Proceedings of the 9th International Conference on VLSI Design, Bangalore, India, January 1996.

Parallel Simulated Annealing Strategies for VLSI Cell Placement

John A. Chandy **Prithviraj Banerjee**
Center for Reliable and High-Performance Computing
University of Illinois
Urbana, Illinois 61801

Abstract

Simulated annealing based standard cell placement for VLSI designs has long been acknowledged as a compute-intensive process, and as a result several research efforts have been undertaken to parallelize this algorithm. Most previous parallel approaches to cell placement annealing have used a parallel moves approach. In this paper we investigate two new approaches that have been proposed for generalized parallel simulated annealing but have not been applied to the cell placement problem. Results are presented on the effectiveness of implementations of these algorithms when applied to the cell placement problem. We find that the first, multiple Markov chains, appears to be promising since it uses parallelism to obtain near linear speedups with no loss in quality. The second, speculative computation, while maintaining quality is not suitable since no speedups are achieved due to the specific nature of the cell placement problem. The two algorithms are compared with the parallel moves approach to parallel cell placement.

1 Introduction

With the rapid advances in VLSI process technology, circuit design is becoming increasingly complex and in turn is placing ever higher demands on CAD tools. The use of parallel processing is fast becoming an attractive solution to reduce the inordinate amount of time spent in VLSI circuit design. This fact 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.

Standard cell placement is particularly expensive because of the inherent compute intensive nature of simulated annealing, the most popular approach to cell placement. There have been several attempts to parallelize this algorithm, usually with quality results that do not compare to the best available sequential algorithm, or with speedups that are less than desirable. In this paper, we examine two parallel algorithms that have been proposed for generalized simulated annealing. Though these algorithms are claimed to be general, they have yet be applied to the problem of cell placement. Thus, this paper tests their generality and attempts to determine the efficacy of these approaches for cell placement. The two algorithms, multiple Markov chains [1] and speculative computation [2], are designed to retain quality while at the same time providing acceptable speedups.

This research was supported in part by the Semiconductor Research Corporation under contract SRC 94-DP-109 and the Advanced Research Projects Agency under Army Research Office contract DAA-H04-94-G-0273.

These algorithms have been implemented as part of the ProperCAD project [3], the major goal of which is to develop portable parallel algorithms for VLSI CAD applications that will run on a range of architectures. These include shared memory multiprocessors such as the Encore Multimax and Sun 4/690MP, distributed memory multicomputers such as the Intel iPSC/860, Intel Paragon, and Thinking Machines CM-5, and networks of workstations. ProperCAD II provides C++ library-based machine-independent runtime support in an object–oriented manner [4].

Through the use of a fundamental object called an actor [5], the library provides mechanisms necessary for achieving concurrency. An actor object consists of a thread of control that 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 explicitly make receive requests from other processors. The runtime system on each processor picks the next available actor thread with some priority and that thread is then allowed to run to completion without interruption.

The rest of this paper is organized as follows. In the following two sections, we introduce the placement problem and review previous work in parallel simulated annealing for placement, particularly the parallel moves approach. Section 4 describes the multiple Markov chains algorithm and variations upon that approach, and section 5 presents the speculative computation algorithm.

2 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.

One of the more popular sequential applications for placement has been the `TimberWolfSC` cell placement tool [6]. `TimberWolfSC`'s core algorithm, simulated annealing, is a suitable approach to problems like VLSI cell placement since they lack good heuristic algorithms. Briefly, simulated annealing is an iterative improvement strategy that starts with a system in a disordered state, and through perturbations of the state, brings the system gradually to a low energy, and thus optimal, state. A significant characteristic of simulated annealing is that, unlike greedy algorithms, perturbations that increase the energy of the system are sometimes accepted with a probability related to the temperature of the system [7]. In the context of cell placement, and `TimberWolfSC` in particular, perturbations are simply moves of the cells to different locations on the layout, and the energy is an approximated layout cost function, consisting of three parts:

- Half perimeter wirelength estimate of all nets
- Penalty for area overlap between cells in the same row
- Penalty for difference between actual and desired row length

Moves are generated by choosing a random cell and then displacing it to a random location on the layout. If a cell is already present at the new location, the two cells are exchanged. A temperature dependent range limiter is used to limit the distance over which a cell can move. `TimberWolfSC` also segments the simulated annealing procedure into 160 iterations. As the algorithm progresses, the temperature is gradually decreased by forcing the acceptance rate to follow a theoretically derived schedule that attempts to keep the acceptance rate close to 44% during the middle region of annealing [8]. `TimberWolfSC 6.0` also uses row bins to aid in the computation of overlap penalties, and early rejection methods are used to speed up the decision process [9].

3 Parallel algorithms for placement

3.1 Previous work

There have been several algorithms proposed for the parallelization of simulated annealing [10]. In this section we review some of the approaches that researchers have investigated in the parallelization of simulated annealing based placement algorithms [11, 12, 13, 14]. Researchers have taken two major approaches to apply parallelism to simulated annealing for placement — move acceleration and parallel moves.

A. Move acceleration. In this approach, each individual move is evaluated faster by breaking up the move into subtasks such as selecting a feasible move, evaluating the cost changes, deciding to accept or reject, and perhaps updating a global database. Concurrency is obtained by delegating these individual subtasks to different processors. Such approaches to parallelization are restricted to shared memory architectures and have limited scope for large scale parallelism.

B. Parallel moves In this method, each processor generates and evaluates moves independently, however, due to the moves made by the other processors the cost function calculations may be incorrect. 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 sequential algorithm are preserved. However, the task of identifying noninteracting moves is a difficult problem [12].

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 been considered to have the maximum scope for parallelism.

Kravitz and Rutenbar [11] 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 [12] 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. [13] worked on speeding up simulated annealing for the placement of macrocells, and achieved speedups of 6 using 8 processors on a shared memory multiprocessor. Rose et al. [14] 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 mincut algorithm and used a parallel moves (B.2.) strategy for lower temperatures. Speedups of 4 on 5 processors were reported. Sun and Sechen have shown results achieving near linear speedup on a network of workstations using a parallel moves approach [15].

3.2 A parallel moves implementation

In this section, we describe an implementation of parallel moves, called `ProperPLACE`, based on work by Kim, et al [16]. The initial random input placement is replicated on each available physical processor, and the ownership of cells is assigned topographically by rows. Each processor is responsible for a set of rows, and thus is only allowed to attempt moves on cells in that row. If a cell is moved to a region owned by another processor, the ownership of the cell is transferred to the new processor and the original processor is no longer responsible for moving that cell. Because an entire row, not a sub part, is owned by an processor, there is no error in the calculation of cell overlaps and row lengths during the simultaneous evaluation of multiple moves. If the number of rows is greater than or equal to the number of processors, the rows must be split into a number of subrows, in which case some overlap penalties may be calculated erroneously.

After partitioning, each processor proceeds with the core annealing algorithm of `TimberWolfSC` described in Section 2. A valid cell is selected for perturbation, and then a displacement or exchange is performed on that cell. If a move is accepted, then the accepting processor must send the move to the other processors so a consistent cell position database can be maintained. In order to amortize the startup cost of sending a message, position update messages are held until a number of moves have been accepted. As the frequency of update messages is reduced, the local cell position database on each processor becomes increasingly inaccurate, thereby causing the cost function calculation error to increase as well. This error, if too large, may prevent the algorithm from converging to an optimal solution. Other researchers [12] have shown that simulated annealing is tolerant to some error in cost function calculations. The frequency of sending position update messages is determined adaptively such that the error in the cost function is kept small at all times.

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

Circuit	Number of cells	Number of nets	Circuit state size (bytes)
fract	125	163	9636
s298	133	138	8884
s420	212	233	13844
primary1	760	1172	65484
struct	1888	1920	119636
industry1	2271	2583	219220
primary2	3014	3817	1901814

Results are presented for a set of benchmark circuits in Tables 2 and 3 (Circuit information is detailed in Table 1). W indicates the normalized wirelength cost of the resultant placement, and S is the speedup compared to a single processor run. The algorithm provides moderate speedup with some loss of quality. Parallel placement algorithms based on parallel moves are hindered by the cumulative error effects on the global cell database causing a significant degradation in quality as the number of processors increases.

Also, because of the necessary global updates, the speedups will likely not scale linearly with the number of processors. Since every processor must perform these updates, this becomes essentially a serial bottleneck. In fact, because updates are done only for accepted moves, speedups are intimately related to the acceptance rate. Consider if t_{me} is the time to propose and evaluate a move, and t_d and t_u are the times to decide, and update a move respectively, the total run time of a serial execution is $t_{ser} = I(t_{me}+t_d+t_u)$ where I is the total number of moves attempted. In a parallel moves implementation the run time is as follows:

$$t_{par} = \frac{I}{N}(t_{me} + t_d + t_u + \alpha(N-1)t_{mu})$$

where N is the number of processors and α is the acceptance rate. t_{mu} is the time required to perform another processor's accepted move. Note because of the way `TimberWolfSC` does incremental wirelength cost estimates, t_{mu} is not equivalent to t_u. The speedup can be characterized as follows.

$$S = \frac{t_{ser}}{t_{par}} = \frac{I(t_{me}+t_d+t_u)}{\frac{I}{N}(t_{me}+t_d+t_u+\alpha(N-1)t_{mu})}$$

Measurements have shown that t_{mu} is close to $t_{me}+t_u$, and $t_{me}+t_u \gg t_d$[1]. Using these assumptions, S simplifies to $\frac{N}{1+\alpha(N-1)}$, essentially limiting the speedup to $\frac{1}{\alpha}$ regardless of N.

In the following two sections, we investigate two new algorithms that have been proposed for generalized simulated annealing. The first, multiple Markov chains, addresses the scalability issues and the second, speculative computation, attempts to achieve parallelism without loss of quality. In previous literature, these two algorithms have not been applied to the cell placement problem, so we are examining their effectiveness when applied to cell placement. Both are promising because they do not allow errors in the database, and thereby afford better quality solutions.

[1] On a SUN4/690MP, $t_{mu} \approx 116\ \mu s$ for cell exchanges and 48 μs for cell displacement. $t_{me}+t_u$ are 118 μs and 58 μs respectively. The decision time, $t_d \approx 5\ \mu s$.

Table 2: Parallel Moves on an Sun 4/690MP

	1 PE		2 PE		4 PE	
Circuit	W	S	W	S	W	S
fract	1.00	1.00	0.98	1.62	1.08	2.17
s298	1.00	1.00	0.96	1.56	1.03	1.90
s420	1.00	1.00	1.00	1.45	1.07	1.92
primary1	1.00	1.00	0.98	1.50	1.07	1.88
struct	1.00	1.00	0.93	1.28	1.11	1.70
industry1	1.00	1.00	1.01	1.54	1.00	1.66
primary2	1.00	1.00	1.06	1.48	1.14	1.65

Table 3: Parallel Moves on an Intel iPSC/860

	1 PE		4 PE		8 PE	
Circuit	W	S	W	S	W	S
fract	1.00	1.00	0.96	2.65	1.04	3.68
s298	1.00	1.00	1.04	2.17	1.02	2.49
s420	1.00	1.00	1.06	2.31	1.08	2.78
primary1	1.00	1.00	1.00	2.41	1.09	3.14
struct	1.00	1.00	1.09	2.59	1.05	3.71
industry1	1.00	1.00	1.02	2.62	1.03	3.74
primary2	1.00	1.00	1.07	2.45	1.12	3.13

4 Multiple Markov chains

In this section we introduce multiple Markov chains, first presented as parallel clustered statistical cooling by Aarts et al [17]. It was further refined by Lee and Lee who introduced an asynchronous approach to this methodology, and in particular applied the algorithm to the graph partitioning problem for shared memory multiprocessors [1]. The algorithm can be understood if the sequential simulated annealing procedure is considered as a search path where moves are proposed and either accepted or rejected depending on particular cost evaluations. The search path is essentially a Markov chain, and parallelization is accomplished by initiating different chains (using different starting seeds) on each processor. Each chain then explores the entire search space by independently performing the annealing perturbation, evaluation and decision steps. Note that unlike with parallel moves approaches, each chain is allowed to perform moves on the entire set of cells and not just a subset. After each processor has completed the annealing schedule, the solutions are compared and the best is selected. Rose, et al used a similar approach with the min-cut algorithm in the high-temperature region [14].

Of course, there is no speedup in this approach since each processor is individually performing the same amount of work as the sequential algorithm. To achieve speedup, we must reduce the number of moves evaluated in each chain by a factor of $\frac{1}{N}$ where N is the number of processors. Since the number of moves determines the runtime of the program, a reduction by a factor of $\frac{1}{N}$ will cause a speedup of N. Obviously, such a reduction alone is not appropriate since the quality will likely decrease accordingly. To take advantage of the presence of multiple processors, some means of interaction between the various chains is necessary.

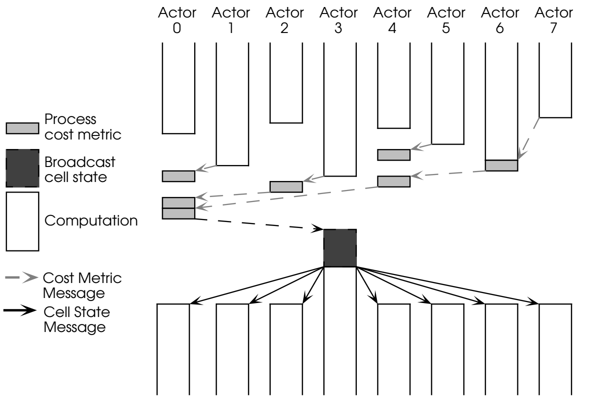

Figure 1: Message flow in actor based synchronous MMC.

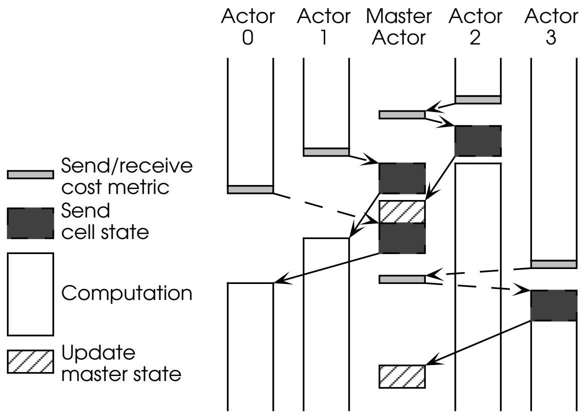

Figure 2: Message flow in actor based asynchronous MMC.

4.1 Synchronous multiple Markov chains

One possible interaction scheme, called *synchronous MMC with periodic exchange* by Lee and Lee, is to periodically compare solutions at fixed intervals. This method allows each Markov chain to update its local database with the best solution, and then continue on. This exchange point serves as the end of a *segment* of computation, and behaves as a synchronization barrier. At the barrier, an application specific metric can then be used to determine the best solution. When applied to `TimberWolfSC`, a natural point for solution exchanges is the end of each `TimberWolfSC` iteration.

In an actor framework, each chain or search path is represented by a separate actor or thread of control. Since the actor model can not assume a shared memory architecture, solution updates must be done with message sends. The barrier at the end of each segment is implicitly achieved through the use of these messages, as shown in Figure 1. When an actor has reached the end of its segment, it propagates a solution metric up to a *master* actor through a reduction tree. This metric is only a cost measurement of the solution and is not the entire global state. The *master* thread will determine the best solution, and then directs the actor with the best solution to broadcast its state to all other actors. In the example in Figure 1, actor 3 is determined to have the best solution.

The barrier could be implemented in a single phase manner if each actor propagates its entire state to the *master* rather than just the cost metric. The *master* could then broadcast the state itself rather than request the winning actor to broadcast it. However, because of the size of the state in cell placement, the two phase method is more efficient. Table 1 shows the message size needed to transfer the relevant circuit state for a sampling of circuits.

4.2 Asynchronous multiple Markov chains

From examining Figure 1, it is clear that barriers can be costly operations; an asynchronous approach is preferred. Very few modifications have to be made to support the asynchronous method. In this approach, the *master* actor does not perform any computation. Instead, it serves as a location for the best available solution at any particular time. When an actor has completed an iteration, it sends its solution metric to the *master* actor, and requests the best solution available. The master thread on receipt of this request will determine if the received solution is better than the local "best" solution. If it is better, the *master* will ask the requestor to send its state back. Remember that the requestor had only sent the metric initially. The requestor will then send its local state back to the master and continue with the next iteration with its own local state. If the *master* has determined that the received solution is worse than the best solution, the *master* will simply send the current best state to the requestor. At the cost of dedicating an extra processor for "master" usage, this asynchronous approach can eliminate much of the idle time that was present earlier.

Graphically, this algorithm is illustrated in Figure 2. For example, Actor 2 has finished its segment and sends its solution metric to the *master* which determines that this solution is the best and sends a message back requesting Actor 2's local state. While waiting for Actor 2's state, the *master* has in the mean time received a solution metric from Actor 1. Since it hasn't yet received Actor 2's state, the *master* must compare Actor 1's metric with the previous state. It then determines that Actor 1 has an inferior solution, and sends the previous state back to Actor 1. Because of the asynchronous behavior, Actor 1 was not able to receive the best current solution from Actor 2. Such erroneous updates are acceptable, since the actors can correct themselves at future iterations, and it also provides an opportunity to escape from local minima.

A straightforward analysis of the asynchronous MMC algorithm arrives at a run time of $t_{par} = \frac{I}{N}(t_{me} + t_d + t_u) + t_c$ where t_c is the communication time. Since the time to send the solution metric is small, t_c is dominated by the time to send the state. If t_c is relatively small, it is clear that the speedup for asynchronous MMC is perfectly linear, i.e, $S = N$. However, t_c is not necessarily small, and we must determine its effect. Examining the data in Table 1 yields 80 as an approximate ratio of state size to number of cells. Since the state is communicated at every one of `TimberWolfSC`'s 160 iterations, $t_c \approx 12800Ct_b$, where C is the number of cells and t_b is the per byte communication cost. With I, the number of total moves, set to $4000C(\frac{C}{500})^{\frac{1}{3}}$ for `TimberWolfSC`, the speedup can then be stated as

$$S = \frac{t_{ser}}{t_{par}} = \frac{5(\frac{C}{500})^{\frac{1}{3}}(t_{me} + t_d + t_u)}{\frac{5}{N}(\frac{C}{500})^{\frac{1}{3}}(t_{me} + t_d + t_u) + 16t_b}$$

On an Intel IPSC/860, $t_{me} + t_d + t_u$ is on the order of 50 to 100 μs, while t_b is only 0.36 μs. Thus it can be seen that t_c is negligible, and the expected speedup should approach N.

Table 4: Multiple Markov chains (Sun 4/690MP)

	2 procs		3 procs		4 procs	
Circuit	W	S	W	S	W	S
fract	1.00	1.00	0.95	1.90	0.95	2.50
s298	1.00	1.00	0.99	1.78	0.99	2.53
s420	1.00	1.00	0.98	1.90	1.04	2.50
primary1	1.00	1.00	1.01	1.45	0.99	2.17
struct	1.00	1.00	1.05	1.67	1.05	2.53
primary2	1.00	1.00	0.99	1.78	1.01	2.10
industry1	1.00	1.00	1.00	1.78	1.01	2.25

Table 5: Multiple Markov chains (Intel iPSC/860)

	2 procs		4 procs		8 procs	
Circuit	W	S	W	S	W	S
fract	1.00	1.00	0.97	2.93	1.04	6.13
s298	1.00	1.00	0.96	2.93	1.03	6.01
s420	1.00	1.00	1.01	2.97	1.05	6.26
primary1	1.00	1.00	1.05	2.88	1.01	5.30

4.3 Experimental observations

We examined the effectiveness of the asynchronous MMC algorithm on the Sun 4/690MP and the iPSC/860 (Tables 4 and 5). We start at 2 processors because of the need for a *master* processor. The quality of the solutions show no degradation as the number of processors increase - in fact, they sometimes show improvements because of the periodic exchange of solutions. The algorithm scales much better on a distributed memory machine because of the lack of memory bus contention. As expected, communication time is insignificant since the number of messages sent is few. Note that the speedup and quality are much better than can be achieved with a pure parallel moves strategy as presented in the previous section.

5 Speculative computation

Another approach recently suggested for generalized parallel simulated annealing is speculative computation [2]. Witte, et al, applied this approach to the task assignment problem and found speedups approaching $log_2\ P$, where P is the number of processors. In this section, we apply the concept of speculative computation to cell placement and determine the applicability of such an approach. We first give a brief description of the algorithm.

5.1 Generalized speculative computation

A sequential simulated annealing schedule is simply a series of move proposals intended to reduce some cost function specific to the particular problem. Each move consists of three parts - the proposal or perturbation, evaluation, and decision. Only after these three parts are completed, is the next move started. Since the decision made by the next move is dependent on the current state as determined by prior moves, simulated annealing is almost inherently serial in nature. Consider the decision tree of moves in Figure 3a. The top node represents a move attempted in a simulated annealing process. There are two possible decisions as a result of this move - acceptance or rejection. Speculative computation will assign two different processors to speculatively work on the two possibilities before the parent move has completed. The rejection child can start at the same time as the parent, since it will assume that the state has not changed. After the parent has completed the move proposal, it can then relay the new state to the accept child.

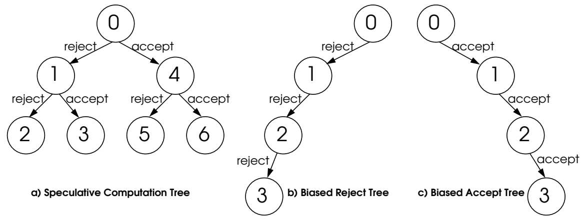

Figure 3: Speculative computation trees.

As the acceptance characteristics of the procedure varies, the shape of the tree can also change. For example, if the acceptance rate is high, it would make sense to generate a linear tree of only acceptance nodes, and likewise a very low acceptance rate would imply the creation of only rejection nodes (Figures 3b and c). This latter mode is essentially the algorithm proposed by [11].

5.2 Speculative computation for placement

Since speculative computation seems to be a promising avenue to achieve at least some speedup in the high temperature region, we decided to investigate the applicability of such an approach to the cell placement problem. The problem fits naturally into an actor based framework, in that each speculated move can be represented by an actor. One of the major changes we made to the algorithm was to add some asynchronous behavior by removing the need for a centralized root processor that was required to start off each tree. Eliminating this synchronization point allows multiple speculative trees to be active at once.

Another major modification made to Witte's algorithm was to transfer just the move proposal to the accept child rather than the entire state after the move. That is, if the root node were to propose moving a cell to a new location, it would convey the cell number and new location to the accept child, and the child would be responsible for duplicating the move. As with multiple Markov chains, this decision was made because of the potentially large size of the cell state. Once a speculative move has been determined to be false, the actor responsible for the move must then abort its move as well as all its parent's moves. It must then update its cell database with the moves from the correct path.

After the modifications were made, the algorithm was run on a variety of circuits as shown in Table 6. As can be seen, the wirelengths are identical, as expected. The speedups, however, are disappointingly poor, as indicated by the drastic slowdowns. The primary reason for this behavior is the faultiness of Witte's model with respect to cell placement, particularly `TimberWolfSC`. Witte makes two main assumptions - the time to perform and propose a move is small compared to the evaluation of the move, and secondly, the outcome of that move or the resultant state is easily communicable. Consider the following optimistic analysis - for

Table 6: Speculative computation (Sun 4/690MP)

	1 procs		2 procs		4 procs	
Circuit	W	S	W	S	W	S
fract	1.00	1.00	1.00	0.30	1.00	0.22
s298	1.00	1.00	1.00	0.25	1.00	0.18
primary1	1.00	1.00	1.00	0.23	1.00	0.20

parallel speculative computation, the execution time is

$$t_{par} = \frac{I}{L}((L-1)(t_c + \alpha t_{sm}) + t_{me} + t_d + t_u)$$

where L is the expected length of the tree, t_c is the time required for communicating a proposal, and t_{sm} is the time required to perform a parent's move. The speedup is

$$S = \frac{L(t_{me} + t_d + t_u)}{(L-1)(t_c + \alpha t_{sm}) + t_{me} + t_d + t_u}$$

The term $(L-1)(t_c+\alpha t_{sm})$ is the cost due to performing parent's moves. If this term is small, then the speedup is simply L, which in the extreme temperature regions is N, and at worst is $logN$. However, our measurements have shown that this cost is not negligible. The speculative move time (t_{sm}), the time that a child node needs to perform its parent's move, is essentially t_{mu}, the update time discussed in Section 3.2. Our measurements show that this time is comparable with a non-speculative move. The other component, α, is also very high due to the relatively high acceptance rate of `TimberWolfSC`. The combination of these factors leads to a very high overhead for parallel speculative computation. In light of these problems, it is clear that speculative computation, though presented as a generalized parallel simulated annealing algorithm, is not a feasible approach to parallelizing cell placement.

6 Conclusions

Most previous approaches to parallel placement have used some form of parallel moves. In this paper, we have investigated two new approaches that have been proposed for generalized parallel simulated annealing, and examined their practicality to simulated annealing based placement algorithms. The asynchronous Markov chains approach is effective and gives good speedups with better quality results. Speculative computation, on the other hand, does not appear to be as promising, due to the characteristics of the cell placement problem, namely, relatively high acceptance rate, large state size, expensive move computation cost relative to evaluation and decision, and early rejection.

Acknowledgements

Our sincere thanks to Dr. Carl Sechen for providing us with the `TimberWolfSC` 6.0 placement code, and to the Argonne National Laboratories for access to their Intel iPSC/860. The authors would also like to thank Dr. Sungho Kim and Dr. Steven Parkes for their assistance in the development of the algorithms for ProperCAD II.

References

[1] S.-Y. Lee and K.-G. Lee, "Asynchronous communication of multiple Markov chains in parallel simulated annealing," in *Proceedings of the International Conference on Parallel Processing*, (St. Charles, IL), pp. III:169–176, Aug. 1992.

[2] E. E. Witte, R. D. Chamberlain, and M. A. Franklin, "Parallel simulated annealing using speculative computation," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 483–494, Oct. 1991.

[3] B. Ramkumar and P. Banerjee, "ProperCAD: A portable object-oriented parallel environment for VLSI CAD," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 829–842, July 1994.

[4] S. Parkes, J. A. Chandy, and P. Banerjee, "A library-based approach to portable, parallel, object-oriented programming: Interface, implementation, and application," in *Supercomputing '94*, (Washington, DC), pp. 69–78, Nov. 1994.

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

[6] 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.

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

[8] J. Lam and J.-M. Delosme, "Performance of a new annealing schedule," in *Proceedings of the Design Automation Conference*, pp. 306–311, 1988.

[9] C. Sechen and K.-W. Lee, "An improved simulated annealing algorithm for row-based placement," in *Digest of Papers, International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 478–481, Nov. 1987.

[10] D. R. Greening, "Parallel simulated annealing techniques," *Physica*, vol. D42, pp. 293–306, 1990.

[11] S. A. Kravitz and R. A. Rutenbar, "Placement by simulated annealing on a multiprocessor," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 534–549, July 1987.

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

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

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

[15] W.-J. Sun and C. Sechen, "A loosely coupled parallel algorithm for standard cell placement," in *Digest of Papers, International Conference on Computer-Aided Design*, (San Jose, CA), pp. 137–144, Nov. 1994.

[16] S. Kim, J. A. Chandy, S. Parkes, B. Ramkumar, and P. Banerjee, "ProperPLACE: A portable parallel algorithm for cell placement," in *Proceedings of the International Parallel Processing Symposium*, (Cancun, Mexico), pp. 932–941, Apr. 1994.

[17] E. H. L. Aarts, F. M. J. de Bont, E. H. A. Habers, and P. J. M. van Laarhoven, "Parallel implementations of the statistical cooling algorithm," *Integration, the VLSI journal*, vol. 4, pp. 209–238, Sept. 1986.