parallel placement algorithms share a common drawback: each are proposed for specific parallel architectures. Secondly, these parallel algorithms were all developed from scratch by rewriting the sequential algorithm, hence the performance of the algorithms in a single processor was not as efficient as the best sequential algorithms. Finally, the error in the cost function evaluation due to parallel move evaluation was not controlled, hence the quality degraded significantly with more processors. In the next section, we present the `ProperPLACE` algorithm that is machine independent and can run on a variety of MIMD machines without any change to its algorithm.

3 `ProperPLACE`

We have built `ProperPLACE` around an existing uniprocessor implementation of a simulated annealing placement tool, `TimberWolfSC` 6.0. In `ProperPLACE`, we exploit parallelism by using parallel moves and hence allow errors in the cost function (Approach B.2. in the previous section).

3.1 Parallel algorithm

`ProperPLACE` begins with a random placement as an input, and this entire input placement is replicated on each processor. It is divided up topographically by rows, with the rows and the cells contained in the rows assigned to separate processors. For example, the placement in Figure 2 has 4 rows and is replicated on each of 4 processors. Each processor owns one row as shown in Figure 2. Because an entire row, not a sub part, is owned by a processor, there cannot be an error in the calculation of cell overlaps and row lengths (cost function components (2) and (3) in Section 2.2) during the simultaneous evaluation of multiple moves. Note that this approach assumes that the number of rows is greater than or equal to the number of processors. If not, the rows can be split into a number of subrows, for which some overlap penalties may be calculated erroneously.

Each processor then generates moves, in parallel, for the cells which it owns, and tests those moves for acceptance. If a move is accepted, then the accepting processor sends the move to the other processors (by sending an `Update` message) so they can maintain a consistent cell position database. To obtain good speedup, each processor sends

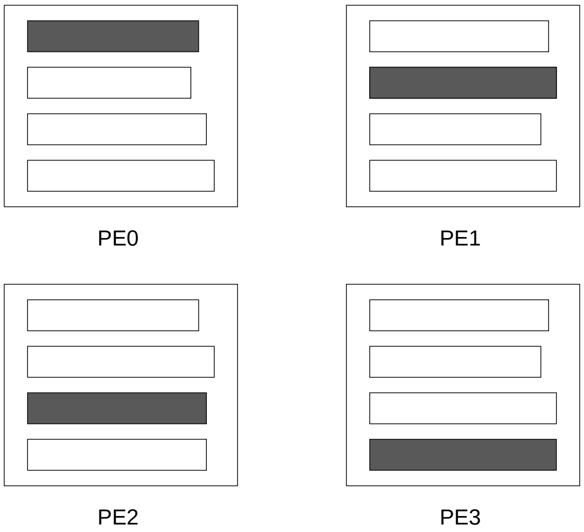

Figure 2: Row ownership in ProperPLACE. Example shows one row per partition.

position updates to other processors after a number of accepted moves, not after each accepted move. Although this reduces the total number of update messages sent among processors, there is a drawback in this approach. Because the local cell position database on each processor becomes increasingly inaccurate as the number of update messages are reduced, the error in the cost function calculation also increases. This error, if too large, may prevent the algorithm from converging to the correct solution. Previous researchers [15, 19, 20, 22] have shown that simulated annealing is tolerant to some error in cost function calculations. In ProperPLACE the frequency of sending position update messages is determined adaptively such that the error in cost function is kept small at all time. This will be discussed in detail in Section 3.3.

After broadcasting a position update message to the other processors, each processor does not remain idle until the update messages sent by other processors are received, but goes ahead with the next sequence of simulated annealing moves. This is why our parallel simulated annealing is asynchronous.

Figure 3 shows how these moves are performed and updated in the ProperCAD environment between two processors. After the ownership is determined topographically as discussed, one Anneal actor and one Placement actor are created per processor. The Anneal actor is responsible for moving cells that the processor owns and the Placement actor is responsible for keeping a coherent state of current placement. If a cell moves to a region owned by another processor, the ownership of the cell goes to the new processor and the annealing process on the original processor is no longer responsible for moving that cell. A user defined threshold, L, is used to limit the number of moves that may be performed in succession without interruption. After L moves, control goes from the Anneal actor to the

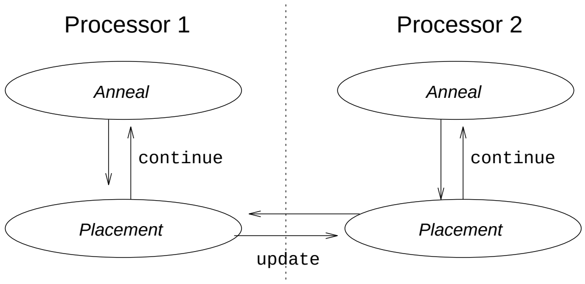

Figure 3: The actors in the ProperPLACE algorithm

Placement actor in the same processor (See the arrow originating from Anneal in Figure 3). The Placement actor sends position update messages to the other processor if it is time to do so. Before returning control back to the Anneal actor, it processes all the Update messages sent from the other processors to make the local database up-to-date. When the control comes back to the Anneal actor, the next L moves are proposed and evaluated. We reiterate the fact that these processes do not remain idle (i.e. waiting for some messages to arrive) at any time, but go ahead with the next task that they were assigned. This complete elimination of the idle times will give greater speedup.

In Figure 4, we provide an outline of the ProperPLACE placement algorithm. Each processor first proposes a move and that move is evaluated in the accept() procedure. There are two sub-classes of moves for both displacement and exchange, or four move types in total:

M1. Intraprocessor Cell Displacement. Cell A moves to a new location owned by the same processor.

M2. Intraprocessor Cell Exchange. Two cells A and B owned by the same processor exchange their locations.

M3. Interprocessor Cell Displacement. Cell A moves to a new location owned by a different processor.

M4. Interprocessor Cell Exchange. Two cells A and B owned by different processors are exchanged.

An example of each type of move is shown in Figure 5. In the figure, assume that the upper row is owned by PE0 and the lower one by PE1. Notice that the three moves (M1, M2, M3) can be done alone by PE0, the owner of cell A. For move type M4, however, PE0 needs permission from PE1 which owns cell B. This is because it is possible that cell B may have already been moved to another location or is frozen due to some pending move. Consider an example interprocessor exchange move in which the exchange of cell A and cell B is proposed by PE0. Because the information about cell B may be out of date in the database of PE0, PE0 locks (or freezes) cell A and cell B locally and sends a request message to PE2, the owner of cell B. Upon receiving the request message, PE2 examines the state of cell B and determines whether to allow the

```
for all processors in parallel {
   repeat {
      repeat {
         E = 0;
         select cell A in this PE's region
         move = M1, M2, M3 or M4
         if (move == M2 || move == M4)
            select cell B
         evaluate cost
         if (accept(A, B, move) == yes) {
            if( move = M4 ) {
               freeze(A, B)
               send AskPermission msg to B's PE
            }
            else {
               accept the move
               update local database
               send Update msg to other PEs
               E = E + error_estimate(move)
            }
         }
      } until equilibrium is reached
      lower T
      if (E < T/21)
         increase update message size
      else
         decrease update message size
   } until stop criterion is met
}
```

Figure 4: An outline of the `ProperPLACE` algorithm

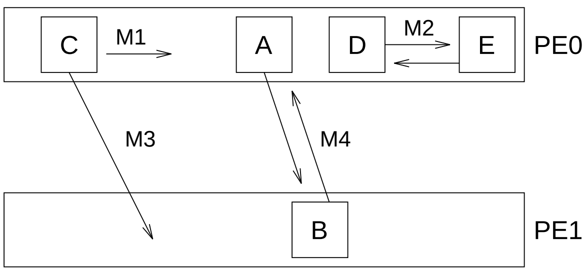

Figure 5: Moves in `ProperPLACE`

exchange. Upon making a decision, PE2 sends its decision back to PE0 by sending the `ReturnAnswer` message. When `ReturnAnswer` is picked up by PE0 from its message queue, PE0 unlocks cells A and B, and it makes the move if the returned answer is yes. It is important to note that PE0 does not wait until the `ReturnAnswer` message is received. Instead, it continues annealing by making other moves with unfrozen cells that it owns. This is another instance of why we call our method asynchronous. Because the interprocessor exchange move (M4) takes the most time due to extra message passing, we introduce a prioritized message scheme to reduce the time taken by M4 in the next section.

3.2 Prioritized message scheme

An interesting feature of the ProperCAD environment system is that the programmer can influence the order in which messages in the work pool are picked for processing by assigning priorities to them. Prioritized execution is instrumental in delivering the performance presented in Section 4 for the `ProperPLACE` placement algorithm. In this section, we describe the use of priorities both to reduce the runtime and to improve the quality of solution. The reduction in runtime is achieved by reducing the time taken by the interprocessor exchange move (M4), which is considerably longer than that of all the other types of moves, and the improved solution quality is achieved by reducing the database inconsistencies.

In our parallel simulated annealing algorithm, the messages passed among processes are one of the following:

- `Update`
- `Continue`
- `AskPermission`
- `ReturnAnswer`

We give the highest priority to `AskPermission` and `ReturnAnswer`. This reduces (1) the probability of cell B, involved in the interprocessor exchange move, being moved to another location while `AskPermission` message is received by the owner of cell B and (2) the time taken to thaw frozen cells. Messages of type `Update` are given the next highest priority. Finally, the message of type `Continue` is given the lowest priority. By giving a higher priority to `Update` than `Continue`, all `Update` messages received by a processor are picked up from the message queue before the `Continue` message. Therefore, the most up-to-date cell location information is always used at the beginning of each block of L moves.

3.3 Dynamic message sizing and error control

To reduce the communication overhead, a processor broadcasts an `Update` message to other processors periodically after accumulating a number of accepted moves, not after each move. Now, the problem is to determine the frequency of this `Update` message, or equivalently, the number of moves between updates (M). We define the $message\_size$ as the number of moves in an `Update` message. If the $message\_size$ is too large, accepted moves by a processor do not appear on time in the other processor's database. Consequently, the error E in the cost function calculation increases and results in degradation of the solution quality. Such a cumulative error in cost function has also been recognized as a severe problem by several researchers [23, 24, 15]. On the other hand, if the message size is too small, the number of messages sent is increased and results in large communication overheads.

A naive approach to determining the message size is to use a static method in which the message size is predetermined and fixed. The problem with such an approach is that there is no good way to determine this message size a priori. In a dynamic approach, the message size is determined during the annealing process by monitoring the size of the error in the cost function in the system. If the error size during the annealing process becomes too large, the message size automatically becomes smaller to reduce the error at the cost of the increased number of messages. On the other hand, if the size of the error becomes very small, the message size increases to reduce the number of messages. As long as the size of the error is bounded, this dynamic approach will produce quality solutions equivalent to that of a sequential simulated annealing algorithm.

In the `ProperPLACE` algorithm, the error in the cost function is defined to be the difference between the real change in cost from the initial to final configurations, and the estimated change in cost equal to the sum of locally perceived changes in cost at each processor. If C_i is the exact initial cost, ΔC_j is the change in cost computed locally at n processors, and C_f is the exact cost of the new configuration after a series of moves, then

$$C_i + \sum_{j=1}^{n} \Delta C_j = C_f + Error$$

Since the processors in the `ProperPLACE` algorithm do not synchronize to exchange cell position information, C_f is available only at the end of the annealing process at which time each processor has the identical copy of the entire circuit. Consequently, $Error$ cannot be obtained during the placement process. Therefore, in an asynchronous approach, one can only guess (or estimate) what the $Error$ will be during the placement process. In the `ProperPLACE` algorithm, instead of estimating $Error$, we estimate, E, the error that each processor contributes by moving its own cells. The advantage of obtaining the error this way is that it can be calculated independently by each processor without any synchronization.

In Figure 6, PE0 makes a sequence of M moves $(m_1, m_2, ..., m_10)$. While these M (=10) moves are proposed, evaluated and accepted, no `Update` message is sent to the other processors. This may cause error in the cost function calculations in the other processors if the accepted moves in M have some nets connected to some cells in the other processors. In addition, earlier moves have a higher

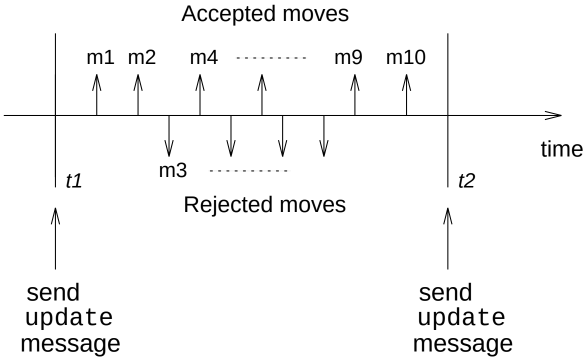

Figure 6: Moves between `Update` messages ($M = 10$)

chance of causing such an error than later ones. For example, m_1, if accepted, is more likely to cause an error than m_{10} because a greater number of remote cells will be moved during the time between m_1 and t_2 than between m_{10} and t_2.

In the `ProperPLACE` algorithm, each processor calculates the error that it is contributing to the other processors by

$$E_{estimate} = \frac{\sum_{i \in A} e(i)p(i)}{N}$$

In the above equation, A is a set of N accepted moves within a time interval $[t_1, t_2]$; $e(i)$ is the upper bound on the amount of error for move i, which is the amount of the cell displacement; and $p(i)$ is the probability that some nets connected to cell i will be moved by other processors during the time between accepting move i and sending the next `Update` message:

$$p(i) = 1 - \prod_{j\ =\ all\ remote\ gates\ sharing\ a\ net\ with\ i} (1 - P(j))$$

where $P(j)$ is the probability that gate j is moved during the time between accepting move i and sending the next `Update` message:

$$P(j) = acceptance\_rate(owner\ of\ j) \cdot \frac{M - move\_attempts}{cells_j}$$

where $move\_attempts$ is the number of moves attempted between sending the last update message and accepting move i, and $cells_j$ is the number of cells owned by the owner of j.

Figure 7 shows an example of the error estimation on a single displacement move. In the figure are three standard cell rows, which are owned by processor 1, processor 2 and processor 3, respectively. Suppose that gate i moves to a new location by the distance of 10 as shown by an arrow in the figure. Since the cells A, B, C, and E, which are connected to cell i through a net, may be moved before PE2 sends the next `Update` message, there may be an error in calculating the cost function by PE1 and PE3. Let us assume that each processor owns 100 cells and that

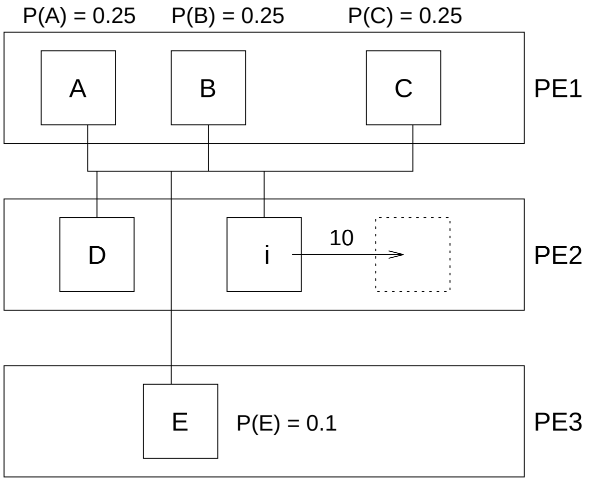

Figure 7: An example of error estimation

the current acceptance rate of simulated annealing is 0.5 for PE1 and 0.2 for PE3. Also, assume that PE2 has attempted 50 moves since broadcasting the `Update` message. Then, $P(A) = P(B) = P(C) = 0.5 \cdot \frac{50}{100} = 0.25$. $P(E) = 0.2 \cdot \frac{50}{100} = 0.1$. Therefore, $p(i) = 1$ - (1-0.25)(1-0.25)(1-0.25)(1-0.1) = 0.62 and $E_{estimate}(i) = 0.62 \cdot e(i)$ where $e(i)$ is the amount of displacement of cell i, an upper bound on the error. Because $e(i) = 10$ in this example, $E_{estimate}(i) = 6.2$.

This error accumulated over M moves is used to control the message size of each processor. In `ProperPLACE`, we put a bound on the error in the cost function as originally reported in [15]. In the simulated annealing of the `ProperPLACE` algorithm, the probability of accepting a move is

$$P = e^{-\Delta C/T} \cdot Prob(\Delta C > 0) + Prob(\Delta C < 0)$$

In the presence of error, the composite acceptance rate changes slightly; however, the probability of generating good or bad moves is invariant with respect to error:

$$P_E = e^{-(\Delta C + -E)/T} \cdot Prob(\Delta C > 0) + Prob(\Delta C < 0)$$

To bound the acceptance rate with error P_E to within 5% of normal, i.e.,

$$\frac{P - P_E}{P} <= 0.05$$

we find a bound on magnitude of error:

$$E <= -T/ln(1 - 0.05) \ approx \ T/21$$

We decrease the message size by $\frac{current_{size}}{2} \cdot (1 - e^{-\frac{E}{k}})$, where $k = 0.0687 \cdot T$, whenever the computed error E is higher than $T/21$. If the error is very low, then the message size is increased similarly as seen in Figure 3(a).

Table 1: Circuit Information
ISCAS and Physical Design Workshop 91 benchmarks

Circuit	Num. cells	Num. nets	Initial wirelength
fract	125	147	111779
s298	133	138	126984
primary1	760	1172	1202241
struct	1888	1920	2668124
primary2	3014	3817	7292946

3.4 Load balancing by interprocessor move suppression

Because the time to evaluate each move differs, some processors perform annealing much faster than others. Also, the number of cells owned by each processor may vary considerably because cells are allowed to move to other processors. To maintain approximately the same number of cells on each processor, we need to rebalance the cells among processors.

In our algorithm, we achieve this balance in the number of cells by varying the type of moves proposed and accepted. For example, when the number of cells falls below two-thirds of the original assignment, we cut down on the number of interprocessor moves. Because this reduces the probability that a cell moves out of this processor, the number of cells moving into this processor's region becomes greater than the number moving away. Therefore, the balance is maintained by this simple technique. Similarly, if a processor owns much more cells than the average, more interprocessor moves are proposed to increase the probability of cells moving out. This change in move types does not affect the placement solution.

We observed that even when the number of cells remains the same, some processors anneal at a much faster rate because some moves are evaluated quicker than others. To reduce such time imbalance, the slow processors reduce the number of interprocessor exchange moves which take the largest evaluation times.

4 Results

The `ProperPLACE` parallel algorithm described in the previous sections has been implemented as part of the ProperCAD project. In this section, we will describe the results obtained by applying `ProperPLACE` on various circuits in the ISCAS and MCNC benchmark suite (Table 1). It should be noted that all previous work in parallel placement have used relatively small circuits for their evaluations. We are using reasonably complex circuits that are accepted in the placement community.

Table 2: `TimberWolf` and `ProperPLACE` Comparison

Circuit	wirelength		run time (s)	
	`Timber-WolfSC`	`Proper-PLACE`	`Timber-WolfSC`	`Proper-PLACE`
fract	20771	21184	398.7	433.1
primary1	252777	249982	954.9	1048
struct	182930	173730	2660	2701
primary2	1417035	1410450	3205	3201

Table 3: `ProperPLACE` on an Encore Multimax

Speedups				
Circuit	1 PE (s)	2 PE	4 PE	8 PE
fract	1491	1.49	1.72	2.76
primary1	4571	1.53	2.09	2.72
struct	12477	1.32	1.86	2.38
primary2	15427	1.48	1.87	2.86
Wirelengths				
Circuit	1 PE	2 PE	4 PE	8 PE
fract	1.00	1.06	1.11	1.20
primary1	1.00	1.01	1.11	1.26
struct	1.00	1.04	1.10	1.31
primary2	1.00	1.02	1.25	1.35

4.1 Results with prioritized messages and dynamic message sizing

In Table 2, we compare the quality of circuits obtained by `TimberWolfSC` 6.0 with that of the `ProperPLACE` algorithm in an uniprocessor environment. One can observe that the quality of the circuits produced by `ProperPLACE` are almost the same (sometimes better) as those produced by `TimberWolfSC` 6.0. The last columns compare the run times for `TimberWolfSC` 6.0 and for `ProperPLACE` for a Sun 4/670MP server. It should be noted that all previous work on parallel placement have had to reimplement the sequential algorithm in a simplified way, and the performances of those algorithms on a single processor were much inferior to the best implementations of the sequential algorithms.

Little changes were made to the underlying uniprocessor placement algorithm. In order to make the work comparable, the number of moves attempted in the inner iteration of simulated annealing by each processor has been reduced:

$$M_{par} = \frac{M_{uni}}{number\ of\ processors}$$

where M_{uni} is the number of moves attempted in `TimberWolfSC` 6.0.

Table 3 presents the results obtained on an Encore Multimax multiprocessor with 8 processors. For different number of processors, the quality of the results in terms of normalized wirelength and the speedups are shown for the benchmark circuits. Likewise, Tables 4 and 5 present results obtained on an Intel iPSC/860 hypercube machine and a Sun 4/670MP shared memory server respectively. Work is in progress to enable the ProperCAD II environment to support networks of workstations as well as other parallel machines such as the Intel Paragon.

From the tables it is clear that the `ProperPLACE` algorithm produces excellent speedups, and maintains quality comparable to that of `TimberWolfSC` 6.0. One can observe that the maximum degradation of quality with 8 processors is less than 25%.

Table 4: `ProperPLACE` on an Intel iPSC/860

Speedups				
Circuit	1 PE (s)	2 PE	4 PE	8 PE
fract	527.3	1.50	1.91	2.94
primary1	1442	1.45	2.13	3.04
struct	3795	1.49	1.96	2.92
primary2	4347	1.50	1.98	3.21
Wirelengths				
Circuit	1 PE	2 PE	4 PE	8 PE
fract	1.00	1.04	1.06	1.07
primary1	1.00	1.06	1.08	1.19
struct	1.00	1.04	1.02	1.30
primary2	1.00	1.03	1.21	1.22

Table 5: `ProperPLACE` on a Sun 4/670MP

Speedups			
Circuit	1 PE (s)	2 PE	4 PE
fract	398.7	1.42	1.63
primary1	1048	1.37	1.73
struct	2701	1.37	2.32
primary2	3201	1.48	1.65
Wirelengths			
Circuit	1 PE	2 PE	4 PE
fract	1.00	1.07	1.05
primary1	1.00	1.03	1.11
struct	1.00	1.00	1.06
primary2	1.00	1.04	1.18

4.2 Effect of message sizing on runtime and quality

In Section 3.3, we have discussed how the estimated cost function error is used to control the message size. This dynamic message sizing scheme has been implemented and, in this section, we evaluate the effectiveness of the dynamic message sizing scheme by performing the following experiment. Figures 8a and 8b show the quality of the solution and runtimes for both static and dynamic approaches for one circuit (s298) on an 8 processor Encore Multimax.

Table 6: Comparison of quality and runtime with and without synchronizing barriers (4 procs)

Circuit	synchronous		asynchronous	
	wire-length	speedup	wire-length	speedup
fract	21336	1.13	23209	1.63
primary1	276620	1.38	289433	1.73
struct	203568	1.06	184613	2.32
primary2	1676353	1.05	1678079	1.65

For the static approach, the program was run three times with the message size set at 20, 40 and 80 (See the first three bars in each graph). One can observe that, as the message size increases in the static method, the quality of solution degrades, while the time taken for each run becomes shorter. The fourth bar in Figures 8a and 8b show the wirelength and the time taken for the same circuit (s298) when a dynamic message sizing scheme was used. Figure 8c shows the performance, $\frac{1}{wirelength \times time}$, of each run. Clearly, the dynamic approach produced the best performance, trading off a less than ideal runtime for negligible loss in wirelength. It is possible to get either better runtimes or better wirelengths but not both.

4.3 Effect of synchronizing barriers on quality and speedup

In this paper, we have presented an asynchronous parallel algorithm, with no synchronization barriers. In this subsection, we demonstrate the effect of synchronization barriers on the speedup and the quality of the placement produced. In this experiment, we put synchronization barriers 12 times inside each inner loop of simulated annealing. As discussed earlier, such synchronization barriers cause processors to wait for the slowest processor to perform its task, but provide perhaps better control of the error. Table 6 compares the results (wirelength and speedups) obtained with the synchronization barriers with the results obtained with asynchronous algorithm of `ProperPLACE`, on 4 processors. The results are presented for the Sun 4/670MP. One can observe that the scheme with synchronizing barriers produce marginally better quality compared to the `ProperPLACE`. But the run times are much worse for the scheme with the synchronization barriers.

5 Conclusions and future work

We have demonstrated that it is possible to design and implement a portable parallel placement tool that is cleanly and efficiently interfaced with a simulated annealing-based uniprocessor algorithm (`TimberWolfSC` 6.0) using the ProperCAD II environment. We have shown that the parallel placement tool runs unchanged on a range of MIMD machines including shared memory machines and message passing machines. A novel parallel algorithm based on the use of priorities and a dynamic message sizing was used to deliver good consistent speedups with little degradation in the wire length. We believe that this is the first parallel placement algorithm to effectively exploit shared memory machines and message passing machines in this manner.

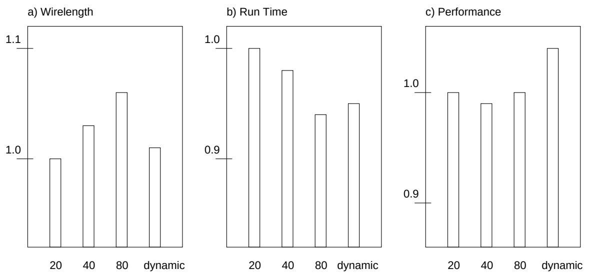

Figure 8: Comparison of quality, runtime and performance using static and dynamic message sizes

Another important feature of this algorithm is that future improvements to the uniprocessor algorithm will result in improvements to the parallel performance of the `ProperPLACE` algorithm with little effort. It is only necessary to keep the interface between the uniprocessor and parallel placement algorithms unchanged. We expect to report improved performance from the parallel placement as and when new optimizations for the uniprocessor placement program, namely future `TimberWolfSC` versions, are developed.

Acknowledgements

Our sincere thanks to Dr. Carl Sechen for providing us with his uniprocessor code for placement (`TimberWolfSC` 6.0), and we would also like to acknowledge the support of Argonne National Laboratories for providing us access to their Intel iPSC/860 hypercube.

References

[1] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal for Solid State Circuits*, vol. SC-20, pp. 510–522, Apr. 1985.

[2] B. Ramkumar and P. Banerjee, "ProperCAD: A portable object-oriented parallel environment for VLSI CAD," in *Proceedings of the International Conference on Computer Design*, pp. 544–548, Oct. 1992.

[3] W. Fenton, B. Ramkumar, V. A. Saletore, A. B. Sinha, and L. V. Kalé, "Supporting machine independent programming on diverse parallel architec-

tures," in *Proceedings of the International Conference on Parallel Processing*, Aug. 1991.

[4] L. V. Kalé, "The Chare Kernel parallel programming system," in *Proceedings of the International Conference on Parallel Processing*, pp. 17–25, Aug. 1990.

[5] B. Ramkumar and P. Banerjee, "ProperEXT: A portable parallel algorithm for VLSI circuit extraction," in *Proceedings of the International Parallel Processing Symposium*, pp. 434–438, 1993.

[6] B. Ramkumar and P. Banerjee, "Portable parallel test generation for sequential circuits," in *Digest of Papers, International Conference on Computer-Aided Design*, pp. 220–223, Nov. 1992.

[7] K. De, B. Ramkumar, and P. Banerjee, "ProperSYN: A portable parallel algorithm for logic synthesis," in *Digest of Papers, International Conference on Computer-Aided Design*, pp. 412–416, Nov. 1992.

[8] K. De and P. Banerjee, "Portable parallel algorithms for factoring and simplication in logic synthesis." submitted to the 1994 Design Automation Conference. University of Illinois, CRHC Tech. Rep. CRHC-93-20/UILU-ENG-93-2235.

[9] S. Kim, "Improved algorithms for cell placement and their parallel implementations." Ph.D. Dissertation, University of Illinois at Urbana-Champaign, July 1993.

[10] S. Parkes, J. A. Chandy, and P. Banerjee, "ProperCAD II: A run-time library for portable, parallel, object-oriented programming with applications to VLSI CAD," Tech. Rep. CRHC-93-22/UILU-ENG-93-2250, Center for Reliable and High-performance Computing, University of Illinois, Urbana, Illinois, Dec. 1993.

[11] G. A. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems.* The MIT Press, 1986.

[12] S. Parkes, P. Banerjee, and J. H. Patel, "ProperHITEC: A portable, parallel, object–oriented approach to sequential test generation," in *Proceedings of the Design Automation Conference*, June 1994. To appear.

[13] A. A. Chien, *Concurrent Aggregates: Supporting Modularity in Massively Parallel Programs.* The MIT Press, 1993.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecci, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[15] P. Banerjee, M. H. Jones, and J. S. Sargent, "Parallel simulated annealing algorithms for standard cell placement on hypercube multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 91–106, Jan. 1990.

[16] S. A. Kravitz and R. A. Rutenbar, "Multiprocessor-based placement by simulated annealing," in *Proceedings of the Design Automation Conference*, pp. 567–573, June 1986.

[17] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of macro-cells," *IEEE Transactions on Computer Aided Design*, vol. CAD-6, pp. 838–847, Sept. 1987.

[18] J. S. Rose, W. M. Snelgrove, and Z. G. Vranesic, "Parallel cell placement algorithms with quality equivalent to simulated annealing," *IEEE Transactions on Computer Aided Design*, vol. 7, pp. 387–396, Mar. 1988.

[19] R. Jayaraman and R. A. Rutenbar, "Floorplanning by annealing on a hypercube multiprocessor," in *Digest of Papers, International Conference on Computer-Aided Design*, pp. 346–349, Nov. 1987.

[20] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "Placement of standard cells using simulated annealing on the Connection Machine," in *Digest of Papers, International Conference on Computer-Aided Design*, pp. 350–353, Nov. 1987.

[21] C. P. Wong and R.-D. Fiebrich, "Simulated annealing-based circuit placement algorithm on the Connection Machine," in *Proceedings of the International Conference on Computer Design*, pp. 78–82, Oct. 1987.

[22] L. K. Grover, "A new simulated annealing algorithm for standard cell placement," in *Digest of Papers, International Conference on Computer-Aided Design*, pp. 378–380, Nov. 1986.

[23] M. D. Durand, "Trading accuracy for speed in parallel simulated annealing algorithms." Ph.D. Dissertation, Columbia University, 1993.

[24] F. Darema-Rogers, S. Kirkpatrick, and V. Norton, "Simulated annealing on shared memory parallel systems," Tech. Rep. RC 12195(54812), IBM, Oct. 1986.