A PARALLEL TABU SEARCH ALGORITHM FOR VLSI STANDARD-CELL PLACEMENT

Sadiq M. Sait *Habib Youssef* *Hassan Barada* *Ahmad Al-Yamani*

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
sadiq@ccse.kfupm.edu.sa

ABSTRACT

VLSI standard-cell placement is an NP-hard problem to which various heuristics have been applied. In this work, tabu search placement algorithm is parallelized on a network of workstations using PVM. The objective of the algorithm is to achieve the best possible solution in terms of interconnection length, overall area of the circuit, and critical path delay (circuit speed). Two parallelization strategies are integrated: functional decomposition strategy and multi-search threads strategy. In addition, domain decomposition strategy is implemented probabilistically. The performance of each strategy is observed and analyzed.

1. INTRODUCTION

Cell placement consists of finding suitable locations for all cells on the final layout of a VLSI circuit. It is a hard combinatorial optimization problem with a number of noisy objective functions. A category of algorithms which were found effective in dealing with this class of problems are iterative search heuristics such as *Simulated Annealing* (SA) [1], *Genetic Algorithm* (GA) [2], *Simulated Evolution* (SE) [3] and *Tabu Search* (TS) [4]. In this work, we propose a *parallel tabu search* algorithm to address this problem.

A placement solution is evaluated with respect to three main objectives: area, wire length, and critical path delay. Prior to final layout these criteria cannot be accurately measured. Further, it is unlikely that a placement that optimizes all three objectives exists. Designers usually have to make tradeoffs. To deal with such complex objectives, we resort to the goal-directed search approach proposed in [5].

In the scheme used, the *acceptable solution* set is modeled as a fuzzy set. For VLSI cell placement problem of minimizing three parameters, the following rule is used to determine the membership in the fuzzy set *acceptable solution*:

If a solution is *within acceptable wire length* AND *within acceptable delay* AND *within acceptable width* **THEN** it is an acceptable solution.

Using fuzzy algebraic notation, while adopting the AND-like ordered weighted averaging operator of Yager [6], the above rule is expressed as follows:

$$\mu(x) = \beta \times min(\mu_1(x), \mu_2(x), \mu_3(x)) + (1-\beta) \times \frac{1}{3}\sum_{i=1}^{3}\mu_i(x)$$

where, $\mu(x)$ is the membership value for solution x in the fuzzy set *acceptable solutions*, and β is an averaging constant. μ_i for $i = 1, 2, 3$ represents the membership values of solution x in the fuzzy sets *within acceptable wire length*, *within acceptable circuit delay*, and *within acceptable width* respectively. The membership function of the fuzzy set corresponding to a particular objective 'i' is shown in Figure 1. The solution which results in the maximum value of $\mu(x)$ is reported as the best solution found.

2. TABU SEARCH

Tabu Search starts with an initial solution s selected randomly or using any constructive algorithm. It then defines a subset $V^*(s)$, called candidate list, of its neighborhood $N(s)$. The algorithm selects the best solution in $V^*(s)$ (in terms of an evaluation function) call it s^*, to be considered as the next solution. If the short term memory does not define the move leading to s^* as *tabu*, it is accepted as the new solution even if it is worse than the current solution in terms of the evaluation function. However, if the move leading to s^* is *tabu*, the solution is not accepted unless it has some feature that makes

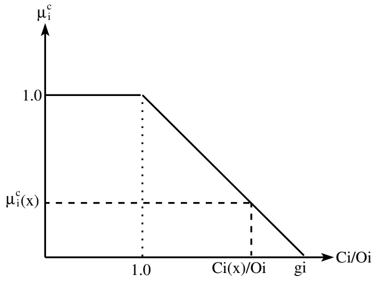


Figure 1: The membership function *within acceptable criterion i*. $C_i(x)$ is cost of solution x and O_i is the lower bound estimate of objective i.

the algorithm override its tabu status to accept it. *Aspiration criterion* is used to check whether the tabu solution is accepted or not [7]. In this work a move consists of the swapping of two cells. N_v pairs of cells are trial swapped and the best swap among them is considered for the next move. A compound move can be made for d times where each time N_v other moves are tested, where d is the desired move depth, and the best move is taken each time. The algorithm checks if the move is tabu by considering only the two cells that were swapped first in the compound move. If the move is found tabu, the *aspiration criterion* is checked. If the move satisfies it, it is accepted; otherwise, it is rejected and the process repeats. The *tabu tenure* i.e., number of iterations for which the move remains tabu is a parameter of the circuit size. If the tabu move leads to a solution whose cost is better than all of those seen so far, then the algorithm overrides the tabu status and the move is accepted.

3. PARALLEL TABU SEARCH FOR STANDARD-CELL PLACEMENT

The algorithm is parallelized on two levels simultaneously. The higher one is at the *Tabu Search* process level where a master starts a number of *Tabu Search Workers* (TSWs) and provides each with the same initial solution (*multi-search threads*). The lower level is the *Candidate List* construction level where each TSW starts a number of *Candidate List Workers* (CLWs), this is *functional decomposition*. The general structure of the parallel algorithm is shown in Figure 2.

The parallel search proceeds as follows. The master initiates a number of TSWs to perform *Tabu Search* starting from the given initial solution. A TSW gets all parameters and the initial solution from the master. It then performs a diversification step where each TSW

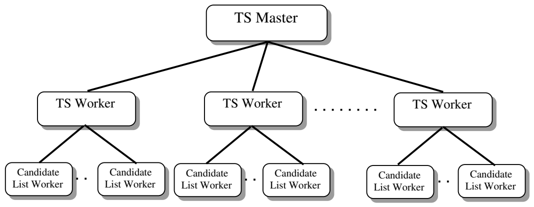


Figure 2: Paradigm of tabu search parallel implementation.

N_i	:	Number of iterations.
X	:	Set of feasible solutions.
bs	:	Current best solution.
bc	:	Current best cost.
$\mathbf{TL}$	:	Tabu list.
N_w	:	Number of workers.

1. Start with an initial feasible solution $bs \in X$.
2. Initialize $\mathbf{TL}$ and bc.
3. Spawn N_w TSW workers to perform Tabu Search.
4. Send($bs, \mathbf{TL}, bc$) to all TSWs.
5. **For** N_i **Do**
6.   Wait for best cost from all workers.
7.   Ask for bs and $\mathbf{TL}$ from the worker that has the overall best.
8.   Receive($s, \mathbf{TL}$).
9.   Update bc.
10.   Send($bs, \mathbf{TL}, bc$) to all workers except sender.
11.   Increment iteration number.
12. **EndFor**

Figure 3: Algorithmic description of master process of parallel TS.

diversifies with respect to a different range of cells so as to enforce that TSWs don't search in overlapping areas. Diversification is performed by moves done within the TSW range to a specific depth such that a different initial solution is used at each TSW. Then each TSW starts a number of CLWs to investigate the neighborhood of the current solution. It sends the parameters and the initial solution to each CLW. It also gives each CLW a range of cells to search the neighborhood with respect to those cells. For every move it makes, the CLW has to choose one of the cells from its range and the other cell from anywhere in the whole cell space. Therefore, the probability that two CLWs perform the same move is equal to $\frac{1}{(n-1)^2}$. The probability that more than two CLWs select the same two cells is 0. This means that the probability that k CLWs make the same move is eliminated completely if $k > 2$.

Each CLW makes a compound move of a predetermined depth and keeps computing the gain. If the current cost is improved before reaching the maximum depth, the move is accepted without further investigation. After finding the compound move that improves the cost the most (or degrades it the least), the CLW sends its best solution to the TSW that started it. The TSW selects the best solution from the CLW that achieves the maximum cost improvement (or the least cost degradation). It then checks if the move is tabu. If it is not, it accepts it. Otherwise, the cost of the new solution is checked against the *aspiration criterion* and the process continues for a number of local iterations. At the end of the local iteration count, each TSW sends its best cost to the master process. The master gets the overall best solution and broadcasts it to all TSWs and the process continues for a number of global iterations (See Figures 3 and 4).

N_i	:	Number of iterations.
X	:	Set of feasible solutions.
s	:	Current solution.
s^*	:	Best admissible solution.
bs	:	Current best solution.
C	:	Objective function.
$\aleph(s)$	:	Neighborhood of $s \in X$.
$\mathbf{V}^*$	:	Sample of neighborhood solutions.
TL	:	Tabu list.
AL	:	Aspiration Level.

```
Receive(s, TL, AL) from master.
For N_i Do
        Perform a diversification step.
        Apply short term TS for fixed # of iterations.
        Send AL to master.
        If the master asks for bs Then
                Send(bs, TL) to master.
        Else
                Receive(bs, TL, AL) from master.
                s = bs.
        EndIf
EndFor
```

Figure 4: A CLW worker process of parallel TS.

4. EXPERIMENTAL RESULTS AND DISCUSSION

We used four *ISCAS-89* benchmark circuits *fract, c532, c1355, and struct*, which have 149, 532 1355, and 2000 cells respectively. We studied the effect of the degree of low-level and high-level parallelization on the algorithm performance, namely quality of best solution and speedup. For this category of algorithms, speedup is defined as follows

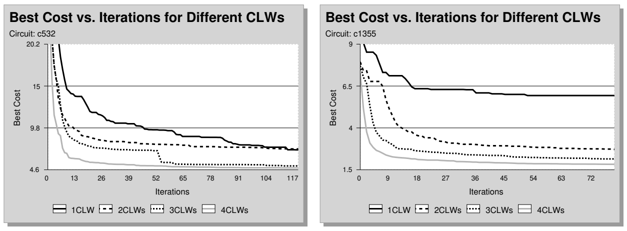
$$Speedup_{(n,x)} = \frac{t_{(1,x)}}{t_{(n,x)}} \qquad (1)$$

where $t_{(1,x)}$ is the time needed to hit an x-quality solution using one *Candidate List Worker* (or *Tabu Search Worker*) and $t_{(n,x)}$ is the time needed to hit the same solution quality using n CLWs (or TSWs). $Speedup_{(n,x)}$ in this case can be greater than n because of the nondeterministic nature of the algorithm.

Effect of Degree of Low-level Parallelization

In this experiment, different numbers of CLWs are tried starting from 1 to 4 for each circuit. The number of TSWs is 4 in all experiments. Twelve machines are used as a parallel virtual machine.

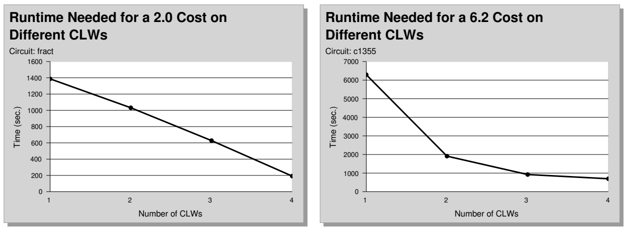
Figure 5 shows the effect of changing the number of CLWs on the best solution quality, it is clear that increasing low level parallelization degree is beneficial.



(a) (b)

Figure 5: Effect of number of CLWs on solution quality (a) Plots for c532 and (b) Plots for c1355.

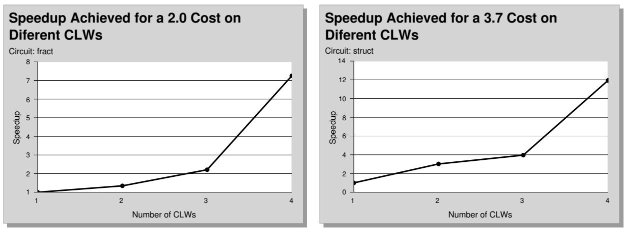
Figure 6 shows the time needed to achieve a specific solution quality for all circuits. In all cases, adding more CLWs resulted in reaching better solutions in less time.



(a) (b)

Figure 6: Runtime needed to achieve a solution of cost less than x for different numbers of CLWs. (a) Plots for *fract* and (b) Plots for c1355.

Figure 7 shows the speedup achieved in reaching a specific solution quality for circuits *fract* and *struct*. The figure shows that as the number of CLWs increases from 1 to 4, the speedup increases and exceeds 4. The sharpness of the speedup increase depends on the circuit size and the goodness of the initial solution. For example, for *fract*, the initial solution is too far from the best reached. As a result, increasing the number of CLWs results in a sharper change in the speedup. For *struct*, the same behavior was observed because the circuit size is large. Because of communication overhead the rate of change in the speedup goes down as the number of CLWs is increased. In all the four experiments, the critical point, where the speedup starts to degrade, is not reached but it is clear in some curves that it is being approached.



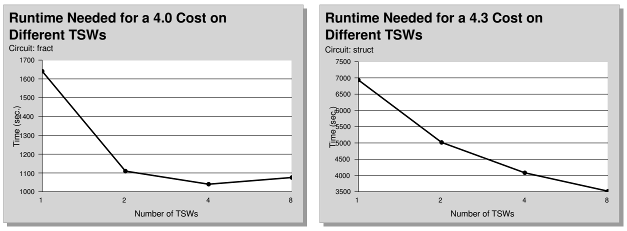
(a) (b)

Figure 7: Speedup achieved in reaching a solution of cost less than x for different numbers of CLWs.

Effect of Degree of High-level Parallelization

In this experiment, different numbers of TSWs are tried starting from 1 to 8 for each circuit. The number of CLWs is fixed to 1 in all experiments. Twelve machines are used as a parallel virtual machine.

Figure 8 shows the time needed to achieve a specific solution quality for two circuits (*fract* and *struct*). Adding more TSWs proved to be beneficial with respect to runtime except for *fract* and *c532* where running 8 TSWs took more time than running 4 TSWs because the circuit sizes are small. This means that communication overhead involved was not compensated for by the quality improvement.

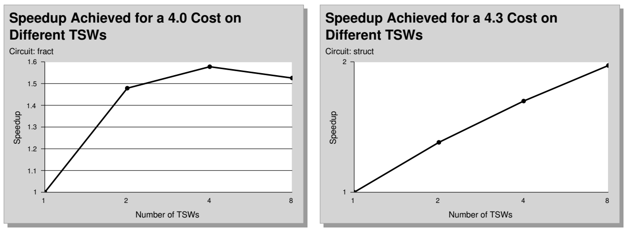


(a) (b)

Figure 8: Runtime needed to achieve a solution of cost less than x for different numbers of TSWs.

Figure 9 shows the speedup achieved in reaching a specific solution quality for two circuits. For *fract* the critical point, occurred at 4 TSWs. Adding more TSWs degraded the speedup. For the other larger circuit (*struct*), the critical point was approached but not reached.

In general, increasing the number of CLWs performed better than increasing the number of TSWs because the CLW is an inner loop for all TSWs running. As a result, the speedup critical point was approached using low-level parallelization faster than high-level parallelization.



(a) (b)

Figure 9: Speedup achieved in reaching a solution of cost less than x for different numbers of TSWs.

Acknowledgment: The authors thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for support.

5. REFERENCES

[1] A. Casotto, F. Romeo, and A. L. Sangiovanni-Vincentelli. A parallel simulated annealing algorithm for the placement of macro-cells. *IEEE Transactions on Computer Aided Design*, CAD-6(5):838–847, September 1987.

[2] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, CAD-6:956–964, November 1987.

[3] R. Kling and P. Bannerjee. ESP: A new standard cell placement package using simulated evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.

[4] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, USA, 1997.

[5] Sadiq M. Sait, Habib Youssef, and Ali Hussain. Fuzzy simulated evolution algorithm for multiobjective optimization of VLSI placement. In *Proceedings of IEEE International Congress on Evolutionary Computation, Washington D.C.*, pages 91–97, July 1999.

[6] Ronald Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions Systems, man, and Cybernetics*, 18(1):183–190, January 1988.

[7] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms and their Applications in Engineering.* IEEE Computer Society Press, 1999.