Accelerating Multiobjective VLSI Cell Placement with Parallel Tabu Search

Sadiq M. Sait Mahmood R. Minhas
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals
Dhahran-31261, Saudi Arabia.
email: {sadiq,minhas}@ccse.kfupm.edu.sa

***Abstract*— Tabu Search has been successfully used to solve a range of hard optimization problems [1], [2]. In this paper, we present a parallelization of TS to increase efficiency of solving the constrained multiobjective VLSI standard cell placement problem. Acceleration of search to decrease runtime requirements of large circuits is investigated and other proposals in literature are experimented with. Candidate list partitioning and distribution of unique moves to slaves is employed. The proposed parallel TS is implemented on a dedicated Linux-based cluster of workstations. Results of experiments on ISCAS-85/89 benchmark circuits, illustrating quality and speedup are presented and compared with serial TS and genetic algorithm implementation.**

I. Introduction

General iterative heuristics such as tabu search are getting more widely adopted to obtain near optimal solutions to numerous hard problems [1]. For small problems, tabu search techniques have reasonable runtime requirements. For example, for a VLSI cell placement problem with few hundred modules, it is possible to find very good solutions in reasonable time. However, most practical circuits are very large and require several hours of computer time to solve by iterative heuristics [3]. One way to adapt iterative techniques to solve large problems and traverse larger search spaces in reasonable time is to resort to parallelization [4], [5]. Although there have been some efforts to parallelize TS approach for solving various optimization problems [6], but in many cases if the reported speed-up and scalability results are observed, effective parallelization of TS does not prove to be an easy task [7].

A. Related Work

In this section we briefly survey some parallelization efforts for the tabu search heuristic.

A number of parallelization techniques have been reported in literature [7]. In the most straightforward and widely adopted approach k tabu search processes are spawned and run concurrently on k processors where each processor carries out independent search [7], [8]. Malek suggested linking independent searches where each slave runs a copy of a serial TS but with different parameter settings [9]. After a specified time the slave processes are halted and the main process selects the best solution found and broadcasts to all slave processes to start the entire search again from this new solution.

Another approach to parallelize search within an iteration is when each process is given a task of exploring a subset of the neighborhood of current solution. Two approaches are followed: *synchronous* and *asynchronous*. In the synchronous approach the various processes are always working with the same solution, but exploring different partitions of the current local neighborhood. A *master* process orchestrates the activities of the *slave* processes [8]. In the asynchronous approach, all processes are peer and usually are not all working with the same current solution [7]. Both approaches require that the set of possible moves be partitioned among the available processors so that each processor will be exploring a distinct sub-region of the current solutions neighborhood.

Suggestions to increase efficiency of TS by parallelizing also include partitioning the search space, which is difficult, or partitioning the problem into smaller sub-problems, determining the best moves for each sub-problem, and then performing a compound move [7].

Attempts to solve several practical NP-hard problems has been reported in literature. For example, in [8] a parallel implementation of the tabu search algorithm for the vehicle routing problem is described. A massively parallel implementation of tabu search for the quadratic assignment problem (QAP) is reported in [10]. This parallelization strategy has been implemented on the Connection Machine CM-2, a massively parallel SIMD machine [11]. A reduction in the runtime per iteration was achieved when compared to other sequential and parallel implementations [12], [10].

Serial implementation of TS have been applied to VLSI cell placement problem [3]. The approach obtained excellent quality of placement solutions but at cost of huge runtime. In this paper we present the parallelization of TS to address the constrained multiobjective VLSI standard cell placement problem. The optimization problem, and its serial implementation was analyzed and its results used to carefully engineer the TS heuristic for increasing the efficiency of search.

This paper is organized as follows: The following section briefly discusses the optimization problem (VLSI cell placement) and the related cost functions. In Section III, the proposed parallel Tabu Search approach is presented, followed by experiments and results in Section IV.

II. Placement Optimization Problem and Cost Functions

In this section, we formulate our problem and the cost function used in our optimization process.

A. Cell Placement

In this paper we are addressing the problem of parallelizing SimE to solve the multiobjective VLSI standard cell placement. The the objectives are optimizing of power consumption, timing performance (delay), and overall wirelength while, considering layout width as a constraint. A semi-formal description of the placement problem can be found in [13].

B. Cost Functions

A semi-formal description of the placement problem can be found in [13]. The multiobjective cost function is similar to the one formulated in [3]. The first objective, wirelength cost ($Cost_{wire}$) is estimated using an approximate Steiner tree algorithm.

The power consumption cost p_i is computed for each net i. Assuming a a fix supply voltage and clock frequency, the estimate can be obtained by $p_i \simeq C_i \cdot S_i$, (where S_i is the switching probability and C_i the total capacitance, of net i). This can be further improved to $p_i \simeq l_i \cdot S_i$ (since interconnect capacitances are a function of the interconnect lengths, and the input capacitances of the gates are constant). The total estimate of the power dissipation reduces to $Cost_{power} = \sum_{i \in M} p_i = \sum_{i \in M} (l_i \cdot S_i)$.

The delay cost is taken as the delay along the longest path in a circuit. The delay T_π of a path π consisting of nets $\{v_1, v_2, ..., v_k\}$, is expressed as: $T_\pi = \sum_{i=1}^{k-1} (CD_i + ID_i)$ where CD_i is the switching delay of the cell driving net v_i and ID_i is the interconnect delay of net vi. The placement phase affects ID_i because CD_i is technology dependent parameter and is independent of placement: $Cost_{delay} = max\{T_\pi\}$.

The layout width is constrained not to exceed a certain positive ratio α to the average row width w_{avg}.

C. Fuzzification of Multiobjectives

Since, we are optimizing three objectives simultaneously, we need to have a cost function that represents the effect of all three objectives in form of a single quantity. Fuzzy logic is used to integrate these multiple, possibly conflicting objectives into a scalar cost function. Fuzzy logic allows us to describe the objectives in terms of linguistic variables. Then, fuzzy rules are used to find the overall cost of a placement solution. In this work, we have used following fuzzy rule:

IF a solution has *SMALL wirelength* **AND** *LOW power consumption* **AND** *SHORT delay* **THEN** it is an *GOOD* solution.

The above rule is translated to *and-like* OWA fuzzy operator [14] and the membership $\mu(x)$ of a solution x in fuzzy set *GOOD solution* is obtained by:

$$\mu(x) = \begin{cases} \beta \cdot \min\limits_{j=p,d,l} \{\mu_j(x)\} + (1-\beta) \cdot \frac{1}{3} \sum\limits_{j=p,d,l} \mu_j(x); \\ \qquad \text{if} \quad Width - w_{avg} \leq \alpha \cdot w_{avg}, \\ 0; \qquad \text{otherwise.} \end{cases} \tag{1}$$

Here $\mu_j(x)$ for $j = p, d, l, width$ are the membership values in the fuzzy sets *LOW power consumption*, *SHORT delay*, and *SMALL wirelength* respectively. β is the constant in the range $[0, 1]$. The solution that results in maximum value of $\mu(x)$ is reported as the best solution found by the search heuristic. The membership functions for fuzzy sets *LOW power consumption*, *SHORT delay*, and *SMALL wirelength* and the lower bounds for different objectives can be found in [3].

III. Parallel Tabu Search Implementation

A generic intuitive strategy for parallelization is to partition the data into small subsets that are distributed among the processors. Each processor is responsible for a data subset and implements a sequential version of the concerned function (or the heuristic) over this data subset.

The sequential implementation of TS was analysed using profiling tools (GNU profiler) to obtain insight into determining the time consuming operations of the code and the usage of resources. For the circuits experimented on, between 60-80% of time was spent on computation of cost of the objectives and their fuzzification. Furthermore, experiments with parameters revealed that for our hard optimization problem with conflicting multiobjectives, large sizes of candidate list (upto 120) were required to obtain high quality solutions. Since the computation of cost for all moves in the candidate list was the most time consuming operation, (in each iteration) the algorithm was designed to partition this workload.

Therefore, the parallel Tabu Search strategy adopted in this work employs dividing the operations within a TS iteration. According to taxonomy given by Crainic et. al [6], our approach can be classified as a synchronous master-slave (one master and remaining slaves), 1-control (each process is responsible for its search), Rigid Synchronous (RS) (all processes are forced to establish communication and exchange information at specific points) and Single Point Single Strategy (SPSS) (all the processes start with the same initial solution and follow the same strategy).

In this implementation, there is an initialization step during which, the master process (Figure 1 generates and sends an initial solution and a disjoint (non-overlapping) partial candidate list (PCL) to each slave process. A move in a PCL assigned to a slave in a particular iteration does not appear in PCLs assigned to other slaves. Each slave process searches its local neighborhood by trying each move in the partial candidate list on the initial solution and computes gains due to them. Then it sends the best move and its corresponding cost (or gain) to the master process. The master process selects the overall best move (OBM) among the moves it received from slave processes subject to tabu restrictions.

Algorithm MasterProcess;
Begin
(* S_0 is the initial solution. *)
(* $BestS$ is the best solution. *)
(* PCL is the Partial Candidate List. *)
(* p is the number of slave processors. *)
(* OBM is the Overall Best Move. *)
Generate S_0 and p number of $PCLs$;
Send S_0 and a PCL to each slave process;
While *iteration-count* $<$ *max-iterations*
Receive best move and cost from each slave;
Find OBM subject to tabu restrictions;
Generate P number of $PCLs$;
Send OBM and a PCL to each slave process;
Update $BestS$; /*by applying OBM on $BestS$*/;
EndWhile
Return ($BestS$)
End. /*MasterProcess*/

Fig. 1. The master process in parallel TS.

Then in each subsequent iteration, the master process sends the overall best move and a new partial candidate list to each slave process. Each slave process now starts by performing the received overall best move so that all the slave processes start their iteration from the same solution. Each slave process searches its local neighborhood and sends the best move and its cost to the master process. The pseudo code of the slave process is given in Figure 2.

IV. Experiments & Results

A. Experimental Setup

The experimental setup consists of the a homogeneous cluster of 8 machines (where 1 machine is always working as a master processor), x86 architecture, Pentium-4 of 2 GHz clock speed, and 256 MB of memory. These machines are connected by 100Mbit/s Ethernet switch. Operating system used in RedHat Linux 7.3 (kernel 2.4.7-10). The paradigm used for parallelization is MPI (Message Passing Interface). Specifically, MPICH (a portable implementation of MPI standard 1.1) is used in our implementation. In terms of GFlops measure, the maximum performance of the cluster, with NAS Parallel Benchmarks was found to be 1.6 GFlops, (using NAS's LU, Class A, for 8 processors). Using this same benchmark for a single processor, the individual performance of one machine was found out to be 0.3 GFlops. The maximum bandwidth that was achieved using PMB was 91.12 Mbits/sec, with an average latency of 68.69 μsec per message. ISCAS-85/89 circuits are used as performance benchmarks for evaluating the proposed parallel TS placement technique. These circuits are of various sizes in terms of number of cells and paths, and thus offer a variety of test cases.

B. Results, Comparison & Discussion

For comparison purposes, we also implemented a parallel genetic algorithm (GA) which is a derivative of a standard distributed GA and follows the island model, with independently evolving sub-populations and periodic exchanges of solutions through migration [15], [16]. A pseudo-diversity approach

Algorithm SlaveProcess;
Begin
Receive S_0 and a PCL from the master process;
$CurS = S_0$; (* Current Solution *)
While *iteration-count* $<$ *max-iterations*
Try each move in PCL and compute cost;
Send the best move and its cost to the master process;
Receive OBM and a PCL from the master process;
Update $CurS$ /* by applying OBM on $CurS$ */;
EndWhile
End. /*SlaveProcess*/

Fig. 2. The slave process in parallel TS.

is taken, wherein similar solutions are not permitted in the population at any time. This diversity serves to widen the search, while limiting the possibility of premature convergence in local minima solution space. The initial population is constructed at the master process and distributed among N slave processes which start running serial GA on their allocated population for a predefined number of iterations called the Migration Frequency (MF). Then each slave process sends MR (Migration Rate) number of its best solutions to the master process, which selects MR overall best solutions and broadcasts them to all slave processes. Each slave process absorbs the incoming best solutions into its population (if they are not already found) by replacing the weakest solutions. Each slave process then continues with the serial GA for another MF iterations. Standard PMX crossover is used to generate offsprings [1].

The quality of solution obtained and runtime required using different number of processors for both TS and GA are tabulated in Table I. For each circuit, the number of cells are given in the table. The '$\mu(s)$ TS' and '$\mu(s)$ GA' columns show the aggregate fuzzy membership of solution obtained by TS and by GA respectively, whereas 'p' denotes the number of processors used. It should be noted that runtimes shown are for achieving a certain fixed quality.

In case of large circuits, parallel GA was unable to find a reasonable quality solution even after running for a large amount of time. Even for smaller circuits, the solution quality obtained using TS is significantly superior to that obtained using GA, and also the speedup trend is very consistent for TS. On the other hand, parallel GA did not show such performance or trend.

The proposed parallel TS has shown a consistent trend in terms of speedup with increasing number of processors. Figure 3 shows a speedup plot for some selected large circuits, and it can be seen that almost linear speedup was obtained.

Another synchronous parallel implementation of TS in which k tabu search processes are spawned and run concurrently on k processors was also experimented with in this work [7]. All slave processes start from the same solution and perform tabu search for a predefined number of iterations and then pass their individual best solutions to the master process, which selects and broadcasts the overall best solution to the slave. . For our multiobjective optimization, in terms of

TABLE I

RUN TIMES AND SOLUTION QUALITY $\mu(s)$ FOR ACHIEVING A TARGET MEMBERSHIP FOR SERIAL AND PARALLEL TS/GA APPROACHES. X INDICATES UNREASONABLY HIGH RUNTIME REQUIREMENT.

Circuit Name	Number of Cells	$\mu(s)$ TS	Time for Serial TS	Time for Parallel TS						$\mu(s)$ GA	Time for Serial GA	Time for Parallel GA		
				p=2	p=3	p=4	p=5	p=6	p=7			p=3	p=5	p=7
s386	172	0.688	52	28	20	17	16	15	14	0.504	15	9.9	5.7	6.7
s641	433	0.785	934	472	332	239	205	171	151	0.616	793	307	390	289
s832	310	0.644	74	40	33	23	22	20	19	0.479	128	43	37	39
s953	440	0.661	195	98	71	53	46	41	36	0.511	309	136	91	108
s1196	561	0.653	374	187	132	97	88	78	67	0.484	988	327	262	205
s1488	667	0.603	259	131	93	69	63	55	49	0.482	1883	677	435	418
s1494	661	0.601	268	137	96	72	65	57	51	0.496	1405	847	638	479
c3540	1753	0.665	2142	1146	703	547	440	370	344	-	X	X	X	X
s3330	1961	0.699	1186	590	451	313	245	210	184	-	X	X	X	X
c5378	2993	0.669	1850	914	601	467	371	312	264	-	X	X	X	X
s9234	5844	0.631	5571	2855	2006	1525	1272	1062	849	-	X	X	X	X

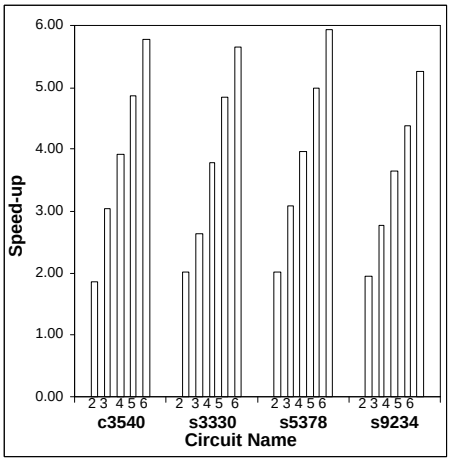

Fig. 3. Speedup obtained for selected large circuits.

speedup obtained, results obtained were not encouraging.

V. CONCLUSIONS

In this work, we presented a parallel tabu search strategy for accelerating the solution to a constrained multiobjective VLSI cell placement problem. A distributed parallel GA was also implemented for the comparison purposes. Experimental results on ISCAS-85/89 benchmarks exhibited excellent trend in terms of speedup, and it was revealed that the proposed parallelization of TS required reduced runtimes for same quality of placement solution.

VI. ACKNOWLEDGMENT

The authors would like to thank King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, for support under project code # COE/CELLPLACE/263.

REFERENCES

[1] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms and their Application to Engineering*. IEEE Computer Society Press, December 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] Sadiq M. Sait, Mahmood R. Minhas, and Junaid A. Khan. Performance and low-power driven VLSI standard cell placement using tabu search. *Proceedings of the 2002 Congress on Evolutionary Computation*, 1:372–377, May 2002.

[4] Prithviraj Banerjee. *Parallel Algorithms for VLSI Computer-Aided Design*. Prentice Hall International, 1994.

[5] Van-Dat Cung, Simone L. Martins, Celso C. Riberio, and Catherine Roucairol. Strategies for the Parallel Implementation of metaheuristics. *Essays and Surveys in Metaheuristics*, pages 263–308, Kluwer 2001.

[6] T. G. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search heuristics. *INFORMS Journal of Computing*, 9(1):61–72, 1997.

[7] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro. Improving search by incorporating evolution principles in parallel tabu search. In *Proc. of the first IEEE Conference on Evolutionary Computation-CEC'94*, volume 1, pages 823–828, June 1994.

[8] Bruno-Laurent Garica, Jean-Yves Potvin, and Jean-Marc Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033, November 1994.

[9] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. "serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem". *Annals of Ops. Res.*, 21:59–84, 1989.

[10] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 41:327–341, 1993.

[11] *Connection Machine Model CM-2, Technical Summary Version 5.1*. Thinking Machines Corporation, Cambridge, MA, May 1989.

[12] E. Taillard. Robust tabu search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[13] Sadiq M. Sait and Habib Youssef. VLSI Physical Design Automation: Theory and Practice. *World Scientific Pubishers*, 2001.

[14] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transaction on Systems, MAN, and Cybernetics*, 18(1), January 1988.

[15] M. Toulouse, T. G. Crainic, and M. Gendreau. Issues in Designing Parallel and Distributed search Algorithms for Discrete Optimization Problems. *Publication CRT-96-36, Centre de recherche sur les transports, Université de Montréal, Montréal, Canada*, 1996.

[16] Erick Cant-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallles, Reseaux et Systems Repartis*, 1998.