HPTS: Heterogeneous Parallel Tabu Search for VLSI Placement

Ahmad Al-Yamani[1] Sadiq M. Sait[1] Hassan R. Barada[2]

[1]Computer Engineering Department,
King Fahd University of Petroleum & Minerals,
Dhahran 31261, Saudi Arabia

[2]Etisalat College of Engineering, Sharjah, UAE

E-mail: yamani@stanford.edu, sadiq@ccse.kfupm.edu.sa, hbarada@ece.ac.ae

Abstract - Parallelizing any algorithm on cluster of heterogeneous workstations is not easy, as each workstation requires different wall clock time to execute the same instruction set. In this work, a parallel tabu search algorithm for heterogeneous workstations is presented using PVM. Two parallelization strategies, i.e., functional decomposition and multi-search thread strategies are integrated. The proposed algorithm is tested on VLSI standard cell placement problem, however, the same algorithm can be used on any combinatorial optimization problem. The results are compared ignoring heterogeneity and are found to be superior in terms of execution time.

I. 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], [2].

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.

In 'Distributed Computing', several interconnected computers work together to solve a large problem [3]. The benefits of creating a distributed computing environment employing a network of workstations has been addressed and investigated. These benefits include: (i) cost effectiveness compared to an expensive large multiprocessor supercomputer alternative [4], (ii) the utilization of the abundant computation power of PCs and workstations that remain idle for a large fraction of time, (iii) the availability of high speed transmission links that have capacities in the order of gigabits/sec [5], etc., to name a few. Researchers from Oak Ridge National Laboratory, the University of Tennessee, and Emory University, developed a Parallel Virtual Machine **(PVM)** system that assists in the implementation of developed parallel algorithms to be executed on a network of heterogeneous workstations. The algorithm is developed as a collection of communicating tasks, where, message passing, format conversion, task scheduling, etc., are handled by PVM [3], [6].

II. Multiobjective 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 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 [7]. In the scheme used, the *acceptable solution* set is modeled as a fuzzy set. For placement problem minimizing 3 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

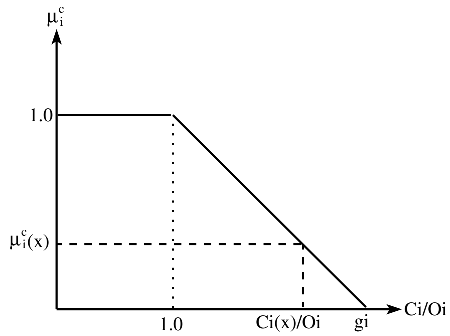

Fig. 1. The membership function *within acceptable criterion i*. $C_i(x)$ is cost of solution x and O_i is the lower bound estimate of objective i.

is an acceptable solution.

Using fuzzy algebraic notation, while adopting the AND-like ordered weighted averaging operator of Yager [8], 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.

III. Parallel Tabu Search Algorithm (PTS)

Sequential 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 the algorithm override its tabu status to accept it. *Aspiration criterion* is used to check whether the tabu solution is accepted or not [1].

The algorithm is parallelized on two levels simultaneously. The higher one is at the **TS** process level where a master starts a number of TSWs (Tabu Search Workers) 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 **CLWs** (Candidate List Workers), this is *functional decomposition*. The general structure of the parallel algorithm is shown in Figure 2.

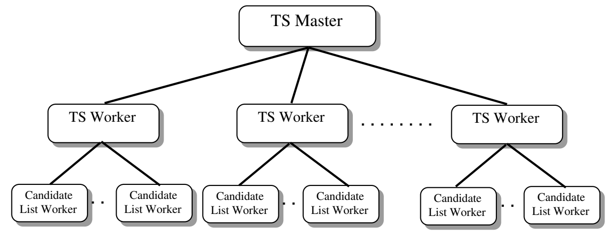

Fig. 2. Paradigm of tabu search parallel implementation.

N_i	:	Number of iterations.
X	:	Set of feasible solutions.
bs	:	Current best solution.
bc	:	Current best cost.
TL	:	Tabu list.
N_w	:	Number of workers.
1.		Start with an initial feasible solution $bs \in X$.
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(s, **TL**).
9.		Update bc.
10.		Send(bs, **TL**, bc) to all workers except sender.
11.		Increment iteration number.
12.		**EndFor**

Fig. 3. Algorithmic description of master process of parallel TS.

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 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

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
```

Fig. 4. A CLW worker process of parallel TS.

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).

IV. PTS On Hetrogeneous Workstations

Normally, a network of workstations is composed of heterogeneous machines. Heterogeneity can be of various types such as the machine architecture, data format, computational speed, network type, machine load, and network load. PVM can take care of machine architecture heterogeneity and data format conversion.

In our implementation of parallel *tabu search*, 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. **TSWs** check for such a message in their buffers periodically (every 10 iterations). Once they receive the message, they kill the currently running **CLWs** and report to the master their best achieved costs.

The same approach is followed in the communication between **TSWs** and their own **CLWs** that check for a message from their parents frequently. That message either kills them, if it is the TS master that is asking the TSW to report, or asks them for their best achieved solutions if half of the CLWs have reported their best. This approach is followed in order to account for the heterogenity in workstation's speeds and loads as well as the varying network load.

Experiments are conducted on three different speed levels of machines and four different architectures. These architectures are IPX/SPARC, SparcStation 10, LX/SPARC and UltraSparc 1. All machines have the same operating system (Solaris 2.5).

V. Experiments and Results

Seven different ISACAS-89 benchmark circuits are used in the experiment. In this experiment, the effect of accounting for speed and load heterogeneity of various machines was seen by performing two runs. In the first one (heterogeneous run), the algorithm was run while accounting for speed and load 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. This is a knowledge collegial mode of operation with respect to the control and communication dimension. In the second run (homogeneous run), each parent waits for all its child processes to finish and return their new best. In all experiments twelve machines were used to make the Parallel Virtual Machine. These machines include seven high-speed machines (UltraSparc 1), 3 medium-speed machines (SparcStation 10), and 2 low-speed machines (LX/SPARC), all running the same operating system (Solaris 2.5) and interconnected by a 10BaseT Ethernet segment.

PVM takes care of distributing processes between machines. In both runs, 4 TSWs and 4 CLWs per TSW were used. 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 and does not force any one to stop searching because others have finished. However, since the number of global it-

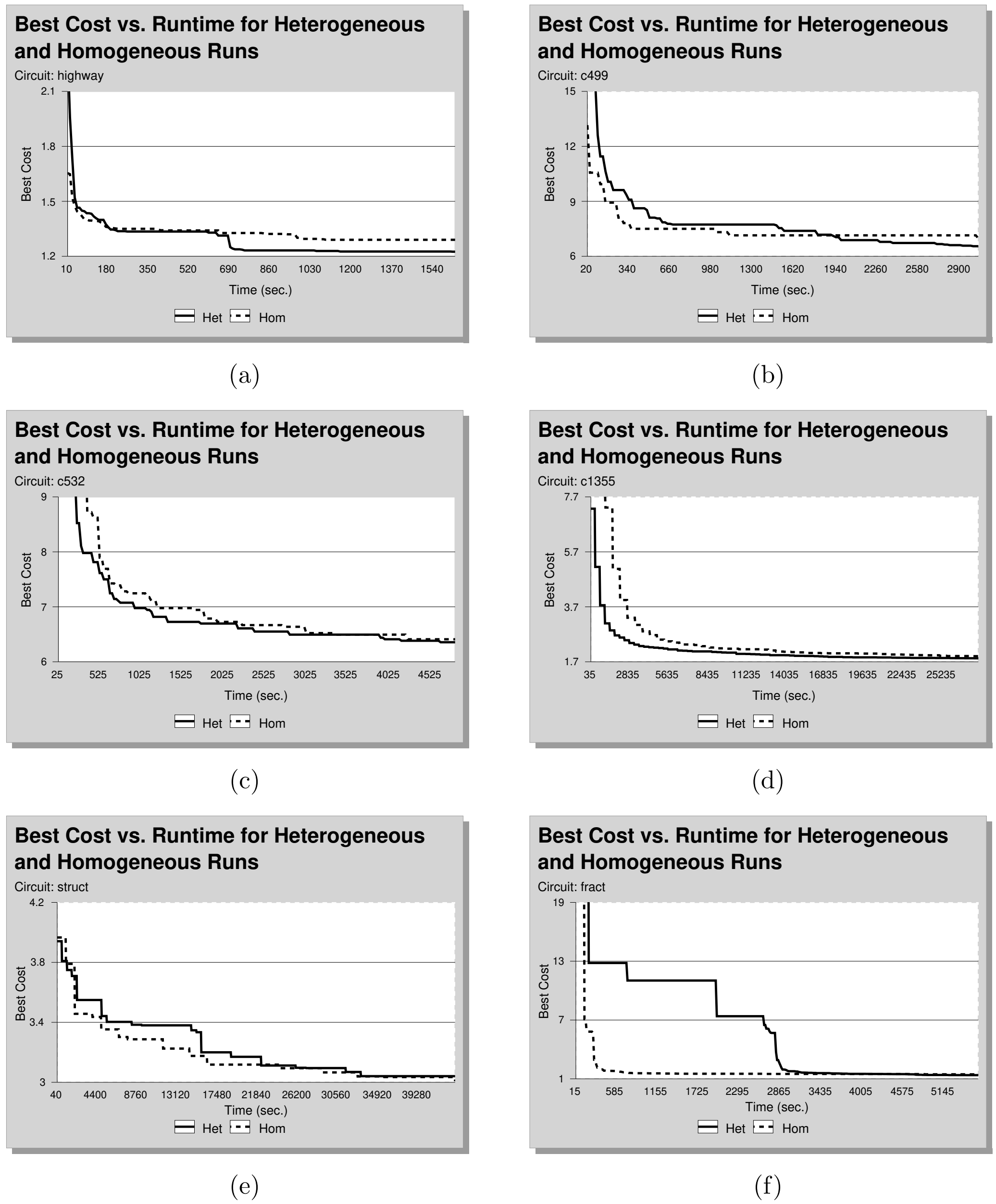

Fig. 5. Best cost versus **runtime** in Seconds for heterogeneous and homogeneous runs.

erations is maintained the same for both cases, the heterogeneous run-time is expected to be far less than the homogeneous runtime.

Figure 5 shows the best quality of solution achieved versus execution time for runs where heterogeneity is ignored (assuming a homogenous environment) and where heterogenity is accounted for. Except for one circuit (*c499*), no noticeable difference in solution quality was observed. For *c499*, there is a difference in solution quality but at the expense of larger runtime.

Table I shows the runtime needed for heterogeneous and homogeneous runs. The table clearly indicates that for all test cases the parallel implementation that accounts for heterogeneity results in a reduction of execution by a factor of 1.4 to almost 2.00 with respect to the homogeneous implementation.

Figure 5 shows that towards the end of the experiment, the heterogeneous run is doing either better than or at least as good as the homogeneous run, but never performs worse. For some circuits like *highway, c532* and *c1355*,

TABLE I
Runtime comparison of homogeneous and heterogeneous runs (in Seconds).

Circuit	Hom. Runtime	Het. Runtime	%Impt
highway	2316	1631	1.42
fract	11194	5626	1.99
c499	5722	3060	1.87
c532	8615	4839	1.78
c880	32361	19550	1.66
c1355	42560	27822	1.53
struct	76954	43332	1.78

the heterogeneous run keeps performing better than the homogeneous run throughout the execution. For *c499*, the heterogeneous run starts by performing worse and afterwards it outperforms the homogeneous run.

In another experiment, it was determined that if it is useful (or not) to include slower machines in the parallel virtual machine, because the master keeps stopping them once the other machines report their best solutions. To see the contribution of the slower machines in the proposed strategy, an experiment was conducted where one high-speed, one medium-speed, and one low-speed machine were used as a parallel virtual machine. A single TSW was spawned on each machine with one CLW per TSW. As before, once a TSW reports its best solution to the master, the master causes all others to stop and report their best solutions to it. By monitoring the number of solutions reported by each TSW within various cost ranges, which machine is contributing more to the search with useful results can be determined. Figure 6 shows the results of the experiment ran on *c499* for 500 global iterations. The results show that the contributions of the three machines are nearly equal in all cost ranges. This behavior is attributed to the non-deterministic nature of the search and to the diversification step performed by the CLWs as well as the synchronization step performed by the master process at the end of each global iteration.

VI. Conclusion

A parallel Tabu Search algorithm on hetrogeneous workstations for VLSI placement is presented in this paper. Functional decomposition and multi-search thread strategies were used for parallelization using PVM. A strategy to run the algorithm on hetrogeneous workstations was proposed and experimented on different benchmark circuits and compared with the test where hetrogeneity was not taken into account. The proposed strategy performed better in terms of run-time for producing same or better quality solutions.

Acknowledgement:

Authors thank King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, for support under project # COE/ITERATE/221.

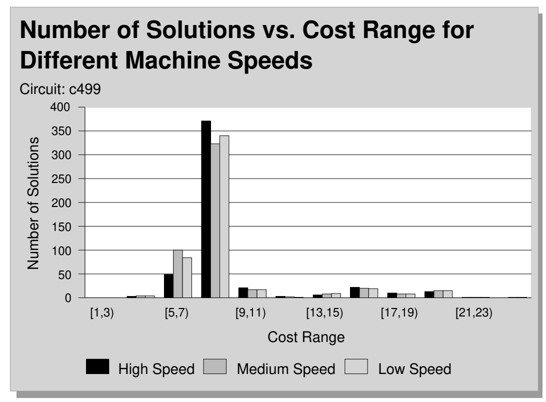

Fig. 6. Number of solutions provided by machines of different speeds within various solution ranges.

References

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

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

[3] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, Cambridge, Massachusetts, London, England, 1994.

[4] M. Lewis and R. Cline. PVM communication performance in a switched FDDI heterogeneous distributed computing environment. In *IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 13–19, October 1993.

[5] P. Crandall, E. Sumithasri, and M. Clement. Performance comparison of desktop multiprocessing and workstation cluster computing. In *5th IEEE Int'l Symposium on High Performance Distributed Computing*, pages 272–281, Aug 1996.

[6] G. Geist and V. Sunderam. The PVM system: Supercomputer level concurrent computation on a heterogeneous network of workstations. In *6th Distributed Memory Computing Conference*, pages 258–261, May 1991.

[7] 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.

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