Multiobjective VLSI Cell Placement Using Distributed Genetic Algorithm

ABSTRACT

Genetic Algorithms have worked fairly well for the VLSI cell placement problem, albeit with significant run times. This is all the more true for multiobjective VLSI cell placement, where the need to optimize conflicting objectives adds another level of complexity. A Master-Slave parallel strategy for GA is presented for VLSI cell placement where the objectives are optimizing power dissipation, timing performance and interconnect wirelength, while layout width is a constraint. Fuzzy rules are incorporated in order to design a cost function that integrates these objectives into a single overall value. Also, a Multi-Deme parallel GA, in which each processor works independently on an allocated subpopulation followed by information exchange through migration of chromosomes, is applied to this multiobjective problem. A pseudo-diversity approach is taken, wherein similar solutions with the same overall cost values are not permitted in the population at any given time. A series of experiments are performed on ISCAS-85/89 benchmarks to show the comparison of these two approaches with respect to solution quality and speedup.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; J.6 [**Computer Aided Engineering**]: [Computer-aided design]

General Terms

Algorithm, Performance

Keywords

Parallel Genetic Algorithms, Cluster Computing, Fuzzy Logic, Genetic Crossover.

1. INTRODUCTION

The cell placement phase is an intermediate step in VLSI circuit fabrication and involves designing and optimizing the circuit layout by positioning cells within a constrained area. Genetic Algorithms have been extensively used for solving NP-hard problems such as VLSI cell placement. [9], [10]. Parallelization of these algorithms has increasingly become an attractive option for accelerating performance, especially due to the consistent growth in performance to cost ratios of cluster computing environments that can be achieved with today's generic high-end PCs and networks. Two parallel GA strategies are described here that target the optimization of width-constrained, multi-objective placement.

2. PARALLELIZATION OF GENETIC ALGORITHMS

Prior to developing parallelization strategies, a profiling analysis of the serial GA was carried out to determine computation intensive functions and routines. It was seen that the runtime intensive operations are fitness calculation and the parent crossover. The first parallel model derived was a 'Global Selection' approach where the population is divided among slave processors, which then carry out individual crossover followed by fitness evaluation of generated offsprings. Selection of the new population is done at the Master, which collects the cumulative offsprings from the slave processors. The performance results of this approach were poor with very low speedup for small circuits. However, with increasing circuit size, which translates into higher complexity and larger search space, there is more potential with distributing the fitness calculation and the crossover. In the case of smaller circuits, any gains achieved by such a distribution would be lost due to communication overheads.

The second approach - the Multi-Deme parallel GA, has often been favored over simplistic data distribution as in the earlier model. In this strategy, the population is distributed among all processors, which independently run their own GA for a predefined number of generations. An extensive study of the parameters governing the performance of this model was done by Cantú-Paz [5]. The pseudo-code of the algorithm is presented in Figure 1.

The initial population constructor on the master (root) processor creates the initial population which is then distributed to all non-root processors. Following this, all nodes, including the root execute the serial GA on their allocated population for a predefined number of iterations called the Migration Frequency (MF). Then each node sends a certain number of its best solutions to the root. The number of solutions sent is controlled by the Migration Rate (MR) parameter. The root determines the MR best solutions from

the collective $MR * (N)$ solutions and broadcasts it to all processors. These migrants if not already present on the processors, are then absorbed into the existing population by weeding out and replacing the weakest solutions. Each processor then continues with the serial GA for another MF number of generations. Every interval between migrations, i.e., the length of time defined by MF number of generations is called as *Epoch*. The stopping criteria is a predefined number of Epochs.

It is important to note that the migrant absorption policy dictates the replacement of worst solutions with incoming migrants only if the migrants already do not exist within the population. Also, logically this model could represent a fully connected topology of non-hierarchical processing elements which cooperate to determine the best MR solutions among themselves and absorb these into their existing populations.

3. RESULTS AND DISCUSSION

The parallel architecture used in this work is a dedicated eight-node cluster connected via a low-latency network. Each of these nodes is a general purpose stand-alone Pentium4 workstation running at 2.0GHz with 256MB memory and running the RedHat Linux distribution. The cluster runs over a Fast-Ethernet switch. Communication between nodes is achieved using the MPICH implementation of the Message Passing Interface.

Results for the Multi-Deme Parallel GA are documented in Table 1. The Migration Frequency and Migration Rate are twenty and one respectively, i.e., all processors run the GA on their allocated sub-population for twenty generations, followed by migration of one chromosome between them. The GA parameters are the same as used for the serial Genetic Algorithm.

An interesting pattern seen from the above results is the lack of effect of the 'population size per processor' on solution quality. As the population is further distributed on each processor, each node works with a smaller number of solutions - this should normally lead to a deterioration in the solution qualities. However, the new migration parameter and the migrant absorption policy mitigate the effect of the truncated population. Regarding parallel performance of this Multi-Deme approach, the speedup, which is defined as:

$$Speedup = \frac{Runtime\ on\ a\ Single\ Processor}{Runtime\ on\ Multiple\ Processors}$$

can be seen in Figure 2. It is consistently increasing across different circuits, which hints towards an independence between the scalability of this model and the size of the search space.

4. CONCLUSION

This paper primarily serves as a demonstration of documented GA parallelization strategies to multiobjective optimization problems. The first approach was a variation of the canonical Master-Slave parallel GA, with both fitness and crossover distributed among processors. Only Selection was

ALGORITHM $Multi-Deme\_Parallel\_GA$
$NOTATION$
$RANK : ROOT$= Root Processor designated by Rank=0
$RANK : NON-ROOT$= All other Processors designated by rank>0
$RANK : ANY$= All processors, including Root
MF= Migration Frequency
MR= Migration Rate
N= Number of Processors
$Epoch$ = Instances of Migration
$EPOCH\_MAX$ = Maximum Number of Migrations Stopping Criteria
Begin
($Multi-Deme\_Parallel\_GA$)

FOR RANK:ROOT
Initial Population Constructor
Distribute Initial Population
ENDFOR RANK:ROOT

FOR RANK:ANY
Receive Allocated Population
ENDFOR RANK:ANY

LOOP-A
FOR RANK:ANY
LOOP-B
Serial GA on Allocated Population:
Choice of Parents
Crossover and Offspring Generation
Fitness Calculation
New Population Selection
END LOOP-B IF [Num_Iterations >= MF]
Send MR Best Solutions and Costs to ROOT
ENDFOR RANK:ANY

FOR RANK:ROOT
Collect the best MR*N solutions
Determine best MR distinct solutions
Broadcast MR solutions
ENDFOR RANK:0

FOR RANK:ANY
Receive MR Best Solutions
IF [Received Migrants not present in existing Population]
Replace Worst Solutions with Received Solutions
ENDIF
ENDFOR RANK:ANY

END LOOP-A IF [Epoch >= EPOCH_MAX]

FOR RANK:0
Return Best solution.
ENDFOR RANK:0

End ($Multi-Deme\_Parallel\_GA$)

Figure 1: Structure of the Multi-Deme Parallel GA.

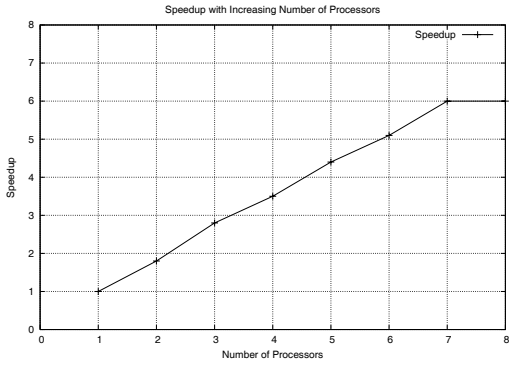

Figure 2: Speedup for circuit s386. The speedup pattern is almost identical for all circuits

Table 1: Multi-Deme Parallel GA: Variation in runtime taken to reach a target fitness with increasing number of processors.

Circuit	Target Fitness	Time taken to reach target fitness							
		P=1	P=2	P=3	P=4	P=5	P=6	P=7	P=8
s298	0.73	219	116	79	62	48	42	37	36
s386	0.63	314	171	109	89	71	62	52	52
s832	0.54	569	306	199	155	122	105	89	88
s953	0.54	1004	549	354	280	222	191	162	162
s641	0.64	2734	1439	933	730	589	520	425	424
s1196	0.54	1538	876	549	439	348	299	247	248
s1494	0.53	1679	942	597	460	367	319	263	268
s1488	0.54	1672	913	592	459	368	316	266	268
s3330	0.50	6818	3959	2584	1933	1523	1317	1090	1094

implemented by the Master. Performance gains in terms of reduced run-time were seen only for larger circuits. On the other hand, the Multi-Deme approach reported consistent performance gains independent of problem complexity and size of the search space.

Acknowledgment:

The authors thank King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia, for support under Project Code COE/CellPlacement/263.

5. REFERENCES

[1] N. Adachi and Y. Yoshida. Accelerating genetic algorithms: protected chromosomes and parallel processing, 1995.

[2] P. Adamidis. Review of genetic algorithms bibliography. *Technical Report, Aristotle University of Thessaloniki, Greece*, 1994.

[3] M. Arakawa and I. Hagiwara. Development of revised adaptive real range genetic algorithms, 1997.

[4] P. Banerjee and M. Jones. A parallel simulated annealing algorithm for standard-cell placement on a hypercube computer. *Proceedings of International Conference on Computer-Aided Design, ICCAD-86*, 1986.

[5] E. Cantú-Paz. Designing efficient master-slave parallel genetic algorithms. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 1998.

[6] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallles, Reseaux et Systems Repartis*, 1998.

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

[8] A. D. Bethke. Comparison of genetic algorithms and gradient-based optimizers on parallel processors. *Technical Report 197, University of Michigan, Ann Arbor*, 1976.

[9] H. Esbensen. A genetic algorithm for macro cell placement. *Proceedings of the 7th International Conference on VLSI Design*, pages 52–57, 1992.

[10] H.Chan, P. Mazumdar, and K. Shahookar. Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome. *Integration, the VLSI Journal*, 12:49–77, 1991.

[11] G. J. Parallel adaptive algorithms for function optimization. *Technical Report No. CS-81-19, Vanderbilt University, Tenessee*, 1981.

[12] S. M. Sait and H. Youssef. VLSI Physical Design Automation: Theory and Practice. *World Scientific Pubishers*, 2001.

[13] S. M. Sait and H. Youssef. Iterative computer algorithms and their application to engineering: Solving combinatorial optimization problems. December 1999.

[14] S. M. Sait, H. Youssef, A. El-Maleh, and M. 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.

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