Parallel Simulated Annealing Algorithm for Standard Cell Placement in VLSI Design

Aaquil Bunglowala, Ph.D
Department of Electronics & Communication
Sri Aurobindo Institute of Technology
Indore, M.P. – India

Manisha Jain
Department of Engineering Mathematics
Sanghvi Institute of Management & Science
Indore, M.P. – India

ABSTRACT
Simulated Annealing (SA) is a stochastic based heuristic optimization technique based on physical process of metal crystallization. Optimization of Non-Deterministic polynomial hard (NP-hard) problems of non-trivial sizes is done using heuristic approach. Until now, simulated annealing (SA), genetic algorithm (GA) and Hopfield neural network (HNN) were individually used for solving the standard cell placement (SCP) problem. Simulated annealing established as a powerful SCP optimization tool, its drawback has always been its appetite for computational resources. In light of this, we are interested in the application of these parallel simulated annealing algorithms with respect to standard cell placement.

Several generalized algorithms proposed for parallelizing simulated annealing, only a few have been applied to cell placement. Parallel moves has been the most popular strategy and in this paper we present a new implementation of this approach.

General Terms
Standard Cell, Optimization, Parallel Moves, Non-Trivial

Keywords
NP-hard, Simulated Annealing, Hopfield Neural Network, Recombinative SA, PMSAA.

1. INTRODUCTION
The Standard Cell Placement (SCP) problem can be stated as: Give an electrical circuit consisting of fixed rectangular shaped cells and a netlist stating interconnection among terminals on the periphery of the cells and construct a layout on the periphery of the circuit itself, indicating the position of each cell such that all the nets can be routed and the total area s minimized [3].

Let c_1, c_2,…….c_k be the cells to be placed on the chip. Each c_n, $1 \leq n \leq k$, has associated with it a height H_n and width W_n. Let $N=\{n_1, n_2, n_3, \ldots\ldots n_j\}$ be the set of nets representing the interconnections between different blocks. Let $S= \{s_1, s_2, s_3, \ldots\ldots s_i\}$ represent the empty space allocated for routing between cells. Let L_n denote the estimated length of net. The placement problem now is to determine a rectangle for each of these blocks in the row denoted by $R=\{r_1, r_2, r_3, \ldots\ldots r_k\}$ such that:

1. Each cell can be placed in the corresponding rectangle in a r_n row with width W_n and height H_n.
2. No two rectangles overlap i.e. $r_p \cap r_q = \varphi$, $1 \leq p, q \leq k$.
3. The placement is routable, i.e.sr, $1 \leq r \leq i$, is sufficient to route all the nets.
4. The total area of the rectangle bonding R and S is minimized, minimizing the total area of chip.
5. The total wire length is minimal.

SCP is computationally NP-hard. These problems can not be solved in polynomial time. A heuristic needs to be used to search through a large number of candidate placement configurations efficiently.

We established three heuristic techniques for SCP problem in the previous publications. They included Genetic Algorithms (GA), Hopfield Neural Network (HNN), Simulated Annealing (SA) based approach [9, 10, 11].

2. SA ALGORITHM
The Simulated Annealing Algorithm, SAA, starts with a high value of the control parameter taken for temperature; and a set of parameters $\{x_i\}$ are defined. An initial solution for a parameter x_k is computed. This initial parameter is then perturbed causing the neighbourhood parameter x_l to generate a new solution. The value of the objective function, the cost, for the new solution is computed [9, 14].

2.1 SCP by Simulated Annealing
Kirkpatrick et al. tried to solve placement and routing problems occurring in VLSI design by the SAA. A great deal of research was conducted to improve the performance of SAA for SCP [1].

Information about the initial placement of the cells on the layout is shown in the table 1 in which the first row is a sequence of cells, the second and third rows provide the coordinates of the corresponding cells on the layout.

Table 1: Representation of Standard Cell Placement

CELL	A	B	C	D	E	F	G	H	I	J
X coordinate	0	20	50	75	90	0	30	55	70	95
Y coordinate	0	0	0	0	0	50	50	50	50	50

The basic algorithm is followed by the underlined procedure.

- Initially, the starting temperature is computed, which in this case is problem-specific.
- The initial solution is generated randomly and the x and y coordinates are determined.
- To generate new solutions, the neighbourhood operator is used.

Read netlist (input file)
*Set temperature **Temp** at high value*
Generate an initial valid sequence of cells randomly (Solution: S_K)
Based on the sequence K calculate x and y coordinates of each cell
Calculate the energy E_K of the initial solution
$S_{Best} = S_K$
$E_{Best} = E_K$
LOOP

LOOP
Iteration = Iteration +1
Apply neighborhood operator to modify S_K
generating new solution $\mathbf{S_L}$
For S_L *calculate x and y coordinates of each cell*
For S_L *calculate the energy* E_L
IF $E_L < E_K$ *THEN*
$S_K = S_L$ *and* $E_K = E_L$
$S_{Best} = S_L$ *and* $E_{Best} = E_L$
Else
Choose a random number **n** *between 0 and 1*
IF $p < exp[-(E_L - E_K)/Temp]$ *THEN*
$S_K = S_L$ *and* $E_K = E_L$
$S_{Best} = S_L$ *and* $E_{Best} = E_L$
UNTIL Iteration reaches a fixed given number
IF P_O & P_R *!= 0 THEN switch to external cycle*
Lower the value of Temp
UNTIL termination criteria

Algorithm 1: SA Algorithm for SCP problem

2.2 Results of SAA with Different Cooling Schedules

A large number of tests were performed with test cases of varying problem sizes and varying complexities of netlists [12]. The problem sizes range from 20 cells to 120 cells. The description of the test cases is given in table 2.

Table 2 : Test cases for establishing the results

Test Case	Number of			Average cells per Net	Average width of cells
	Cells ami- lib	Nets	Rows		
A	20	20	4	6	30
B	20	18	4	5	30
C	20	16	4	5	30
D	40	15	5	8	25
E	40	14	5	8	25
F	40	11	5	10	25
G	80	22	8	10	25
H	80	24	8	12	20
I	120	36	12	8	35
J	120	44	12	10	35

The important parameters of simulated annealing are the stating temperature, the factor by which temperature is reduced and the lowest temperature at which the algorithm should stop.

The temperature after each inner loop is lowered by a fixed factor 0.995. Hence the cooling can be expressed mathematically as $T_{new}=0.995*T_{old}$.

The most factor parameter is the cooling factor. With regard to this factor, four versions of SAA were tested:

SAA-I: with cooling factor: 0.98
SAA-II: with cooling factor: 0.999
SAA-III: with cooling factor: 0.9999
SAA-IV: with cooling factor: 0.99999

Table 3: Results of SAA [Wire Length]

Test Case	Number of		Average Cell Width	Result Quality	Wire Length (µm)			
	Cells	Nets			SAA I	SAA II	SAA III	SAA IV
A	**20**	**20**	**30**	Best	3708	3695	3715	3680
B	**20**	**18**	**30**	Best	2886	2828	2768	2744
C	**20**	**16**	**30**	Best	2744	2728	2720	2720
D	**40**	**15**	**25**	Best	3548	3526	3486	3428
E	**40**	**14**	**25**	Best	2528	2546	2528	2528
F	**40**	**11**	**25**	Best	2465	2564	2465	2465
G	**80**	**22**	**25**	Best	3777	3726	3688	3658
H	**80**	**24**	**20**	Best	3850	3850	3850	3850
I	**120**	**36**	**35**	Best	5788	5723	5640	5638
J	**120**	**44**	**35**	Best	6754	6003	5988	5947

Table 4 : Results of SAA [CPU-Time]

Test Case	No. of		Average Cell Width	Result Quality	CPU Time (in second)			
	Cell	Net			SAA I	SAA II	SAA III	SAA IV
A	**20**	**20**	**30**	Best	**97**	**130**	**288**	**935**
B	**20**	**18**	**30**	Best	**103**	**144**	**296**	**944**
C	**20**	**16**	**30**	Best	**104**	**148**	**303**	**948**
D	**40**	**15**	**25**	Best	**224**	**246**	**964**	**2488**
E	**40**	**14**	**25**	Best	**215**	**238**	**948**	**2344**
F	**40**	**11**	**25**	Best	**208**	**234**	**932**	**2283**
G	**80**	**22**	**25**	Best	**648**	**1224**	**2432**	**7826**
H	**80**	**24**	**20**	Best	**785**	**1438**	**2834**	**9276**
I	**120**	**36**	**35**	Best	**1546**	**3012**	**5882**	**14586**
J	**120**	**44**	**35**	Best	**1726**	**3649**	**7512**	**18442**

3. SCP BY PARALLEL MOVES SAA

In Parallel Moves Simulated Annealing Algorithm, each processor generates and evaluates moves independently as if the other processors are not making any moves. One problem with this approach is that the cost function calculations may be incorrect due to the moves made by the other processors. This can be handled by either evaluating only moves that do not interact, or handle interacting moves with some error tolerance procedure [4,5,6,7].

3.1 Parallel Moves SAA [PMSAA]

PMSAA exploits parallelism by using parallel moves and allowing errors in the cost function. Using the Accumulate feature of the PMSAA, an Accumulate named Design is constructed to manage access to the Design structure and maintain a coherent state of the current placement. Each processor will have one manager of Accumulate responsible for its local copy of the Design. In addition, an Anneal Process [AP] is created per physical processor to perform the annealing steps - i.e. move, evaluate, and decide.[4] Figure 1 shows the relationship between the accumulate and its dependent AP.

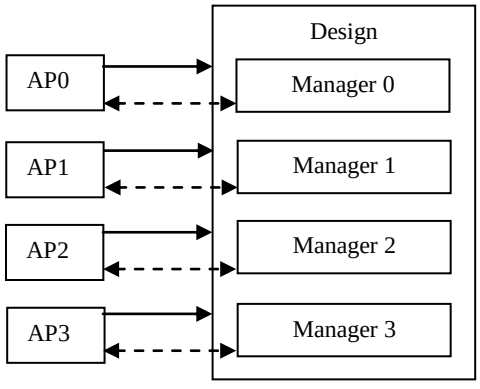

Figure 1: Processes in PMSAA

After the creation of the AP, the placement is divided up topographically by rows, with the rows and its cells assigned to separate AP**.** If the number of rows is not greater than or equal to the number of processors, the rows must be split into a number of sub-rows, in which case some overlap penalties may be calculated erroneously..

A valid cell is selected for perturbation, and then a displacement or exchange is performed on that cell. As detailed below, there are two sub-classes of moves for both displacement and exchange, or four move types in total. The move type is determined by the intended location of the selected cell A.

move1: Cell A moves to a new location owned by the same AP.

move2: Two cells B and C owned by the same AP exchange their locations.

move3. Cell D moves to a new location owned by a different AP.

move4. Two cells E and F owned by different APs are exchanged.

An example of each type of move is shown in Figure 2. In the figure, assume that each row is owned by different Anneal APs. Notice that the three moves (move1, move2, move3) can be done alone by AP0, the owner of cell A. For move4, however, AP0 needs permission from AP1 which owns cell F, as it is possible that cell F may have already been moved to another location or is frozen due to some pending move. Because the information about cell F may be out of date in the database of AP0, it locks (or freezes) cell E and cell F and sends a SeekPermission message to AP1. After receiving the SeekPermission message, AP1 examines the state of cell F and determines whether to allow the exchange. The decision is sent back to AP0 by sending the Reply message. Upon receipt of the Reply message, AP0 unlocks cells E and F, and the move is attempted if the reply is yes. AP0 does not wait idly until the Reply message is received - instead, it continues annealing by making other moves with unfrozen cells that it owns. Since the inter-AP exchange 'move4' takes the most time due to extensive message passing, we introduce a 'Message Priority' scheme to reduce the time taken by move4.

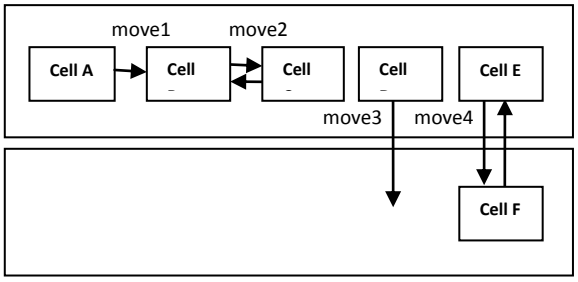

Figure 2: Moves in PMSAA demonstrated

If a move is accepted, then the accepting AP must send the move to the Design accumulate so a consistent cell position database can be maintained. In order to amortize the startup cost of sending a message, position update messages are held until a number of moves have been accepted. Although this reduces the total number of messages sent among processors. As the frequency of messages is reduced, the local cell position database on each Design representative becomes increasingly inaccurate, thereby causing the cost function calculation error to increase as well. This error, if too large, may prevent the algorithm from converging to an optimal solution.

Since AP methods are non-blocking, the AP's annealing process must give up control every so often to allow the accumulate to gain computation time to perform the updates. Therefore, a limit is placed on the number of moves that may be performed in succession without interruption. The Design Accumulate can then process any waiting Update messages to keep the local database up-to-date. AP will have rescheduled itself by sending itself a Continue message that will enable control to come back to the AP and the next set of moves can then be proposed and evaluated.

After broadcasting its set of moves through the Accumulate, an AP does not wait idly until all the Update messages sent by other APs have been processed, but it goes ahead with the next sequence of simulated annealing moves. In synchronous approach to parallelization APs finishing a block of moves must wait for slower APs to finish. The time to evaluate different moves is not the same, leading to some APs remaining idle, and thus reducing the overall speedup. In an asynchronous approach, APs become idle only at the end of the entire simulated annealing procedure. The overall idle time will have been reduced leading to greater speedup than the synchronous method. However, synchronization does offer an advantage in that the error in the cost function calculation becomes zero at each synchronization barrier, thereby making error control much easier. In the asynchronous approach an effective error control scheme is necessary to control the accumulated error in the system.

3.3 Results for PMSAA

Results obtained by using PMSAA on test cases A-J discussed in table table 2 are shown in table 5.

Table 5 presents the wire-length in case of PMSAA for two, four and eight parallel processors in comparison to uni-processor based SAA-IV. PMSAA gives a small overhead of 2% to 7% increase in wire lengths in all the test cases.

Table 5: Results of PMSAA viz. a. viz. SAA [WL]

Test Case	Number of		Average Cell Width	Result Quality	Wire Length (µm)			
	Cells	Nets			SAA IV	PM SAA 2P	PM SAA 4P	PM SAA 8P
A	**20**	**20**	**30**	Best	3680	3761	3788	3938
B	**20**	**18**	**30**	Best	2744	2804	2826	2936
C	**20**	**16**	**30**	Best	2720	2774	2800	2902
D	**40**	**15**	**25**	Best	3428	3503	3531	3668
E	**40**	**14**	**25**	Best	2528	2580	2598	2710
F	**40**	**11**	**25**	Best	2465	2520	2532	2638
G	**80**	**22**	**25**	Best	3658	3738	3768	3914
H	**80**	**24**	**20**	Best	3850	3928	3958	4128
I	**120**	**36**	**35**	Best	5638	5762	5807	6033
J	**120**	**44**	**35**	Best	5947	6078	6125	6363

Table 6: Results of PMSAA viz. a. viz. SAA [CPU-Time]

Test Case	No. of		Avg. Cell Width	Result Quality	CPU Time (in second)			
	Cells	Nets			SAA I	PM SAA 2P	PM SAA 4P	PM SAA 8P
A	**20**	**20**	**30**	Best	935	497	443	322
B	**20**	**18**	**30**	Best	944	502	447	326
C	**20**	**16**	**30**	Best	948	504	449	327
D	**40**	**15**	**25**	Best	2488	1220	1091	818
E	**40**	**14**	**25**	Best	2344	1149	1028	771
F	**40**	**11**	**25**	Best	2283	1119	1001	751
G	**80**	**22**	**25**	Best	7826	3692	3316	2516
H	**80**	**24**	**20**	Best	9276	4375	3931	2983
I	**120**	**36**	**35**	Best	14586	6570	5881	4354
J	**120**	**44**	**35**	Best	18442	8307	7436	5505

4. CONCLUSIONS

Simulated Annealing is tolerant to some error in cost function calculations while in PMSAA the frequency of sending position Update messages is determined adaptively such that the error in the cost function is kept small at all times. As shown in table of result for PMSAA and SAA, PMSAA gives a significant improvement in execution time as compare to SAA. The improvement is in the order of 2 to 3 times to that of SAA-IV. Also as the size of test set increases the speed-ups are more significant. All these indicators point to the fact that the parallel algorithms are better utilized for higher end complex designs. From the above tables it is clear that PMSAA produces acceptable speedups, while maintaining quality comparable to that of SAA (refer figure 3 and figure 4).

5. REFERENCES

[1] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1989. Optimization by Simulated Annealing. Science 220, 671-680.

[2] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E. 1953. Equation of State Calculation by Fast Computing Machines. J. of Chem. Phys., 21, 1087-1091.

[3] Khushro, Shahookar, Pinaki, Mazumder 1990. A Genetic Algorithm for Standard cell Placement", Proceedings of EURO-DAC, 660-664.

[4] Banerjee, M., Jones, H. & Sargent, J. S. 1990. Parallel Simulated annealing algorithms for standard cell placement on hypercube multiprocessors. IEEE Transaction on Parallel and Distributed Systems.

[5] Durand, M. D. 1989. Accuracy vs. speed in placement. IEEE Design & Test of Computer. 8-34.

[6] Kravitz, S. A., & Rutenbar, R. A. 1987). Placement by simulated annealing on a Multiprocessor. IEEE Transaction on Computer-Aided Design 6.

[7] Rose, J. S., Snelgrove, W. M., & Vranesic, Z. G. 1988. Parallel standard cell placement algorithms with quality equivalent to si.mulated annealing. IEEE Transactions on Computer- Aided Design 7.

[8] Rutenbar, R. A. 1989. Simulated Annealing Algorithms, An Overview. IEEE Circuits and Devices Magazine, 5, No. 1, 19-26.

[9] Bunglowala, A., & Singhi, B. M. Memetic Algorithms as a Solution to combinatorial Optimization Problem. Proceedings of 2nd PIMR International Conference.

[10] Bunglowala, A., Singhi, B.M. 2008. Performance Evaluation And Comparison and Improvement of standard Cell placement in VLSI Design. International Conference on Emerging Trends in Engineering and Technology.

[11] Bunglowala, A., & Singhi, B. M. 2008. A solution to Combinatorial Optimization Problem using Memetic Algorithm. International Journal of Computer System Application.

[12] Hajek, B. 1988. Cooling Schedules for Optimal Annealing. Mathematics of Operations Research, 13, No. 2, 311-329.

[13] Beumont, O., Legrand, A. and Robert, Y. 2003. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Processing Symposium.

[14] Mitra, D., Romeo, F., and Sangiovanni-Vincentelli, A. 1986. Convergence and Finite Time Behavior of Simulated Annealing. Advances in Applied Probability, 18, 747-771.

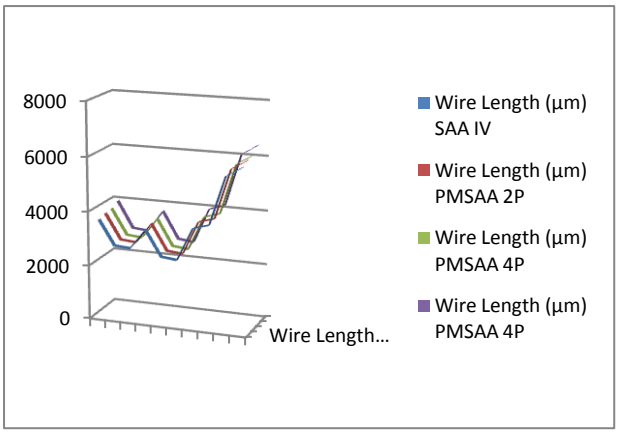

Figure 3: WL of PMSAA-2, 4, 8P viz. a. viz. SAA-IV

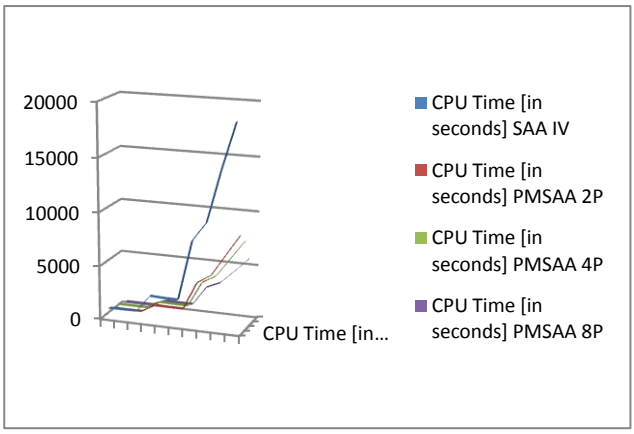

Figure 4: CPU Time of PMSAA-2, 4, 8P viz.a.viz. SAA-IV