A Parallel Tabu Search Algorithm for Optimizing Multiobjective VLSI Placement

Mahmood R. Minhas and Sadiq M. Sait

College of Computer Sciences & Engineering,
King Fahd University of Petroleum & Minerals,
Dhahran-31261, Saudi Arabia
{minhas, sadiq}@ccse.kfupm.edu.sa

Abstract. In this paper, we present a parallel tabu search (TS) algorithm for efficient optimization of a constrained multiobjective VLSI standard cell placement problem. The primary purpose is to accelerate TS algorithm to reach near optimal placement solutions for large circuits. The proposed technique employs a candidate list partitioning strategy based on distribution of mutually disjoint set of moves among the slave processes. The implementation is carried out on a dedicated cluster of workstations. Experimental results using ISCAS-85/89 benchmark circuits illustrating quality and speedup trends are presented. A comparison of the obtained results is made with the results of a parallel genetic algorithm (GA) implementation.

1 Introduction and Related Work

General iterative heuristics such as tabu search and genetic algorithms (GAs) have been widely used to solve numerous hard problems [1]. This interest is attributed to their generality, ease of implementation, and ability to reach near optimal solutions by escaping from local minima. However, depending on size of a problem, such heuristics may have huge runtime requirements. This is also true for VLSI placement problem of modern industry-size circuits for which, iterative heuristics require huge run times to reach near optimal solutions [2, 3]. With rapidly increasing density of VLSI circuits, the run time dilemma of iterative techniques is aggravating and hence there is a need of accelerating their search process.

One way to adapt iterative techniques to large problems and to efficient traversing of large search space is their parallelization [4, 5]. An effective parallelization strategy must consider issues such as proper partitioning of the problem to facilitate uniform distribution of computationally intensive tasks. At the same time, it should be capable of thorough traversal of the complex search space. In the subsequent paragraphs, we present a brief review of some earlier efforts towards parallelization of TS.

A number of parallelization techniques have been reported in literature [6]. A taxonomy of parallel tabu search strategies was given by Crainic et. al [7]. In the most straightforward and widely adopted approach, k tabu search processes are spawned and run concurrently on k processors where each processor

© Springer-Verlag Berlin Heidelberg 2005

carries out independent search [6, 8]. Malek et al. suggested linking independent searches where each slave runs a copy of a serial TS but with different parameter settings [9]. After specified time intervals, all the slave processes are halted and the master process selects the overall best solution found so far and broadcasts it to all the slave processes. Each slave process then restarts its search process from the best solution it receives from the master.

Another proposed approach is to parallelize search process within an iteration of TS. In this approach, each process is given the task of exploring a subset of the neighborhood of the current solution. Here two approaches are followed: *synchronous* and *asynchronous*. In the synchronous approach, various processes are always working with the same solution but exploring different partitions of the neighborhood. The master process orchestrates the activities of the slave processes [8]. In the asynchronous approach, all processes are peer and are usually not working with the same current solution [6]. Both of these approaches require that the set of possible moves be partitioned among the available processors so that each processor can explore a distinct sub-region of the neighborhood.

Some other suggestions for efficient parallelization of TS include partitioning the search space, or partitioning the problem into smaller sub-problems with determining the best moves for each sub-problem, and then performing a compound move [6].

Attempts to solve some hard optimization problems have also been reported in literature. Parallel tabu search algorithms for the vehicle routing problem are presented in [8, 10]. A massively parallel implementation of tabu search for the Quadratic Assignment Problem (QAP) is reported in [11]. The parallel algorithm was implemented on a Connection Machine CM-2 (a massively parallel SIMD machine). A reduction in runtime per iteration was reported when compared to some other sequential and parallel implementations [11, 12].

In this paper, we present a parallel TS algorithm for efficient optimization of a hard optimization problem namely, constrained multiobjective VLSI standard cell placement. The rest of the paper is organized as follows: The next section briefly discusses the placement problem and the related cost functions. In Section 3, some implementation details of the proposed parallel tabu search algorithm are presented, followed by experimental results and comparisons in Section 4. Section 5 presents some concluding remarks.

2 The Placement Problem and Cost Functions

We address a multiobjective VLSI standard cell placement problem where the objectives are optimization of and total wirelength, power consumption, and timing performance (delay), whereas layout width is considered as a constraint. Cell placement is one of the intermediate steps in VLSI physical design and is a generalization of QAP.

The VLSI cell placement problem can be stated as follows [13]: Given a collection of cells or modules with ports (inputs, outputs, power and ground pins) on the boundaries, the dimensions of these cells (height, width, etc), and a collection

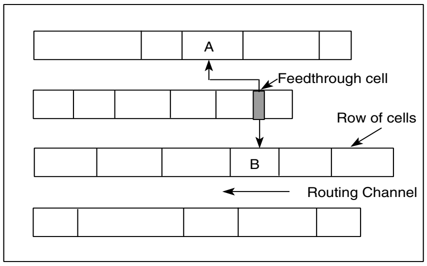


Fig. 1. A Typical Standard Cell Layout

of nets (which are sets of ports that are to be wired together), *placement* problem consists of finding suitable physical locations for each cell on the layout. By suitable we mean those locations that minimize given objective functions, subject to some constraints imposed by the designer, the implementation process, or layout strategy and the design style.

In this work, we deal with standard cell placement, where all the circuit cells are constrained to have the same height, while the width of the cell is variable and depends upon its complexity. A typical standard cell layout is shown in Figure 1. As can be seen in the figure, cells are arranged in rows with routing channels between the rows. Due to varying width of cells, row widths may be unequal depending on the type and number of cells placed in a row. An approximation would be to treat cells as points, but in order to have a more accurate estimate of wirelength, widths of cells are taken into account. Heights of routing channels are estimated using the vertical constraint graphs constructed during the channel routing phase. With this information, a fairly accurate estimate of power dissipation, delay and total wirelength can be obtained [13]. Next, we formulate the cost function used in our optimization process.

2.1 Cost Functions

Now we formulate cost functions for our three objectives and for the width constraint.

Wirelength Cost: Interconnect wirelength of each net in the circuit is estimated using steiner tree approximation. Total wirelength is computed by adding all these individual estimates:

$$Cost_{wire} = \sum_{i \in M} l_i \tag{1}$$

where l_i is the wirelength estimation for net i and M denotes total number of nets in circuit.

Power Cost: Power consumption p_i of a net i in a circuit can be given as:

$$p_i \simeq C_i \cdot S_i \tag{2}$$

where C_i is total capacitance of net i, and S_i is the switching probability of net i. Here, C_i depends on wirelength of net i, hence Equation 2 can be written as:

$$p_i \simeq l_i \cdot S_i \tag{3}$$

The cost function for total power consumption in the circuit can be given as:

$$Cost_{power} = \sum_{i \in M} p_i = \sum_{i \in M} (l_i \cdot S_i) \tag{4}$$

Delay Cost: The delay cost is determined by computing delay along the longest path in a circuit. The delay of any given path is computed as summation of the delays of all the nets belonging to the path and the switching delays of the cells driving these nets. Hence, the delay T_π of a path π consisting of nets $\{1, 2, ..., k\}$ is expressed as:

$$T_\pi = \sum_{i=1}^{k-1} (CD_i + ID_i) \tag{5}$$

where CD_i is the switching delay of the cell driving net i and ID_i is the interconnect delay of net i. The placement phase affects ID_i only because CD_i is technology dependent and is hence independent of the placement process. The cost function for delay of the circuit can be given as:

$$Cost_{delay} = max\{T_\pi\} \tag{6}$$

Width Cost: Width cost is given by the maximum of all the row widths in the layout. We have constrained layout width not to exceed a certain positive ratio α to the average row width w_{avg}, where w_{avg} is the minimum possible layout width obtained by dividing the total width of all the cells in the layout by the number of rows in the layout. We can express width constraint as below:

$$Width - w_{avg} \leq \alpha \times w_{avg} \tag{7}$$

2.2 Fuzzification of Multiobjectives

Since we are targeting to optimize three (possibly conflicting) objectives simultaneously, we need to formulate an aggregating cost function which can expresses the costs of the objectives in form of a single quantity. We resorted to using fuzzy in designing our aggregating cost function. Fuzzy logic allows to describe the objectives in terms of linguistic variables following which, the membership functions can be defined. The membership functions for *SMALL wirelength*, *LOW power consumption*, and *SHORT delay* are same as described in [3]. Finally, fuzzy rules are used to design the aggregate cost function. In this work, we have used the following fuzzy rule:

Rule 1: 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:

$$\mu(x) = \begin{cases} \beta \cdot \min_{j=l,p,d} \{\mu_j(x)\} + (1-\beta) \cdot \frac{1}{3} \sum_{j=l,p,d} \mu_j(x); \\ \qquad\qquad \text{if } \ Width - w_{avg} \leq \alpha \cdot w_{avg}, \\ 0; \qquad\quad \text{otherwise.} \end{cases} \tag{8}$$

Here $\mu_j(x)$ for $j = \{l, p, d\}$ are the membership values in the fuzzy sets *SMALL wirelength*, *LOW power consumption*, and *SHORT delay* respectively, whereas β is a constant in range $[0, 1]$. A placement solution that results in a higher value of $\mu(x)$ is considered a better solution.

3 Parallel Tabu Search Algorithm for Optimizing VLSI Placement

The parallel tabu search strategy adopted in this work was engineered after a careful performance analysis of our sequential TS implementation [3]. The analysis was performed using some profiling tools (like GNU profiler) to obtain insight into determining the time consuming operations of the code and the usage of resources. For the circuits experimented on, it was found that almost 60% to 80% of the total run time was spent on cost computation of the three 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 cost computation for all moves in the candidate list was the most time consuming operation, the proposed algorithm was designed with a view of partitioning this compute-intensive task. The proposed algorithm employs a candidate list partitioning strategy based on distribution of mutually disjoint set of moves among the slave processes. The pseudo code of the master and the slave processes in the proposed parallel TS are shown in Figures 2 and 3 respectively.

According to taxonomy given by Crainic et. al [7], 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 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

```
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. 2. The master process in parallel TS

```
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. 3. The slave process in parallel TS

master process. The master process selects the overall best move (OBM) among the moves it received from slave processes subject to tabu restrictions. 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.

4 Experimental Results and Discussions

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.

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 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].

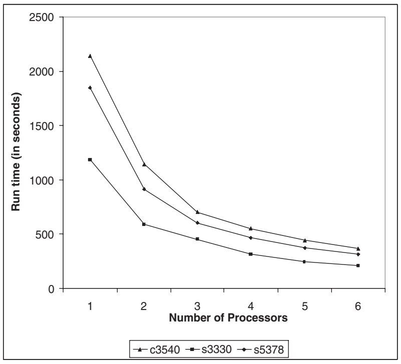
The quality of solution obtained and run time required using different number of processors for both TS and GA are tabulated in Table 1. For each circuit, the number of cells are given in the table. The '$\mu(s)$ TS' and '$\mu(s)$ GA' columns show the aggregate fuzzy membership of solution obtained by TS and by GA respectively, whereas 'p' denotes the number of processors used. It should be noted that run times 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.

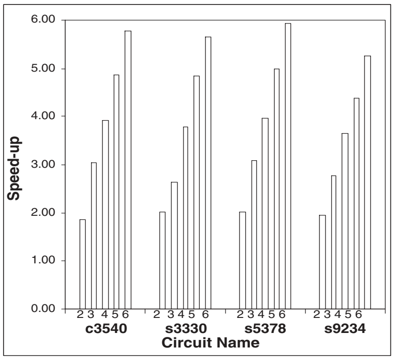
The proposed parallel TS has shown a consistent trend in terms of speedup with increasing number of processors. Figure 4 shows a run-time as well as a speedup plot of parallel TS for some selected large circuits, and demonstrates an almost linear speedup. As can be seen, there is a consistent decreasing trend in run time. This trend is more pronounced for medium to large circuits than for smaller ones and reveals the good scalability of the proposed approach.

Table 1. Run times and solution quality $\mu(s)$ for achieving a target membership for serial and parallel TS/GA approaches. X indicates unreasonably high run time requirement

Circuit Name	Number of Cells	$\mu(s)$ TS	Time for Serial TS	Time for Parallel TS p=2	p=3	p=4	p=5	p=6	p=7	$\mu(s)$ GA	Time for Serial GA	Time for Parallel GA 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



(a)



(b)

Fig. 4. (a) Decrease in run times for selected large circuits with increasing number of processors for parallel TS. (b) Speedup obtained for selected large circuits for parallel TS

5 Conclusions

In this work, we presented a parallel tabu search strategy for accelerating the solution to a constrained multiobjective VLSI cell placement problem. The proposed strategy belongs to p-control, RS, MPSS class and was implemented on a dedicated cluster of workstations. A distributed parallel GA was also implemented for the comparison purposes. Experimental results on ISCAS-85/89 benchmarks exhibit that the proposed parallel TS shows an excellent trend in terms of speedup and requires far lesser run times as compared to serial TS for obtaining the same quality of placement solutions. When compared with results obtained by parallel GA, the proposed parallel TS clearly outperforms both in terms of solution quality as well as run time.

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. Sadiq M. Sait, Habib Youssef, Aiman El-Maleh, and Mahmood R. Minhas. Iterative Heuristics for Multiobjective VLSI Standard Cell Placement. *Proceedings of IJCNN'01, International Joint Conference on Neural Networks*, 3:2224–2229, July 2001.
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. 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.
7. 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.
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. E. Taillard. Parallel Iterative Search Methods for the Vehicle Routing Problem. *Networks*, 23:661–673, 1993.
11. J. Chakrapani and J. Skorin-Kapov. Massively Parallel Tabu Search for the Quadratic Assignment Problem. *Annals of Operations Research*, 41:327–341, 1993.
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, Singapore, 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.