Parallel Tabu Search in a Heterogeneous Environment

Ahmad Al-Yamani[1] *Sadiq M. Sait*[1] *Hassan Barada*[2] *Habib Youssef*[1]

[1]King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

[2]Etisalat College of Engineering
Emirates Telecommunications Co. (Etisalat)
P.O. Box 980, Sharjah, UAE
hbarada@ece.ac.ae

Abstract

In this paper, we discuss a parallel tabu search algorithm with implementation in a heterogeneous environment. Two parallelization strategies are integrated: functional decomposition and multi-search threads. In addition, domain decomposition strategy is implemented probabilistically. The performance of each strategy is observed and analyzed in terms of speeding up the search and finding better quality solutions. Experiments were conducted for the VLSI cell placement. The objective was to achieve the best possible solution in terms of interconnection length, timing performance circuit speed, and area. The multiobjective nature of this problem is addressed using a fuzzy goal-based cost computation.

1. Introduction

Tabu Search (TS) belongs to the class of general iterative heuristics that are used for solving hard combinatorial optimization problems. It is a generalization of local search that searches for the best move in the neighborhood of the current solution. However, unlike local search, TS does not get trapped in local optima because it also accepts bad moves if they are expected to lead to unvisited solutions [1].

Among the iterative stochastic heuristics applied to combinatorial optimization problems are Simulated Annealing (SA) [2, 3], Genetic Algorithm (GA) [4] and Simulated Evolution (SE) [5]. A common feature of these stochastic iterative heuristics is that they are memoryless. They do not have memory or use any memory structure to keep track of previously visited solutions. On the other hand, Tabu Search (TS) utilizes some memory to make decisions at various stages of the search process [6]. Memory structures are used to prevent reverses of recent moves by keeping their attributes in a tabu list (also known as short-term memory) in order to prevent cycling back to already visited solutions. Memory structures are also used to (1) force new solutions to have different features from previously visited ones (*diversification*); (2) force the new solution to have some features that have been seen in recent good solutions (*intensification*).

Because of its search strategy, the parallelization of TS can result in improved solution quality and reduced execution time. Encouraging results are obtained for computationally intensive tasks even with a small number of workstations in a local area network LAN. However, most LANs today consist of a set of heterogeneous workstations. Therefore, in order to use LANs efficiently, parallel algorithms have to be designed such that the heterogeneity of system is taken into account. In this paper, we discuss the parallelization of the tabu search algorithm in a heterogeneous environment. We implement different parallelization strategies on a cluster of workstations using the PVM tool [7]. Experiments were conducted for the VLSI cell placement, an NP-hard problem.

2. VLSI Cell Placement

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 solution is evaluated with respect to three main objectives: wire length, critical path delay, and area, which is a function of cell delays and interconnection delays. 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 and imprecise objectives, a fuzzy goal-directed search approach is applied [5].

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

3. Tabu Search

TS 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. However, if the move leading to *s** is *tabu*, the solution is not accepted unless a certain criterion, *aspiration criteria,* is satisfied [8]. A move, in our problem, consists of swapping two cells on the layout of a VLSI circuit. *m* pairs of cells are trial swapped and the best swap among them is taken as the next move. A compound move can be made *d* times where each time *m* pairs are tested, where *d* is the desired move depth. The best move is taken each time. The basic description of TS is shown in Figure 1.

```
Algorithm Tabu_Search;
    X    :  Set of feasible solutions.
    s    :  Current solution.
    s*   :  Best admissible solution.
    C    :  Objective function.
    N(s) :  Neighborhood of s ∈ X.
    V*   :  Sample of neighborhood solutions.
    TL   :  Tabu list.
    AL   :  Aspiration Level.

1.  Start with an initial feasible solution s ∈ X.
2.  Initialize tabu lists and aspiration level.
3.  For fixed number of iterations Do
4.      Generate neighbor solutions V* ⊂ N(s).
5.      Find best s* ∈ V*.
6.      If move s to s* is not in TL Then
7.          Accept move and update best solution.
8.          Update tabu list and aspiration level.
9.          Increment iteration number.
10.     Else
11.         If C(s*) < AL Then
12.             Accept move and update best solution.
13.             Update tabu list and aspiration level.
14.             Increment iteration number.
15.         EndIf
16.     EndIf
17. EndFor
End. (* Tabu_Search *)
```

Figure 1. Algorithmic description of TS.

4. Classification of Parallel Tabu Search

According to Crainic *et. al* taxonomy [9], a possible parallelization strategy of *tabu search* is to distribute the computation that requires the most CPU time on available machines (*functional decomposition*). Another strategy is to perform many independent searches (*multi-search threads*). A third strategy is to decompose the search space among processes (*domain decomposition*). Using a different taxonomy, Crainic et. al., classify TS along three dimensions. The first dimension is *control cardinality* where the algorithm is either *1-control* or *p-control*. In a *1-control* algorithm, one processor executes the search and distributes numerically intensive tasks on other processors. In a *p-control* algorithm, each processor is responsible for its own search and the communication with other processors. The second dimension is *control and communication type* where the algorithm can follow a *rigid synchronization* (*RS*), a *knowledge synchronization* (*KS*), a *collegial* (*C*), or a *knowledge collegial* (*KC*) strategy. *RS* and *KS* correspond to synchronous operation mode where the process is forced to exchange information at specific points; *C* and *KC* correspond to asynchronous operation modes where communication occurs at regular intervals. Collegial approaches exchange more information than non-collegial ones. The third dimension is *search differentiation* where the algorithm can be *single point single strategy (SPSS)*, *single point different strategies (SPDS)*, *multiple points single strategy (MPSS)*, or *multiple points different strategies (MPDS)*.

4.1 Proposed Algorithm for Cell Placement

The proposed parallel Tabu search algorithm (PTS) consists of three types of processes: (i) a master process, (ii) Tabu Search Workers (TSWs), and (iii) Candidate list Workers (CLWs). The algorithm is parallelized on two levels simultaneously. The upper one is at the TS process level where a master starts a number of TSWs and provides each with the same initial solution. The lower level is the *Candidate List* construction level (local neighborhood search) where each TSW starts a number of CLWs.

The parallel search proceeds as follows. The master initiates a number of TSWs to perform TS 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 diversifies with respect to a different subset 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 initial solution after diversification. It sends the

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

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 $\frac{1}{(n-1)^2}$ where n is the number of cells. The probability that more than 2 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 its parent TSW. 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 fixed number of global iterations. The completion of all iterations by the TSWs and selection of new current solution by the TS master is considered one global iteration. The TS iterations executed by each TS worker are called local iterations.

The processes described in Figures 2, 3, and 4, work together to get a high quality solution with minimum communication between them. A TSW process and a CLW process exchange only the best solution between them while the master and TSW exchange the best solution as well as the associated tabu list.

4.2 PTS in a Heterogeneous Environment

We have implemented our proposed PTS algorithm on a network of heterogeneous workstations using the PVM tool. In our implementation, we account for speed and load heterogeneity by letting the master receive the best cost from any TSW that has finished the local iterations. Once the number of TSWs that gave their best cost to the master reaches half the total number of TSWs, the master sends a message to all other TSWs forcing them to report whatever best cost they have achieved. The same approach is followed in the communication between TSWs and their own CLWs.

```
Algorithm Parallel_TS_Master_Process;
    N_i  : Number of iterations.
    X    : Set of feasible solutions.
    bs   : Current best solution.
    bc   : Current best cost.
    TL   : Tabu list.
    N_w  : Number of workers.
1.  Start with an initial feasible solution bs ∈ X.
2.  Initialize TL and bc.
3.  Spawn N_w TSW workers to perform Tabu Search.
4.  Send(bs, TL, bc) to all TSWs.
5.  For N_i Do
6.      Wait for best cost from all workers.
7.      Ask for bs and TL from the worker
        that has the overall best.
8.      Receive(bs, TL).
9.      Update bc.
10.     Send(AL, bs, TL, bc) to all workers except sender.
11.     Increment iteration number.
12. EndFor
End. (*Parallel_TS_Master_Process*)
```

Figure 2. Master process of parallel TS

```
Algorithm TSW;
    GI  : Number of global iterations.
    LI  : Number of local iterations.
    cs  : Current solution.
    bs  : Current best solution.
    bsi : Best solution sent by CLW (i).
    TL  : Tabu list.
    AL  : Aspiration Level.

1.  Receive(cs, TL) and a range from master.
2.  For GI Do
3.      Perform a diversification step.
4.      For LI Do
5.      Send cs and a unique range of cells to each CLW.
6.      Receive bsi from all CLWs.
7.      Check if the best bsi is tabu.
8.          if no, update cs to the best bsi.
9.          if yes, check if it satisfies aspiration criterion.
10.             if yes, update cs to the best bsi else cs is not changed.
11.     EndFor
12.     If the master asks for bs Then
13.         Send(bs, TL) to master.
14.     Else
15.         Receive(bs, TL) from master.
16.         cs = bs.
17.     EndIf
18. EndFor
End. (*TSW*)
```

Figure 3. TSW process.

```
Algorithm CLW;
    LI  : Number of local iterations.
    cs  : Current solution.
    TL  : Tabu list.
    AL  : Aspiration Level.

1.  Receive(cs, TL) and a range of cells from TSW.
2.  For LI Do
3.      Try m random pairs of cells of which one has to belong to the range.
4.      Perform best swap and update cs.
5.      Send cs to TSW.
6.      If the TSW asks for cs Then
7.          Send(cs, TL) to TSW.
8.      Else
9.          Receive(cs, TL) from TSW.
10.     EndIf
11. EndFor
End. (*CLW*)
```

Figure 4. CLW process.

4.3 Classification of PTS

As mentioned earlier, the algorithm is parallelized on two levels simultaneously. The upper one is at the *tabu search* process level where a master starts a number of TSWs and provides them with the same initial solution. This is a *multi-search threads* approach where each TSW performs its own search. The lower level is the *Candidate List* construction level where each TSW starts a number of CLWs. This level belongs to the strategy of *functional decomposition* because CLWs are spawned only to investigate the neighborhood of the current solution. The algorithm falls into *p-control* class at the higher parallelization level because the search control is distributed among all TSWs. The lower level parallelization belongs to the *1-control* class because the TSW controls the search done by its CLWs. On the *control and communication type* dimension, the algorithm follows *rigid synchronization* because the master waits for its children or stops them. It is a *multiple points single strategy* (*MPSS*) search on the *search differentiation* dimension because TSWs diversify from the initial solution at each global iteration using the diversification scheme proposed by Kelly et. [10].

5. Experiments and Discussion

We present and discuss various experiments that are performed using the proposed parallel *tabu search* algorithm for VLSI standard cell placement. Experiments were conducted on three different speed levels of machines and four different architectures. Four ISCAS-89 benchmark circuits of different sizes were used in the experiments. These circuits are: *highway* (56 cells), *c532* (395 cells), c1355 (1451 cells), and *c3540* (2243 cells).

In the paper, we study the effect of the degree of low-level and high-level parallelization on the algorithm performance, namely quality of best solution and speedup. We also study the effect of diversification performed by TSWs and the effect of heterogeneity of the environment. The definition of speedup for non-deterministic algorithms such as TS is different from that used for deterministic constructive algorithms. For this category of algorithms, speedup is defined as:

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

where $t_{(1,x)}$ is the time needed to hit an x-quality solution using one CLW (or TSW) and $t_{(n,x)}$ is the time needed to hit the same solution quality using n CLWs (or TSWs).

5.1 Effect of Low-level Parallelization

In this experiment, different number of CLWs is tried, from 1 to 4, for each circuit. The change in the best solution quality is monitored as the number of CLWs is changed. All other algorithm parameters are fixed. 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 for the four circuits. For most of the circuits, it is clear that increasing the degree of low level parallelization is beneficial. For *highway*, the circuit size is small. That makes adding CLWs beyond 2 not useful. Figure 6 shows the speedup achieved in reaching a specific solution quality for 2 of the circuits. It is clear from the figure that in most of the experiments, as the number of CLWs increases from 1 to 4, the speedup increases. The sharpness of the speedup increase depends on the circuit size and the goodness of the initial solution.

5.2 Effect of High-level Parallelization

In this experiment, different numbers of TSWs are tried, from 1 to 8, for each circuit. The change in the best solution quality is monitored as the number of TSWs is changed. The number of CLWs per TSW is fixed to 1 in all experiments. As mentioned earlier, 12 machines are used as a parallel virtual machine. Figure 7 shows the effect of changing the number of TSWs on the best solution quality for all circuits. It is clear that, for all circuits, adding TSWs beyond 4 is not useful. Figure 8 shows the speedup achieved in reaching a specific solution quality for two of the circuits. For *c532*, and *c3540* the critical point, occurred at 4 TSWs. Adding more TSWs degraded the speedup.

5.3 Effect of Diversification

In this experiment, we try to see the effect of the diversification step performed by the TSWs at the beginning of each global iteration. Figure 9 shows a comparison between two runs of four TSWs and one CLW per TSW. In one run, diversification is done while in the other run, no diversification is performed. It is clear from the figure, that the diversified run outperforms the non-diversified run significantly.

The message conveyed in Figure 9 is that some diversification is always useful. However, it is known that too much diversification without enough local investigation might mislead the search by making it jump from place to another without enough investigation any where. Figure 10 shows the results of an experiment where the number of global iterations is decreased (less diversification) as the number of local iterations is increased (more local investigation) for all circuits. It is clear from the figure that no general conclusion can be made about the best number of global iterations versus local iterations. It all depends on the problem instance itself. This experiment is used as a guide for the most

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

suitable number of local and global iterations that should be used to continue searching for the best achievable solution and to achieve the highest speed.

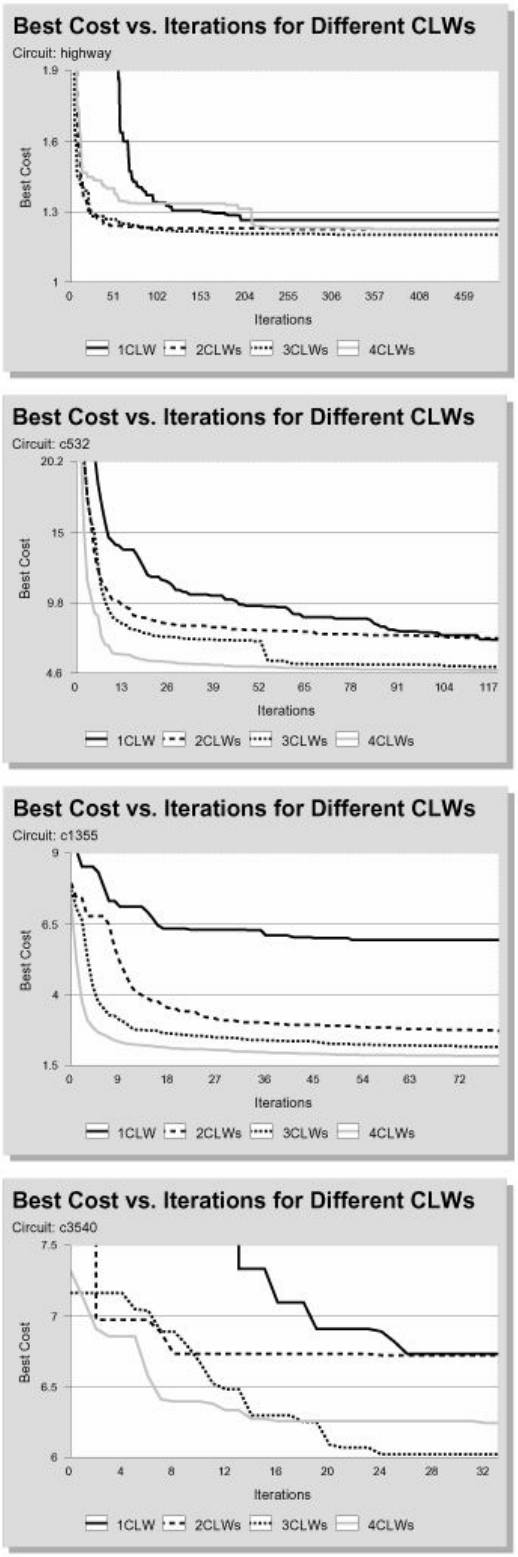


Fig. 5. Effect of number of CLWs on solution quality.

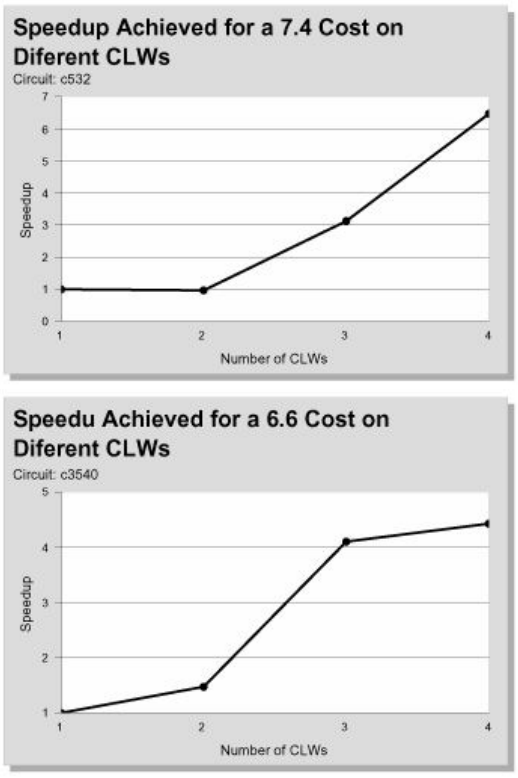


Fig. 6. Speedup achieved in reaching solution of cost less than x for different number of CLWs.

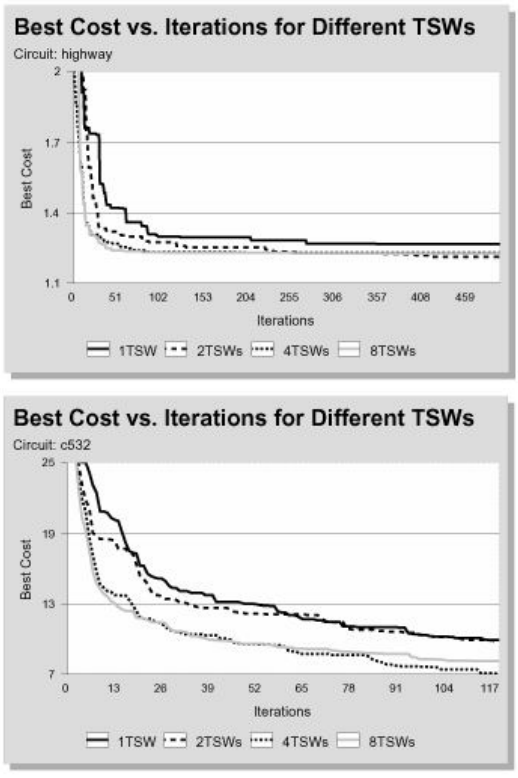
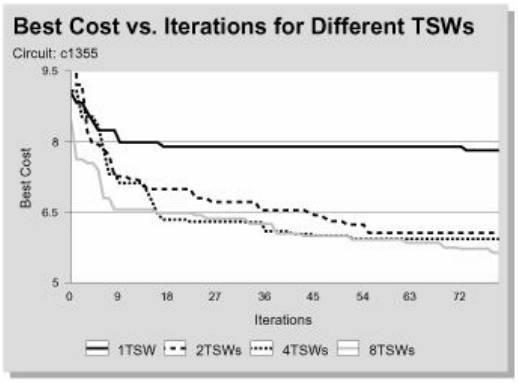


Fig. 7. Effect of number of TSWs on solution quality.



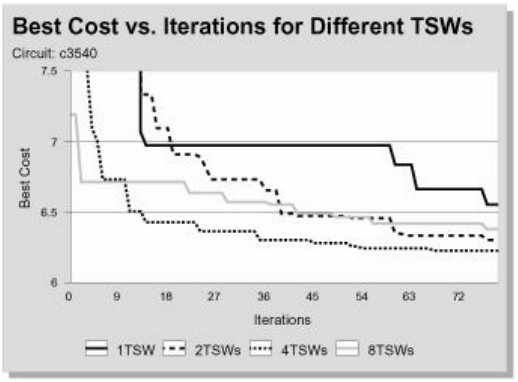
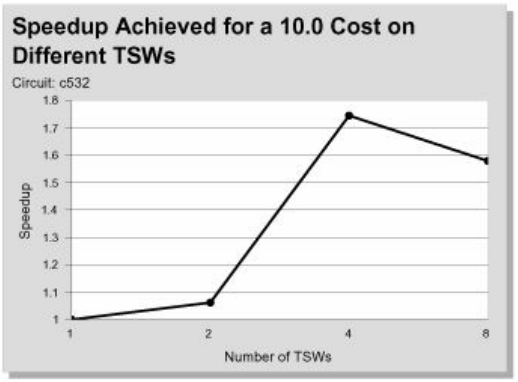


Fig. 7. Effect of number of TSWs on solution quality.



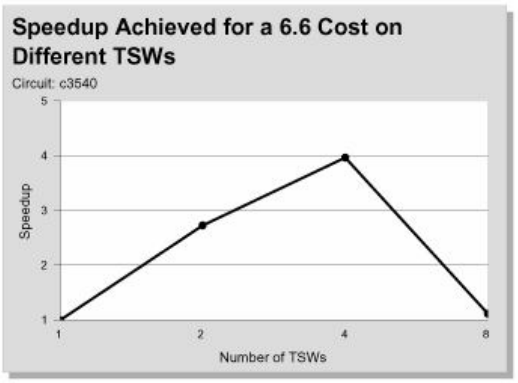
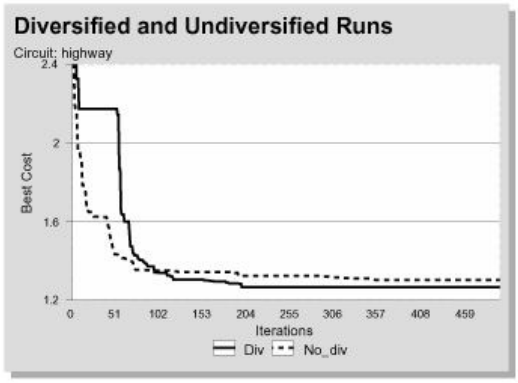
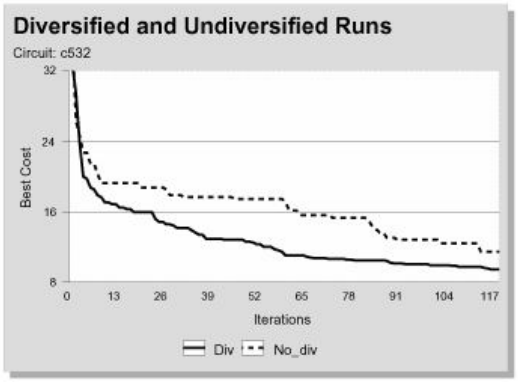
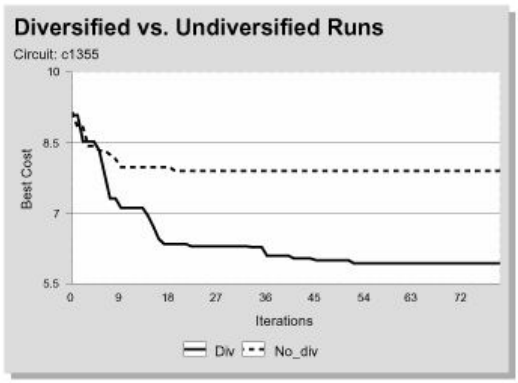


Fig. 8. Speedup achieved in reaching a solution of cost less than x for different numbers of TSWs.







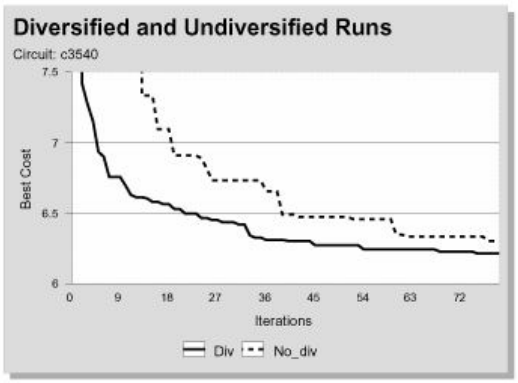


Fig. 9. Effect of diversification.

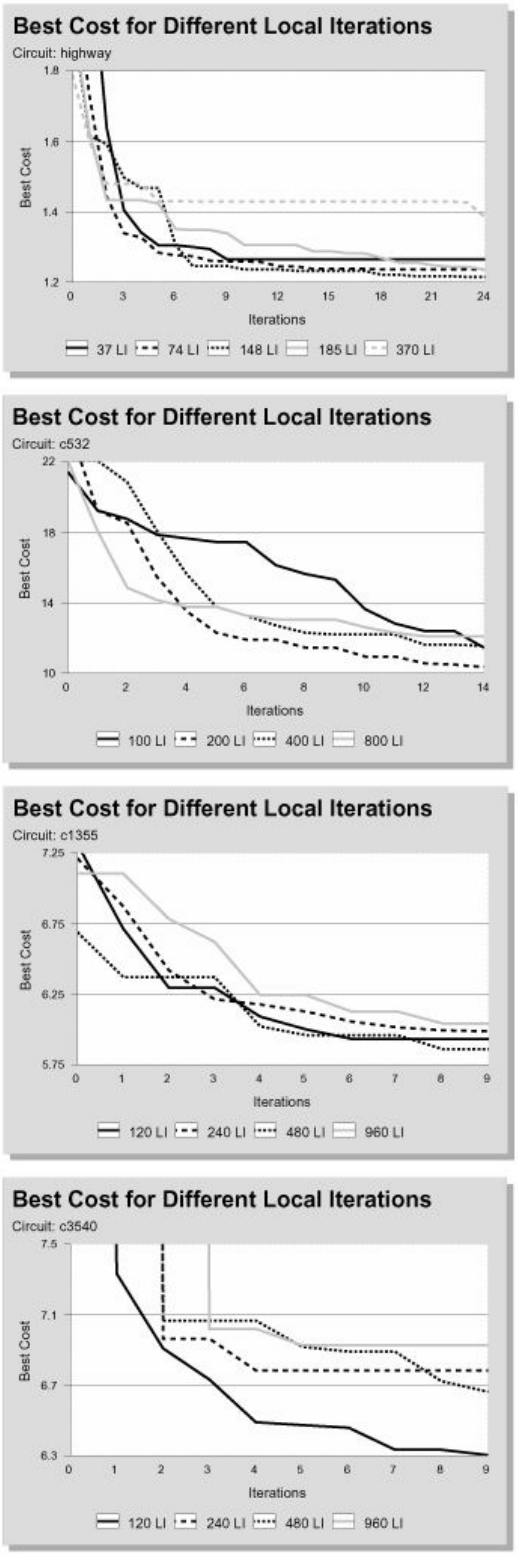


Fig. 10. Local versus global iterations.

5.4 Accounting for Heterogeneity

In this experiment, we try to see the effect of accounting for speed and load differences of various machines by performing two runs. In the first one, heterogeneous run, we run the algorithm while accounting for heterogeneity by making the master ask for best solutions from all TSWs once half of them complete all assigned iterations, and report their best to their parent. TSWs do the same by asking their CLWs to submit their best solutions once half of them report their best to the parent. In the second run, homogeneous run, each parent waits for all its child processes to finish and return their new best. In all experiments we used twelve machines to make the parallel virtual machine. These machines include seven high-speed machines, 3 medium-speed machines, and 2 low-speed machines.

In both runs, we use 4 TSWs and 4 CLWs per TSW. The run that does not account for heterogeneity is supposed to give better solutions because the parent waits for all of its children to give their best solutions. However, since the number of global iterations is maintained the same for both cases, the heterogeneous run-time is expected to be far less than the homogeneous runtime. Figure 11 shows the best quality of solution achieved versus runtime for the homogeneous and heterogeneous runs. For the three circuits shown here, we observed no noticeable differences in solution quality. Figure 11 shows that towards the end of experiment, the heterogeneous run is doing either better than or at least as good as the homogeneous run, but never performs worse.

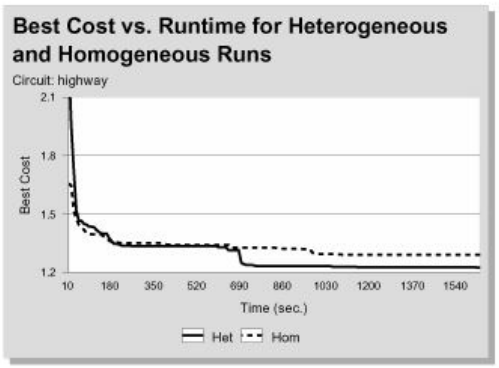
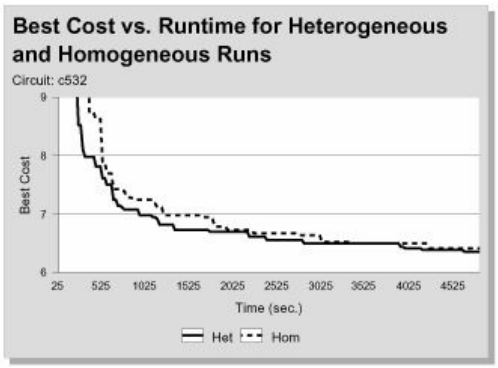


Fig. 11. Best cost versus runtime for heterogeneous and homogeneous runs



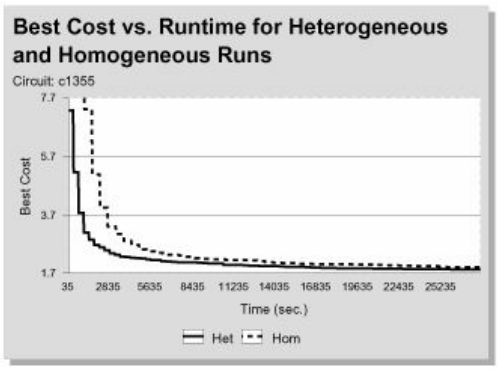


Fig. 11 (cont.). Best cost versus runtime for heterogeneous and homogeneous runs.

6. Summary of results

The goal of parallelization is to speedup the search and to improve solution quality. Observations support that both parallelization strategies are beneficial, with functional decomposition producing slightly better results. Below, we summarize our observations from extensive experiments carried out on circuits of various sizes.

- For most test circuits, increasing the degree of low-level parallelization and the degree of high level parallelization was beneficial. However, in general the most effective strategy seems to be a mix of high and low level parallelization. Low level or high level alone is not as effective.
- In order to achieve a specific solution quality, for all circuits, adding more CLWs or more TSWs to a certain limit resulted in reaching better solutions in less time.
- Speed and load differences of machines are taken into account by making the master ask for best solutions from all TSWs once half of them have completed all iterations. This strategy resulted in higher speedup.

Acknowledgement

The authors would like to thank King Fahd University of Petroleum and Minerals for all the support provided. Dr. Barada would also loke to thank Etisalat College of Engineering for support.

References

[1] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. Annals of Operations Research, 41:3-28, 1993.

[2] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice.* McGraw-Hill Book Co., Europe, 1995.

[3] 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*, 6(5): 838-847, September 1987.

[4] K. Shahookar and P. Mazumder, "A genetic approach to standard cell placement using metagenetic parameter optimization", *IEEE Transactions on Computer Aided Design*, 9(5):500-511, May 1990.

[5] Sait, S.M., H. Youssef, and H. Ali, "Fuzzy Simulated Evolution Algorithm for multi-objective optimization of VLSI placement", *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999, pp. 91-97.

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

[7] Al Geist et. Al.. PVM Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing. The MIT Press, Cambridge, 1994.

[8] Youssef Habib and Sadiq Sait. 1999. *Iterative Algorithms and Their Applications in Engineering*. IEEE Computer Society Press, CA.

[9] T. Crainic, M Toulouse, and M. Gendreau, "Towards a Taxonomy of Parallel Tabu Search Heuristics", *INFORMS Journal of Computing*, 9(1): 61-72, 1997.

[10] J.P. Kelly, M. Laguna, and F. Glover, "A study of diversification strategies for the quadratic assignment problem", *Computers & Operations Research*, 21(8): 885-893, 1994.