# Multiobjective VLSI Cell Placement Using Distributed Simulated Evolution Algorithm

Sadiq M. Sait  Mustafa I. Ali  Ali Mustafa Zaidi

King Fahd University of Petroleum & Minerals
Computer Engineering Department
Dhahran 31261, Saudi Arabia
E-mail: {sadiq,mustafa,alizaidi}@ccse.kfupm.edu.sa

*Abstract*— **Simulated Evolution (SimE) is a sound stochastic approximation algorithm based on the principles of adaptation. If properly engineered it is possible for SimE to reach near-optimal solutions in lesser time then Simulated Annealing [1], [2]. Nevertheless, depending on the size of the problem, it may have large run-time requirements. One practical approach to speed up the execution of SimE algorithm is to parallelize it. This is all the more true for multi-objective cell placement, where the need to optimize conflicting objectives (interconnect wirelength, power dissipation, and timing performance) adds another level of difficulty [3]. In this paper a distributed parallel SimE algorithm is presented for multiobjective VLSI standard cell placement. Fuzzy logic is used to integrate the costs of these objectives. The algorithm presented is based on random distribution of rows to individual processors in order to partition the solution and distribute computationally intensive tasks while efficiently traversing the complex search space. A series of experiments are performed on ISCAS-85/89 benchmarks to compare speedup with serial implementation and other earlier proposals. Discussion on comparison with parallel implementations of other iterative heuristics is included.**

## I. Introduction

Simulated Evolution algorithm (SimE) is a general search strategy for solving a variety of combinatorial optimization problems [2]. It operates on a single solution, termed as $population$. Each population consists of elements. In case of the placement problem, these elements are cells to be moved. The algorithm has one main loop consisting of three basic steps, Evaluation, Selection and Allocation. In the *Evaluation* step, *goodness* of each element is measured as a single number between '0' and '1', which is an indicator of how near the element is from its optimal location: a higher value of $goodness$ means that the element is near its optimal location. For single objective optimization, the goodness can be calculated as $g_i = \frac{O_i}{C_i}$, where $O_i$ is the estimated optimal cost and $C_i$ is the actual estimate of the cost.

Then comes *Selection*, which is the process of selecting elements which are unfit (badly placed) in the current solution. For that purpose, the goodness of each individual is compared with a random number (in the range [0,1]); if the goodness is less than the random number then the cell is selected. Therefore, an individual having high goodness measure still has a non-zero probability of being *selected*. It is this element of non-determinism that gives **SimE** the capability of escaping local minima. The next step, *Allocation*, has the most impact on the quality of solution. Its main function is to mutate the population by altering the location of selected cells.

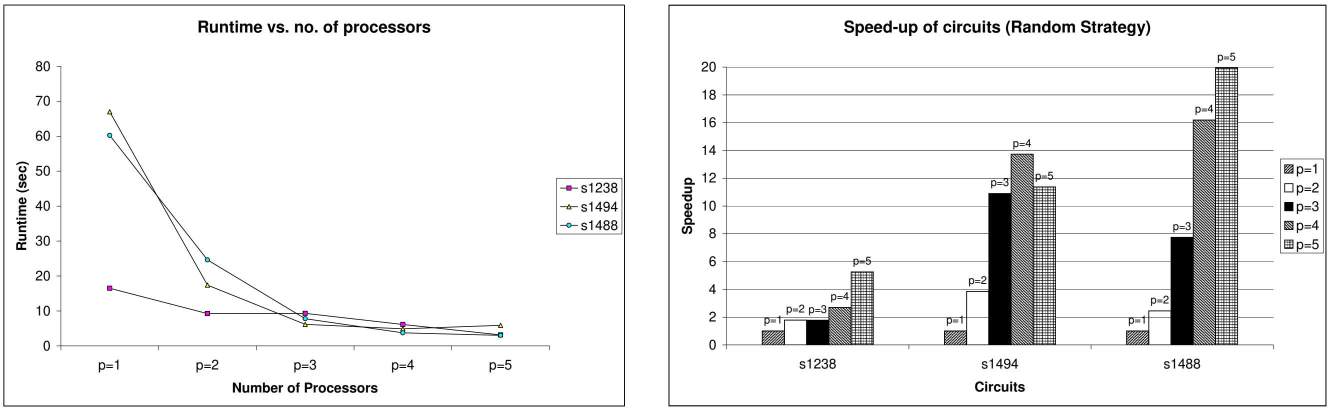
The above three steps are executed in sequence until no noticeable improvement to the population goodness is observed after a number of iterations, or a fixed number of iterations are completed.

The pseudo-code of SimE is similar to that given in Figure 1 [1]. Although the illustration depicts the slave process to be discussed later, if the entire set of rows is allocated to a single processor, then the execution of the algorithm is the same as that of the serial SimE.

For large test cases, SimE has large runtime requirements. The reason is that, like other stochastic iterative algorithms, SimE is blind. It has to be told when to stop. Depending on which stopping criteria are used, as well as the size of the problem, SimE may consume hours of CPU time before it stops. The most practical approach to speed up the execution of SimE algorithm is to parallelize it. Unlike Simulated Annealing [4] Genetic Algorithms [5], [6] and Tabu Search [7] the parallelization of SimE has not been the subject of much research. Kling and Banerjee suggested three ways of speeding up the SimE algorithm: (1) a distributed memory MIMD parallel algorithm, (2) a shared memory MIMD parallel algorithm that is based on an extension of basic SimE using the concepts of windowing and hierarchy, and (3) hardware acceleration which consists of implementing time consuming parts in hardware (namely, goodness computation) [2], [8].

A parallelization strategy for VLSI cell placement for a single objective (wirelength) was attempted on a network of workstations [9], where each station is assigned a number of rows of the placement problem, in a pre-determined order. The stations executes one iteration of the SimE algorithm on the cells of the rows assigned to it. In each iteration, the rows are redistributed among the processors [2], [9].

In the next section we present the details of our NP-hard, multiobjective, VLSI cell placement problem. Problem formulation and models for estimating the costs for the various objectives to be optimized are presented. In Section III, the distributed algorithm is detailed. Experimental setup, results obtained on ISCAS benchmark circuits, and other observations are given in Section IV, followed by Conclusion in Section V.




a b

Fig. 3. (a) The decrease in runtime to reach a pre-defined fitness objective with increasing number of processors; (b) Speedup versus number of machines.

## II. Problem Formulation

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 by using a cluster of low cost PCs. The goal is to achieve a placement quality very near or equal to that achieved by serial algorithm, but with run times that decrease linearly (or super-linearly) with increasing number of processors.

### B. Cost Functions

The objectives considered in our problem include: optimizing power consumption, improving timing performance (delay), and reducing overall wirelength, while, considering layout width as a constraint. A semi-formal description of the placement problem can be found in [3]. The multiobjective cost function is similar to the one formulated in [10], [11]. 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 $Cost_{delay} = max\{T_\pi\}$ in a circuit. The delay $T_\pi$ of a path $\pi$ 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 $vi$ 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 layout width is constrained not to exceed a certain positive ratio $\alpha$ 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, this can be expressed as $Width - w_{avg} \leq \alpha \times w_{avg}$.

### 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 the form of a single quantity. We use 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.

The above rule is translated to *and-like* OWA fuzzy operator [12] and the membership $\mu(x)$ of a solution $x$ in fuzzy set *GOOD solution* is obtained by:

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

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. $\beta$ 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 [10].

## III. Distributed Simulated Evolution Algorithm

The parallelization of the SimE algorithm is carried out by partitioning the workload among available processors. The workload for each slave in the cell placement problem is the

```
Algorithm Slave_Process(CurS, Φ_s)
Notation
 (* B is the bias value. *)
 (* CurS is the current solution. *)
 (* Φ_s are the rows assigned to slave s. *)
 (* m_i is module i in Φ_s. *)
 (* g_i is the goodness of m_i. *)
Begin
    Receive Placement_ And_ Indices
 Evaluation:
    ForEach m_i ∈ Φ_s evaluate g_i;
Selection:
    ForEach m_i ∈ Φ_s DO
      Begin
        If Random > Min(g_i + B, 1)
        Then
          Begin
            S = S ∪ m_i; Remove m_i from Φ_s
          End
      End
Sort the elements of S
Allocation:
      ForEach m_i ∈ S Do
        Begin
          Allocate(m_i, Φ_s)
 (* Allocate m_i in local partial solution rows Φ_s. *)
        End
Send_Partial_Placement_Rows
End. (*Slave_Process*)
```

Fig. 1. Structure of the Distributed Simulated Evolution Algorithm.

```
 Algorithm Parallel_Simulated_Evolution
   Read_User_Input_Parameters
   Read_Input_Files
Begin
   Construct_ Initial_ Placement
   Repeat
   Generate Random_ Row-Indices
       ParFor
         Slave_ Process(CurS, Φ_s)
   (* Broadcast Cur Placement And Row-Indices. *)
       EndParFor
       ParFor
         Receive_Partial_Placement_Rows
       EndParFor
     Construct_Complete_Solution
     Calculate_Cost
   Until (Stopping Criteria is Satisfied)
 Return Best_Solution.
End. (*Parallel_Simulated_Evolution*)
```

Fig. 2. Outline of Overall Parallel Algorithm.

computation of SimE operations of Evaluation, Selection, and Allocation on it's assigned rows [13].

The partitioning is done according to rows, which results in minimal dependency between the partitioned data, thereby reducing expensive communication between slaves. When there is not much variation in the number of cells per row, the workload is partitioned equally among the processors.

The row allocation pattern that was proposed in [13] is made up of two alternating sets of rows. In the even iterations, each slave gets a slice of $\lceil \frac{K}{m} \rceil$ rows, (where $m$ is the number of slaves, and $K$ is the total number of rows in the placement) while in the odd iterations the $j^{th}$ slave gets the set of rows $j$, $j+m$, $j+2m$, and so on. It has been shown that with the above fixed pattern of assigning rows to slaves in alternate steps, each cell can move to any position on the grid in at most two steps [9].

The consequence of row partitioning however is that the each processors has only a partial view of the placement. This hinders free cell movement, making it more difficult for cells to reach their optimal locations. Results from implementing this strategy on our multiobjective optimization problem revealed that even when given a large amount of time, the best solution obtained was poorer than one achieved by the serial implementation.

Though the lack of a global placement view will always exist in case of a distributed algorithm, the effects of restrictive cell movement can be alleviated by using a better row allocation pattern. Use of a pattern that induces a variety of mixing among the rows sounds intuitively better. This lead us to the use of a random row allocation pattern.

The pseudo code of the parallel simulated evolution is illustrated in Figures 1 and 2. As can be seen, one of the processor (the master) is in-charge of running SimE on a particular partition as well as performing the following tasks periodically at the end of each SimE iteration: (1) receive the partial placements from all other processors and combine them into a new solution and evaluate its fitness, (2) partition the new solution to obtain a new row allocation, and finally, (3) distribute the resulting sub-populations among the processors. The number of rows randomly assigned depends on the size of the placement and the number of processors. This is repeated for all iterations until the termination condition is met.

## IV. Results and Discussion

The parallel SimE strategy mentioned was implemented in C/C++ using MPICH Message Passing Interface implementation ver 1.2.4. for communication between nodes [**?**]. The experimental environment used consists of a dedicated cluster of 8 Pentium IV 2 GHz PCs with 256 MB RAM, running RedHat Linux ver 7.xx connected with a fast Ethernet switch. ISCAS-89 circuits are used as performance benchmarks for evaluating the parallel SimE placement techniques. These circuits are of various sizes in terms of number of cells and paths, and thus offer a variety of test cases.

Table I shows the amount of time taken to reach a predefined fitness objective with increasing number of processors for both the prosed random row allocation strategy, and the fixed row allocation strategy. For the proposed strategy, as can be seen, there is a constant decrease in runtime for all circuits. Better trends are observed for medium to large circuits, than for smaller ones, as can be seen in Figure 3(a). Speedup is also illustrated in the bar-chart given in Figure 3(b). Due to space restrictions, and scaling factor limitations, not all results have been included in the same figure for sake of clarity.

The fitness values achieved with the proposed row allocation are consistently higher in all test cases when compared to the fixed row allocation scheme, with runtimes decreasing with increase in number of processors. The speedup is super linear in most cases and is attributed to modified working space of the selection and allocation operators on each slave, as in each iteration different sets of rows are addressed. This has resulted in even more rapid solution times than with workload partitioning alone.

Though the fitness values achieved are not constant among all all circuits, a lower fitness does not necessary imply an inferior solution for a particular circuit, as the quality is relative to the initial starting solution, and the lower bounds estimate for the objectives.

### A. *Comparison With Other Iterative Heuristics*

The runtimes and solution quality was also compared with those obtained from parallelizing genetic algorithms (a distributed search space parallel strategy) [5], tabu search [7] and simulated annealing [4]. For GAs, the time for completion to obtain solutions of a certain pre-specified quality were exorbitantly high. And in some cases, for the given run-time, acceptable solutions could not be obtained. For example, for the S1494, the serial implementation took 1883 seconds, and when the parallel version was executed on 7 processors the best time was 418 seconds (with 8% inferior quality than that obatined by SimE).

Since cost computation of new generated solutions is very expensive in our problem, TS was parallelized by partitioning and distributing the candidate list to various slaves. While better quality was obtained in some cases at the cost of high computation time, for the same quality the run-time requirements for TS were about 3 times more than that required by parallel SimE. For example, for s1494, the time taken by serial TS was 268 seconds, and when parallel TS was run on 6 processors, the runtime for similar quality as SimE was 57 Seconds with better quality, and three times more time for the same quality.

A similar trend was seen for all circuits.

For simulated Annealing, the Asynchronous Multiple-Markov Chain Parallelization Strategy was chosen [reference]. Like TS, Parallel SA was also able to achieve better quality solutions than SimE, given enough time. However, for a fixed lower quality, SimE was seen to be increasingly faster than SA as processors were increased. For instance, for s1494, with 2 processors SA took 86 seconds to achieve the desired quality, while SimE took only 17 seconds. With 5 processors, SA required 63 seconds on average, while SimE needed only 6. Similar trends are seen for most circuits.

For medium quality solutions, Distributed Simulated Evolution with Random Row distribution has exhibited dramatic speedups as the number of processors is increased, even when compared to other, more established heuristics such as Tabu Search and Simulated Annealing. The results obtained suggest that in scenarios where placement quality considerations are overridden by design time constraints, the use of Distributed Simulated Evolution with Random Row Distribution should be favored over the more established heuristics like TS and SA.

## V. Future Work, Conclusion & Discussion

This paper presented the application of a modified Distributed SimE algorithm to a multi-objective VLSI cell placement problem. The algorithm focused on distributing the work load among processors. Random allocation of work load in each iteration resulted in better traversal of search for our complex multiobjective NP-hard design problem.

The results showed a significant reduction in runtime for all circuits, although the speedup was more obvious for larger ones. This speedup trend was compared to a parallel Tabu Search approach from literature, and was shown to be more consistent with increasing number of processors.

This work can be extended along several lines. One would be to investigate suitable parameters for the SimE algorithm that will enable better quality and run-times. At the moment, the same parameters that have been set for serial execution are used. Another approach is to relieve computational resources during execution when the quality ceases to improve. This can be achieved by modifying the stopping criteria. If the quality does not improve for the last $j$ iterations on $k$ processors, then the number of processors can be reduced to $k-1$, and this can continue until all processors are relieved. The effects of this experiment will be, that while execution continues to improve the obtained best solution, the distribution of increased number of rows on reduced number of processors will enable exploring different regions of the search space in the same run, and will hopefully result in better quality with reduced resources. Our initial experiments on this idea have been encouraging.

## Acknowledgment

The authors thank King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia, for support under

TABLE I

TABLE DEPICTING THE RUN TIMES FOR A SPECIFIED FITNESS, FOR SERIAL, AND 2, 3, 4, AND 5 PROCESSORS, FOR BOTH RANDOM AND FIXED ROW ALLOCATION STRATEGIES. UH INDICATES UNREASONABLY HIGH TIMES.

| Circuit Name | # of Cells | $\mu(x)$ Rnd | $N_p$=1 | Random Row Distribution $N_p$=2 | $N_p$=3 | $N_p$=4 | $N_p$=5 | $\mu(x)$ Fixed | Fixed Row Distribution $N_p$=2 | $N_p$=3 | $N_p$=4 | $N_p$=5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s641 | 433 | 0.577 | UH | 4.99 | 4.97 | 3.99 | 3.87 | 0.460 | 9.14 | 1.08 | 0.76 | 0.55 |
| s1238 | 540 | 0.533 | 16.5 | 9.24 | 9.29 | 6.12 | 3.14 | 0.511 | 17.83 | 8.47 | 11.30 | 5.71 |
| s1494 | 661 | 0.542 | 67 | 17.4 | 6.15 | 4.88 | 5.89 | 0.446 | 2.77 | 1.85 | 1.76 | 4.34 |
| s1488 | 667 | 0.559 | 60.23 | 24.6 | 7.78 | 3.72 | 3.02 | 0.540 | 22.0 | 4.89 | 5.1 | 16 |
| s3330 | 1961 | 0.535 | UH | 678.02 | 115 | 108.5 | 49.14 | 0.181 | 316 | 215 | 4.6 | 3.4 |
| s5378 | 2993 | 0.385 | UH | 1620 | 338.2 | 286.6 | 178.6 | 0.180 | UH | UH | 124.3 | 95.0 |

Project Code COE/CELLPLACE/263.

## REFERENCES


[1] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. IEEE Computer Society Press, California, December 1999.

[2] R. M. Kling. *Optimization by Simulated Evolution and its Application to cell placement*. Ph.D. Thesis, University of Illinois, Urbana, 1990.

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

[4] Robert Azencott, editor. *Simulated Annealing Parallelization Techniques*. John Wiley & Sons, 1992.

[5] Erick Cant-Paz. Designing efficient master-slave parallel genetic algorithms. *Genetic Programming*, 1998.

[6] 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.

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

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

[9] Ralph Michael Kling and Prithviraj Banerjee. Concurrent ESP: A placement algorithm for execution on distributed processors. *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 354–357, 1987.

[10] 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.

[11] 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.

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

[13] Ralph Michael Kling and Prithviraj Banerjee. ESP: A new standard cell placement package using simulated evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.