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.

Keywords

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

1. INTRODUCTION

As VLSI (Very Large Scale Integration) technologies continue to proceed towards further submicron-scale circuit fabrication, the issues of design and optimization have become all the more demanding. The cell placement phase which is one of the intermediate steps in the physical design stage of these circuits, involves designing and optimizing the circuit layout by positioning cells within a constrained area. Given the inherently NP-hard complexity of this design stage, conventional constructive techniques have often proved inadequate. Genetic Algorithms on the other hand have been quite effective in reaching satisfactory layout designs, albeit with long run-times [9], [10], [14]. As such, various search acceleration strategies have been applied to Genetic Algorithms such as modifying crossover and mutation operators [1], [3]. However, Genetic Algorithms, due to their working with a population of solutions lend themselves naturally to parallelization. Such distributed implementations are further attractive due to the advances in cluster computing, especially the use of generic networked computers and the increasingly reduced cost to performance ratios.

1.1 Related Work

A generic intuitive strategy for achieving parallelization is to partition the data into small subsets distributed among the processors [4], [7], [13]. Each processor is responsible for a data subset and implements a sequential version of the concerned heuristic over this data subset. This model maps into the Genetic Algorithm structure efficiently, as GAs work with a population of independent solutions.

The earliest study that discussed the parallel implementation of Genetic Algorithms was by Bethke [8]. He analyzed the efficiency of global parallel genetic algorithm implemented over a SPMD machine and identified some bottlenecks that limit their parallel efficiency. A more thorough treatment of the subject was attempted by Grefenstette, where he mapped genetic algorithms to existing parallel architectures [11]. He proposed four prototypes, three of which were variants of a master-slave model, where a single population is maintained on the master, while the slaves are responsible for evaluating and returning the fitness. The fourth was a multiple population scheme, where the complete set of solutions is divided into multiple subpopulations.

Contemporarily, parallel Genetic Algorithms (ParGA) implemented over distributed parallel computing environments can be classified into two main categories [2], [6]:

1. Global single-population master-slave GAs
2. Multiple-population coarse-grained GAs

The former represents a master-slave paradigm, wherein the master maintains the population while the slaves are responsible for application of genetic crossovers and/or fitness calcuations. This approach is straightforward in that the evolutionary behavior can be maintained identical to the serial GA. The master generates the initial population, chooses

mating-pairs and implements the crossover for offspring generation. These offsprings are then distributed among several slave processors. The calculated fitness is aggregated back at the Master, which then selects the new population for the next generation. The speedup achievable in this model and the optimal number of slave processors can be predicted to a fairly accurate estimate [5], [6].

The second parallelization approach - the multiple-population GAs - provide a more sophisticated parallelization strategy wherein several subpopulations evolve independently on individual processors and exchange individuals periodically. This exchange of solutions is called *migration* and is a core aspect of this parallel model. Multi-population GAs are known with different names. They are referred to as Multi-Deme parallel GAs (drawing on the analogy of natural evolution), Distributed GAs (as they are often implemented on distributed parallel architectures), and Coarse-grained GAs (since the computation to communication ratio is usually high). This model of parallel GAs is very popular, but also the most difficult to understand due to the effect of migration and the new influential parameters it introduces.

Two parallel GA strategies described in this paper target the optimization of width-constrained, multiobjective placement. The first model is a derivative of the standard Master-Slave approach involving distribution of crossover in addition to fitness calculation. The second model demonstrates the application of Multi-Deme parallel GAs to this multiobjective optimization problem.

The serial GA which provides the baseline for comparison follows an aggressive pseudo-diversity approach, whereby similar solutions are not allowed in the population. However, instead of comparing the exact solution strings, which would be a runtime-expensive endeavor, the fuzzy fitness is used as the distinguishing attribute, i.e., no two solutions in the population are allowed to have the same fitness values. This approach, which is applied to both the serial GA, and the two parallelization strategies, serves to widen the search, while limiting the possibility of premature convergence of the search process in local minima solution space.

The rest of the paper is organized as follows: The cost functions for the objectives - minimizing wirelength, delay and power dissipation, and the application of fuzzy logic to aggregate these objectives is documented in the following section. The Global Selection and the Multi-Deme parallel GAs are described in Sections 3 and 4. This is followed by results and discussion in Section 5. Section 6 concludes the paper.

2. PROBLEM AND COST FUNCTION MODELING

In this section, we document the problem and the cost functions used in the optimization process.

2.1 Problem Formulation

We are addressing the problem of VLSI standard cell placement with the objectives of optimizing power consumption, timing performance (delay), and wirelength while considering layout width as a constraint. Semi-formally, the problem can be stated as follows:

A set of cells or modules $M = \{m_1, m_2, ..., m_n\}$ and a set of signals $S = \{s_1, s_2, ..., s_k\}$ is given. Moreover, a set of signals S_{m_i}, where $S_{m_i} \subseteq S$, is associated with each module $m_i \in M$. Similarly, a set of modules M_{s_j}, where $M_{s_j} = \{m_i | s_j \in S_{m_i}\}$ is called a signal net, is associated with each signal $s_j \in S$. Also, a set of locations $L = \{L_1, L_2, ..., L_p\}$, where $p \geq n$ is given. The problem is to assign each $m_i \in M$ to a unique location L_j, such that all of our objectives are optimized subject to our constraints [12].

2.2 Cost Functions

Now we formulate cost functions for our three above-mentioned objectives.

Wirelength Cost:

Interconnect wirelength of each net in the circuit is estimated and then total wire length is computed by adding the 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:

$$p_i \simeq \frac{1}{2} \cdot C_i \cdot V_{DD}^2 \cdot f \cdot S_i \cdot \alpha \tag{2}$$

where C_i is total capacitance of net i, V_{DD} is the supply voltage, f is the clock frequency, S_i is the switching probability of net i, and α is a technology dependent constant. Assuming a fix supply voltage and clock frequency, the above equation reduces to the following:

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

The capacitance C_i of cell i is given as:

$$C_i = C_i^r + \sum_{j \in M_i} C_j^g \tag{4}$$

where C_j^g is the input capacitance of gate j and C_i^r is the interconnect capacitance at the output node of cell i. At the placement phase, only the interconnect capacitance C_i^r can be manipulated while C_j^g comes from the properties of the cell from the library used and is thus independent of placement. Moreover, C_i^r depends on wirelength of net i, so Equation 3 can be written as:

$$p_i \simeq l_i \cdot S_i \tag{5}$$

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

$$Cost_{power} = \sum_{i \in M} p_i = \sum_{i \in M} (l_i \cdot S_i) \tag{6}$$

Delay Cost:

This cost is determined by 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:

$$T_\pi = \sum_{i=1}^{k-1} (CD_i + ID_i) \tag{7}$$

where CD_i is the switching delay of the cell driving net vi 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. The delay cost can be estimated as:

$$Cost_{delay} = max\{T_\pi\} \tag{8}$$

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. Formally, we can express width constraint as below:

$$Width - w_{avg} \leq \alpha \times w_{avg} \tag{9}$$

Overall Fuzzy Cost Function:

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. We propose the use of fuzzy logic 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.

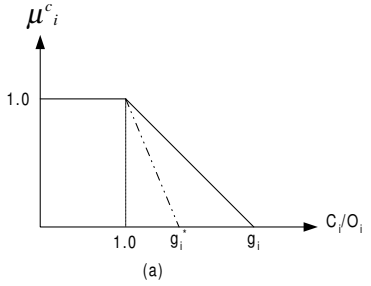

Figure 1: Membership functions.

The above rule is translated to *and-like* OWA fuzzy operator [15] and the membership $\mu(x)$ of a solution x in fuzzy set *GOOD solution* is given as:

$$\mu(x) = \begin{cases} \beta \cdot \min\limits_{j=p,d,l} \{\mu_j(x)\} + (1-\beta) \cdot \frac{1}{3} \sum\limits_{j=p,d,l} \mu_j(x); \\ \qquad \text{if } \ Width - w_{avg} \leq \alpha \cdot w_{avg}, \\ 0; \qquad \text{otherwise.} \end{cases} \tag{10}$$

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* are shown in Figure 1. We can vary the preference of an objective j in overall membership function by changing the value of g_j . The lower bounds O_j for different objectives are computed as given in Equations 11-14:

$$O_l = \sum_{i=1}^{n} l_i^* \quad \forall v_i \ \in \ \{v_1, v_2, ..., v_n\} \tag{11}$$

$$O_p = \sum_{i=1}^{n} S_i l_i^* \quad \forall v_i \ \in \ \{v_1, v_2, ..., v_n\} \tag{12}$$

$$O_d = \sum_{j=1}^{k} CD_j + ID_j^* \qquad \forall v_j \ \in \ \{v_1, v_2, ..., v_k\} \ in \ path \ \pi_c \tag{13}$$

$$O_{width} = \frac{\sum_{i=1}^{n} Width_i}{\#\ of\ rows\ in\ layout} \tag{14}$$

where O_j for $j \in \{l, p, d, width\}$ are the optimal cost estimates for wirelength, power, delay and layout width respectively, n is the number of nets in layout, l_i^* is the optimal wirelength of net v_i, CD_i is the switching delay of the cell i driving net v_i, ID_i is the optimal interconnect delay of net v_i calculated with the help of l_i, S_i is the switching probability of net v_i, π_c is the most critical path with respect to optimal interconnect delays, k is the number of nets in π_c and $Width_i$ is the width of the individual cell driving net v_i.

3. GLOBAL SELECTION PARALLEL GENETIC ALGORITHM

The serial Genetic Algorithm used as the basis for speed-up comparison is a variant of the canonical GA, in that an aggressive diversity approach is adopted to ensure that no two solutions within the population and generated offsprings are similar. However, instead of comparing actual solution strings and thus incurring significant computational overhead, the fuzzy fitness value is adopted as the distinguishing attribute. This approach serves to widen the search, while limiting the possibility of premature convergence of the search process in local minima solution space. The encoding of the solution and the application of genetic operators such as parent choice, crossover, mutation, and selection are taken from [14].

Before approaching plausible strategies of parallelization, a profiling of the serial GA is required to determine computation intensive functions and routines. Using gprof, Table 1 shows the percentile distribution of runtime between the components of the genetic algorithm. As seen in the table, the expensive operations are the fitness calculation as well as the crossover. However, with increasing circuit size and complexity, crossover starts consuming larger chunks of time.

Given the above profile, it can be seen that the canonical model of Global Parallel GA, wherein only the fitness is distributed among processors will fail here. This model, which has been widely quoted in literature, assumes that application of genetic operators is trivial, with most of the

Table 1: Percentage Time Spent in Genetic Operators versus Fitness Calculation.

Circuit	Cells	Rows	Fitness (%)	Crossover (%)	Selection (%)
s298	136	5	49.0	44.1	1.4
s386	172	5	53.4	39.3	1.3
s832	310	7	47.7	45.2	1.7
s1196	561	9	57.5	36.9	1.0
s1494	661	11	43.8	51.6	0.4
s1488	667	11	44.2	51.6	0.6
s3330	1961	17	12.3	75.4	0.7

ALGORITHM *Global_Selection_Parallel_GA*
NOTATION
$RANK : ROOT$= Root Processor designated by Rank=0
$RANK : NON-ROOT$= All other Processors designated by rank>0
$RANK : ALL$= All processors, including Root
N= Number of Processors
POP_SIZE = Population Size, same as total number of offsprings
$PROC\_NUM\_OFFSPRING$ = Number of offsprings generated by each processor
$NUM\_GENERATIONS$ = Number of generations to run the GA

Begin (*Global_Selection_Parallel_GA*)

FOR RANK:ROOT
Initial Population Constructor
Compute Fitness of Population
ENDFOR RANK:ROOT

LOOP-A

FOR RANK: ROOT
Broadcast Population
Generate list of parents pairs based on roulette choice
Broadcast list of parent pairs
ENDFOR RANK: ROOT

FOR RANK: NON-ROOT
Receive Population
Receive list of parent pairs
ENDFOR RANK: NON-ROOT

LOOP-B

FOR RANK: ALL
Read Parents from the received Parent Pair list
Crossover between Parents and Offspring Generation
Avoid Duplicates in generated Offspring
Fitness Calculation of offspring
ENDFOR RANK: ALL

END LOOP-B IF [Offsprings >= PROC_NUM_OFFSPRING]

FOR RANK: ALL
Gather offsprings at ROOT Processor
Gather offspring fitness at ROOT Processor
ENDFOR RANK: ALL

FOR RANK: ROOT
New Population Selection based on roulette selection
ENDFOR RANK: ROOT

END LOOP-A IF [Iterations >= NUM_GENERATIONS]

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

End (*Global_Selection_Parallel_GA*)

Figure 2: Structure of the Global Selection Parallel GA.

time spent in fitness calculation [2],[6]. This model, which is the simplest strategy follows a behavior pattern identical to that of serial GA, and thus can be readily analyzed using convergence studies applied to serial GA.

However, based on the above profiling analysis, a more appropriate approach would be to distribute both crossover (offspring generation) as well as fitness calculation. This would be best achieved by dividing the population among slave processors, where each would carry out individual crossover, followed by fitness computation of the resulting offspring. However, selection of the new population would be completed on the Master, which would be collect the cumulative offsprings from the slave processors. This step maintains an evolutionary pattern similar to the Serial GA, while at the same time, achieving speedups. The schematic of this approach is illustrated in Figure 2.

4. MULTI-DEME PARALLEL GENETIC ALGORITHM

The Multi-Deme model for 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 3.

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.

5. RESULTS AND DISCUSSION

5.1 Experimental Setup

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.

In terms of GFlops measure, the maximum performance of the cluster, with NAS Parallel Benchmarks is 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 achieved using PMB was 91.12 Mbits/sec, with an average latency of 68.69 μsec per message. ISCAS-89 circuits are used as performance benchmarks for evaluating the proposed parallel GA placement technique. These circuits are of various sizes in terms of number of cells and paths, and thus offer a variety of test cases.

The profiling and performance tools used in the program development consisted of standard GNU applications available natively on Linux such as the ubiquitous gdb, gprof, vmstat, as well as MPI-specific software such as Upshot and Vampir/VampirTrace for measuring program performance and behavior.

5.2 Global Selection Parallel Genetic Algorithm

Results for our Global Selection Parallel GA are tabulated in Table 2. Though the performance gain is almost null for the smallest circuit 's298', larger circuits show better speedup values with increasing number of processors. 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 are lost due to communication overheads.

5.3 Multi-Deme Parallel Genetic Algorithm

Results for the Multi-Deme Parallel GA are documented in Table 3. The Migration Frequency and Migration Rate are twenty and one respectively, i.e., all processors run the GA on their allocated sub-population for 20 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 population size on solution quality. As the population is further distributed on each processor, the quality of solution should normally deteriorate. However, the new migration parameter and the migrant absorption policy mitigate the effect of the truncated population. Regarding parallel performance, the scale-up is consistent across differing circuits, which hints towards an independence between the scalability of this model and the size of the search space. A clearer view of scalability is seen in Figure 4.

6. 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 3: Structure of the Multi-Deme Parallel GA.

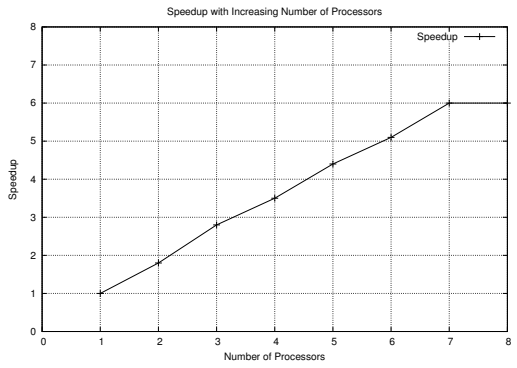

Figure 4: Speedup for circuit s386. The speedup pattern is almost identical for all circuits

Table 2: Global Selection 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	0.71	384	327	361	340	396	358	355	357
s386	0.69	1011	602	393	322	355	310	281	268
s832	0.56	1066	747	741	647	750	757	732	684
s1238	0.57	4406	2748	1887	1471	1196	1023	928	915
s641	0.51	5664	3361	2348	2154	1879	1761	1878	1646
s1196	0.54	4198	2686	1937	1679	1179	1027	948	930
s1494	0.53	4216	2757	1993	1520	1352	1141	1046	1052
s3330	0.41	10920	6663	5397	4630	4574	4270	4120	4055

Table 3: 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.

7. 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.