Soft Computing Approach for Digital Circuit Layout based on Graph Partitioning

Maninder Kaur
Assistant Professor,
School of Mathematics and
Computer Applications,
Thapar University, Patiala.

Kawaljeet Singh
Director,
University Computer Center,
Punjabi University, Patiala

ABSTRACT
Soft computing based approaches are increasingly being used to solve different NP complete problems, the development of efficient parallel algorithms for digital circuit partitioning, circuit testing, logic minimization and simulation etc. is currently a field of increasing research activity. This paper describes evolutionary based approach for solving circuit-partitioning problem. That implies dividing a circuit into non-overlapping sub circuits while minimizing the number of cuts after the division and balancing the load associated to each one. The paper shows the effective partitioning for achieving peak chip performance and reducing the cost and time of the design and manufacturing process.

Keywords
Soft computing, VLSI circuits, Circuit partitioning, Genetic algorithms, simulated annealing

1. INTRODUCTION
The main objects in VLSI design are placements of rectangles and lines, and the basic problems in physical design deal with arrangements of these rectangles on circuit layout. The relationship between these objects, such as overlap and distance, are very critical in development of physical design algorithms. Graphs are a well-developed tool used to study relationships between the objects. Naturally, graphs are used to model many VLSI physical design problems and they play a very important role in almost all VLSI design algorithms. In this paper, we introduce various graphs that are used in modeling of physical design problems [1]. The process of digital circuit design can be thought as a sequence of the various steps: The outcome of the first step, high level synthesis, is usually a high level description of the circuit function and is expressed using high level language that describes the RTL that implements digital circuit behavior. The second step, logic synthesis optimization, performs an optimization on the result of the first step targeted to area minimization, power dissipation minimization and speed optimization. Subsequently the circuit is simulated and its operation is verified (simulation verification). Last but not least before manufacturing is the step of layout synthesis (placement and routing) which we examine in this work. This step is usually performed in four sub steps:

- *System partitioning*: the system circuitry is partitioned in circuits that usually perform a distinct function.
- *Floor planning*: the chip is partitioned in "rooms" called blocks similarly to architectural design and each block is assigned to a part determined in the first step.
- *Placement*: the placement of standard cell units inside each individual block area of the floor planning is determined.
- *Routing*: all connections (inter block and intra block) are routed so as to minimize the wire length and the number of routing metal levels.

Efficient designing of any complex system necessitates decomposition of the same into a set of smaller subsystem. Subsequently, each subsystem can be designed independently and simultaneously to speed up the design process. The process of decomposition is called partitioning. It plays a key role in the design of a computer system in general, and VLSI chips in particular. A computer system is comprised of tens of millions of transistors. It is partitioned into several smaller modules/blocks for facilitation of the design process. Each block has terminals located at the periphery that are used to connect the blocks. A VLSI system is partitioned at several levels due to its complexity. At the highest level, it is partitioned into a set of sub systems whereby each subsystem can be designed and fabricated independently on a single PCB. The circuit partitioning problem arises in many VLSI applications [1].

At any level of partitioning, the input to the partitioning algorithm is a set of components and a net list. The output is a set of sub circuits which when connected, function as the original circuit. In addition to maintaining the original functionality, partitioning process optimizes certain parameters subject to certain constraints [16]. The constraints for the partitioning problem include area constraints and terminal constraints. The objective function for a partitioning problem includes the minimization of the number of nets that cross the partition boundaries. Partitioning efficiency can be enhanced within three broad parameters.

- The system must be decomposed carefully so that the original functionality of the system remains intact.

- An interface specification is generated during the decomposition, which is used to connect all the subsystems. The system decomposition should ensure minimization of the interface interconnection between any two subsystems.

- Finally, the decomposition process should be simple and efficient so that the time required for the decomposition is a small fraction of the total design time.

2. RELATED WORK TO PARTITION PROBLEM

A number of heuristic techniques are there to generate approximate solutions to the partitioning problem. A constructive partitioning heuristic starts from a seed component. Then other components are selected and added to the partial solution until a complete solution is obtained. An iterative heuristic receives two things as inputs, one, the description of the problem instance, and two, an initial solution to the problem. The iterative heuristic attempts to modify the given solution so as to improve the cost function. Usually the constructive algorithms are deterministic while iterative algorithms may be deterministic or stochastic. Some of the classical approaches used to generate solutions to the partitioning problem are

- The Kernighan-Lin algorithm
- Genetic algorithm
- Simulated annealing
- Tabu Search

The most basic approaches to the partitioning problem treat the circuit as a graph. This is true for the first, and most famous partitioning algorithm, called the Kernighan-Lin algorithm [2]. This algorithm was originally designed for graph partitioning rather than circuit partitioning, so to apply the algorithm, one must first convert the circuit into a graph. The initial partition is generated at random. Then the two sub circuits S1, and S2 are created. If the circuit has n gates, the first n/2 are assigned to S1, and the rest are assigned to S2. Because the gates in a circuit description appear in what is essentially a random order, the initial partition appears to be random. A solution is acceptable only if both sub circuits contain the same number of gates. It is assumed that the number of gates is even. The algorithm can be tweaked to handle an odd number of gates. The goodness of a solution is equal to the number of graph edges that are cut. Suppose the edge (V, W) exists in the graph derived from the circuit. (V and W represent the gates.) There are two possibilities. V and W can be in different sub circuits, or they can be in the same sub circuit. If V and W are in different sub circuits, we say that the edge (V, W) is cut. Otherwise we say that (V, W) is uncut. The technique for generating new solutions from old solutions is to select a subset of gates from S1 and a subset of gates from S2 and swap them. To maintain acceptability, we always select two subsets of the same size.

Genetic algorithms (GA) were developed by John Holland [3] and since then have been used in various fields of engineering. GA has been used quite successfully for combinatorial problems that are NP-complete. More recently GA has been used for solving some VLSI problems [4 - 7]. A genetic algorithm is a randomized parallel search method modeled on natural selection and genetics [8]. In contrast to more standards search algorithms, GA bases their progress on the performance of a population of candidate solutions, rather than on a single candidate solution. The motivation behind this is that by simultaneously searching many areas of the design space the risk of getting stuck at local optima is greatly reduced. GA are probabilistic in nature and start off with a population of randomly generated candidates and evolve toward better solutions by applying genetic operators, modeled on the natural genetic process.

Simulated annealing [10] belongs to the class of non-deterministic algorithms. Kirkpatrik, Gelatt and Vecchi first introduced this heuristic in 1983. Simulated Annealing (SA) is a general iterative improvement algorithm that can be used for many different purposes. In partitioning, SA starts with a random partition. A new state is computed by selecting a gate at random from each of the two subsets, and swapping them. As before, the swap remains tentative, until the quality of the new partitioning is computed. The number of nets cut is the measure of goodness. If the new state is better than the old state, it is accepted and the swap is made permanent. If the new state is worse than the old state, it might be accepted and it might not.

Tabu search is another well-known heuristic approach that has been applied to a number of optimization problems. It was introduced by Glover. The Tabu heuristic combines the recent history of the optimization with iterative improvement to seek feasible solutions. A list of the *r* most recent moves is kept (with *r* a constant that is determined at the start of the optimization), where a move is defined in terms of the neighborhood operator [8]. The list is called the Tabu list (from "taboo") and indicates the moves that cannot be made (these moves have been prohibited). An aspiration criterion can be used to temporarily release a move from its Tabu status. Stopping criteria for the algorithm are also required and this may be a (fixed) maximum number of moves after which the search routine will terminate. The Tabu heuristic requires careful implementation as cycling behavior (where the algorithm repeats the same set of vertex moves over and over) must be avoided.

3. PROBLEM FORMULATION

The partitioning problem can be expressed more naturally in graph theoretic terms. A graph *G=(V, E)* representing a partitioning problem can be constructed as follows. Let $V=\{v_1, v_2 \ldots v_n\}$ be a set of vertices and $E=\{e_1, e_2 \ldots e_m\}$ be a set of edges. Each vertex represents a component. There is edge joining the vertices whenever the components corresponding to these vertices are to be connected. Thus, each edge is a subset of the vertex set i.e., $e_i \subseteq V,\ i=1,2 \ldots m$. Let *edge* represents a function which when called with first vertex of edge, returns the second vertex of edge. The modeling of partitioning problem into graphs allows us to represent the circuit-partitioning problem completely as a graph-partitioning problem. The partitioning problem is to partition V into $V_1, V_2 \ldots V_k$, where

$$V_i \cap V_j = \varnothing, \quad i \neq j \tag{1}$$

$$\cup V_i = V \tag{2}$$

These partitions can be obtained by first efficiently partitioning the graph into two parts and then recursively applying the same approach. Partition is also referred to as a cut. The cost of partition is called the cut size, which is the number of edges crossing the cut. However, at the chip level, the partitioning algorithms usually have interconnections between partitions as an objective function. The number of interconnections at any level of partitioning has to be minimized.

Reducing the interconnections not only reduces the delay but also reduces the interface between the partitions making it easier for independent design and fabrication. A large number of interconnections increase the design area as well as complicates the task of placement and routing algorithms. Minimization of the number of interconnections between partitions is called the min cut problem. The minimization of the cut is a very important objective function for partitioning algorithms for any level or any style of design. This function can be stated as:

$$\sum_{i=1}^{k} \sum_{j=1}^{k} c_{ij} \quad , \quad (i \neq j) \quad \text{is minimized} \qquad (3)$$

The c_{ij} represent the crossing edge from node *i* to node *j* crossing a partition. The min cut problem is NP complete, it follows that general partitioning problem is also NP complete [19]. The circuit bi-partitioning optimization is focused on finding an acceptable solution cut-set cost. The cut-set cost is the number of inter-partition connects, which if not selected carefully, will immensely degrade the overall solution quality

4. THE PROPOSED METHOD SCACP

As the general partitioning problem is NP complete, various heuristic algorithms for partitioning have been developed [2,15,16] .Kernighan and Lin [2] proposed a graph bisection algorithm for a graph having the time complexity of $O(n^3)$, which is considered too high even for moderate size problems. However the similar bisecting approach can be used much more effectively by applying the SCA computation to circuit partitioning problem. This paper presents such an approach, which makes use of SCA computing to solve the instance of partitioning problem. The advantage of the proposed method is that output values are computed and stored parallel. The presented approach makes use of soft computing method [17] for computing the solution that has the obvious advantage that the SCA strands used in the process are reusable.

To solve the instance of Partitioning problem with $G=(V, E)$ ($|v| = n$) start with 2^n identical single stranded SCA memory strands each with $2n$ bit regions. The first *n* bit regions will represent the presence/absence of vertex in the first partition and the rest *n* bit regions will represent the presence/absence of an edge crossing the partition.

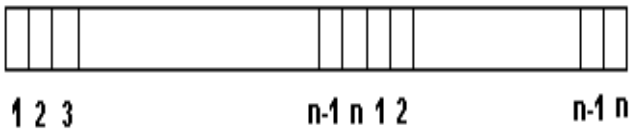

Fig 1: Bit regions of a SCA Strand.

The algorithm begins with an initialization phase during which each individual in the population is defined by a uniformly distributed random point in the solution space. Each iteration of the SCA begins evaluating the fitness of the current generation. The application of crossover and mutation to the individuals creates a new generation of offspring. *Solution representation.* Let *K* be the number of sub circuits into which the circuit with graph G is divided, and let *n* ($n=|P|$) be the number of logic gates of the original circuit; then each solution is represented by an array *S* of *n* elements as *S*=*A*1, *A*2, *A*3, ... , *An.* with *Ai* in [1,....,K] where the *Ai* element in array S represents the subcircuit to which the logic gate *i* belongs to. *Initial solution.* Let *N* be the population size, the algorithm has been run using two different initial solutions, *s1* or *s2*. They have been obtained using fast algorithms that assign *n/K* strongly connected nodes to each partition.

Partitioning is done in which the circuit graph is traversed in a depth-first way starting from the inputs, obtains the initial solution *s1*. The fitness function is obtained directly from the cost function. *Selection criteria*. After evaluating all the population with the fitness function, the individuals of the next generation will be chosen by a proportional criterion, called *roulette proportional criterion*, which will guarantee that the best individuals of the current generation have more possibilities of passing to the next one.

In the SCA proposed here, the operators are based in moves of gates between neighborhood partitions. The variants are deterministic, pseudo-random and random. When the current generation is equal to this input, the algorithm finishes. Other condition to finish the evolutionary process occurs when the solutions don't improvement. Thus, when the solutions in the last *N* generations are very similar (almost *n* improvement results) the algorithm finishes. Parameter *N* depends of the circuit, because in a large circuit there are more possibilities of escaping from a local minimum than in a smaller one.

5. SIMULATION RESULTS

The experiment has been performed by taking the different sets of graphs for different values of *n* (number of nodes) and *v* (set of edges). The results are plotted between the number of nodes *n*, which specify a search space of size 2^n (along x-axis) and total number of iterations taken to find the required partition (along y-axis). Comparison has been made between Kernighan-Lin, Simulated Annealing, Genetic Algorithm and SCA based approach. SCA notations are developed utilizing the properties of massive parallelism and replications. The corresponding results are shown in Table 1.

The Kernighan-Lin (K-L) algorithm starts with a random initial partition and then uses pair wise swapping of vertices between partitions with a complexity $O(n^3)$. In simulated annealing we try to search the required partitions from a given starting configuration. In genetic algorithm we have used crossover, mutation and population for finding the partition. Three different graphs have been plotted between Kernighan-Lin, simulated annealing, genetic algorithm and SCA based approach for different sets of values of *n* and edges *v*.

Table. 1: Comparison among the proposed algorithm and other partitioning techniques

Partitioning Algorithms	**K-L**	**SA**	**GA**	**SCACP**
Decomposition Time(s)	25	19	15	10
Iterations	89	76	45	22

Per Iteration Time(s)	0.5353	0.4231	0.2153	0.999

- Partitioning size = 8 cuts.
- Decomposition Time: Time taken to prepare the partition for each of the different partitioning algorithms.
- Number of Iterations: Number of iterations to convergence taken by CPU.
- Iteration times: Average time for a single iteration

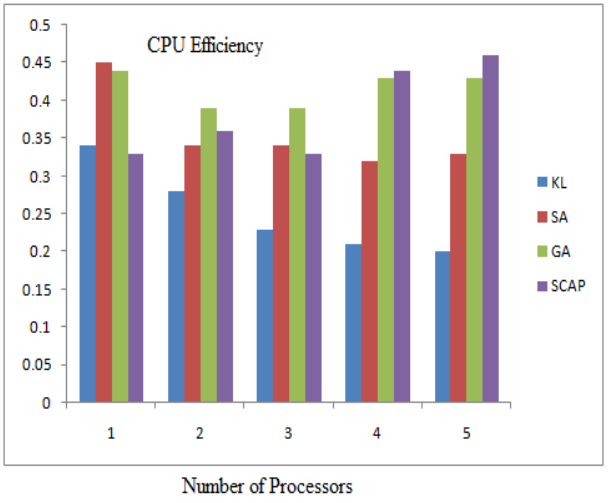

Fig 2: Shows the number of processors of different partitioning Algorithms.

The performance and efficiency of the proposed algorithm in comparison with other deterministic algorithms is done by simulation. The results shows that proposed SCA based algorithm gives better performance measure in terms of CPU efficiency compared to other algorithms.

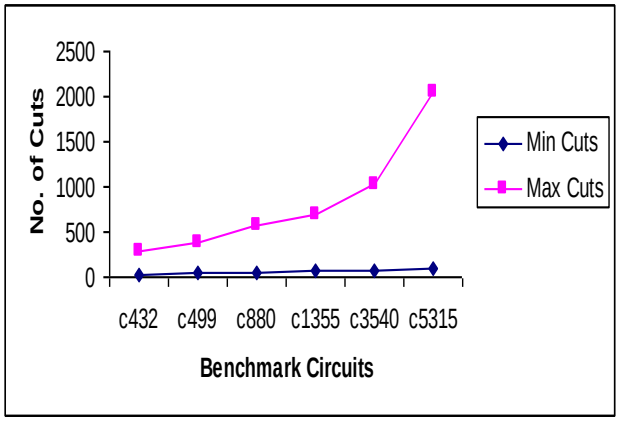

Fig 3: Shows the number of cuts on different Benchmark circuits

- If both the row *i* and the column *i* of the edges are not cut, then assign vector elements *xi* and *bi* to the processor assigned vertices from row *i* and column *i* graphs shown in Figure 2.
- If the row *i* edge is cut and the column *i* edge is not cut, then assign vector elements *xi* and *bi* to the processor assigned vertices from column *i* edge.
- If the row *i* edge is not cut and the column *i* edge is cut, then assign vector elements *xi* and *bi* to the processor assigned vertices from row *i* edge.
- If both the row *i* edge and the column *i* edge are cut, then let *Ri* denote the set of processors that contain row *i* edge elements and let *Ci* denote the set of processors that contain column *i* edge elements.

Table 2: Benchmarks ISCAS ’85

Circuit	Nodes	No. of Gates	Partition Area	Min Cuts	Max Cuts
c432	**15**	**153**	**1232**	**29**	**289**
c499	**21**	**170**	**2346**	**45**	**376**
c880	**26**	**357**	**5542**	**58**	**578**
c1355	**48**	**514**	**7754**	**68**	**679**
c3540	**69**	**1647**	**9887**	**78**	**1023**
c5315	**93**	**1723**	**10234**	**94**	**2056**

The simulation work was carried out by writing a program in *C++* and running on a SUN Ultra SPARC-III processor under the Sun Blade workstation architecture. The program has been written with approximately 3,500 lines of code. To evaluate the SCACP, several experiments are performed on different circuits. In order to allow readers to more easily evaluate the results, the SCACP is compared with other existing methods.

The circuits used to evaluate the performance of the algorithms proposed here are the ISCAS´85 that are common benchmark circuits as shown in Table 3 and Figure 3 in the context of test pattern generation, application where the circuit partitioning problem appears

6. CONCLUSION AND FUTURE SCOPE

The results show that the proposed SCA algorithm is able to partition the circuit graph taking less no. of iterations as compared to other available approaches. Exhaustive search is clearly not feasible because complexity grows exponentially as the search space increases. Simulated Annealing on the other hand is effective in parallel environment. Genetic Algorithm is takes large amount of CPU time and also numbers of iterations are more. In these situations SCA algorithms provide the better

solution by taking the help of massive parallelism and recombination properties of SCA.

SCA computing like much of today's cutting-edge research is multidisciplinary. It involves mathematics, biology and computer science. In many papers, [20] and [21] it is stressed that high levels of collaboration between academic disciplines will be essential to gear up the progress in SCA computing. Many of the search problems in computer science are unsolvable not theoretically but because of astronomical resources required for their solution. SCA Algorithms can be applied to these problems. They can be used to extract statistics such as mean, median, minimum element from an unsorted data that can be used to speed up algorithms for NP problems. It can also be used to speed up the search for keys to crypt systems such as DES, PGP etc. The main disadvantage of these algorithms is the lack of hardware support and measurement difficulties.

7. REFERENCES

[1] Alpert, C.J., Kahng, A., (1995). "Recent Developments in Netlist Partitioning: A survey*",* in *Integration: the VLSI Journal*, **19**, 1-18.

[2] Kernighan, B.W., Lin S., (1970), "An Efficient Heuristic Procedure for Partitioning Graphs", *the Bell Sys. Tech. Journal*, 291-307.

[3] Holland, J., (1975), "Adaptation in Natural and Artificial Systems", *Ann Arbor*: University of Michigan Press.

[4] Cohoon, J., Paris, W., (1987). "Genetic Placement," *IEEE Transactions on Computer-Aided Design*, **6**, 956-964.

[5] Louis, S., Rawlins, G., (1991), "Using genetic algorithms to design structures. Technical Report", *Indiana University*, Bloomington, IN 47405.

[6] Vemuri, R. M., Vemuri, R.R.,(1990), "Genetic-synthesis: Performance-driven logic synthesis using genetic evolution", *Proc. First Great Lakes Symposium on VLSI*, Kalamazoo, MI, 312-317.

[7] Shahookar, K., Mazumdar, P., (1990), "A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization", *IEEE Transactions on Computer-Aided Design*, **9**, 500-511.

[8] Goldberg, D., (1989), "Genetic Algorithms in Search, Optimization, and Machine Learning. Reading", *MA*, Addison-Wesley.

[9] Vignaux, G.A., Michalewicz, Z., (1991), "A genetic algorithm for the linear transportation problem", *IEEE Trans. SMC*, **21**(2), 445-452.

[10] Kirkpatrick, S., Gelatt, C.D., Jr, Vecchi, M.P., (1983). "Optimization by Simulated Annealing", *Science*, **220**, 671-680.

[11] Adleman, L., (1994), "Molecular computation of solutions to combinatorial problems", *Science New series*, **266**,1021-1024.

[12] Lipton, R., (1995), "DNA solution of hard computational problems", *Science*, **268**, 542-545.

[13] Adleman, L., Rothemund, P., Roweis, S., Winfree, E., (1996), "On Applying Molecular Computation to the Data Encryption Standard", *2nd DIMACS workshop on SCA based computers*, Princeton. 28-48.

[14] Lipton, R., (1996), "Speeding up Computations via Molecular Biology", *1st DIMACS workshop on SCA based computers*. Princeton, **27**, 67-74

[15] Shin, H., Kim, C., (1993), "A simple yet effective technique for Partitioning", *IEEE Trans. VLSI Systems*, 1/3.

[16] Fiducia, C.M., and Mattheyses, R.M., (1982),"A linear time heuristic for improving network partitioning", in *Proc ACM/IEEE Design Automation*, 175-181.

[17] Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N., Goodman, M., Rothemund, P., Adleman, L., (1998), "A Sticker-Based Model for SCA Computation", *Journal of Computational Biology*, **4**, 615-629.

[18] Zhang, Y. J., Shi, Y. B. , Zhou, S. Q,(2008), "Partitioning algorithm for innovation graph state estimation", *Journal of Harbin Engineering University* **29(11),** 1166-1171.

[19] Garey, M.R., Johnson, D.S., (1979), "Computers and Interactability: A Guide to the Theory of NP-Completeness", *W.H. Freeman & Company*, San Francisco.

[20] Adleman, L., (1996), "On Constructing a Molecular Computer. 1st DIMACS workshop on SCA based computers*", Princeton, In DIMACS series*, **27**, 1-21.

[21] Robertson, M., Ellington, A., (1997), "New directions in nucleic acid computing: Selected ribosome's that can implement re-write values", *3rd DIMACS workshop on DNA based computers*, 69-73.