Multiobjective VLSI Cell Placement Using Distributed Genetic Algorithm

Sadiq M. Sait, Mohammad Faheemuddin, AbdulWaheed AbdulSattar, Mahmood R. Minhas
King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
Email: {sadiq,awaheed,faheem,minhas}@ccse.kfupm.edu.sa

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 multi-objective cell placement, where the need to optimize conflicting objectives adds another level of complexity. A modified distributed parallel 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. A pseudo-diversity approach is taken, wherein 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 compare speedups. The speedup gains achieved using this parallel GA are compared with the speedup of a parallel Tabu Search strategy.

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 sheer complexity of modern circuit densities, this design stage is an inherently NP-hard problem for which conventional constructive techniques have often proved inadequate. Genetic Algorithms on the other hand have proven quite effective in reaching satisfactory layout designs, albeit with long run-times [Chan (1991), Esbensen (1992), Sait (2001)]. As such, significant efforts have been put into speeding Genetic Algorithms, the most promising of which is their parallelization.

The parallel GA described in this paper targets the optimization of width-constrained, multi-objective placement. It is a derivative of the standard distributed parallel GA, which follows an island approach, with independently evolving subpopulations and periodic exchange of solutions through migration.

The cost functions for the objectives - minimizing wire-length, delay and power dissipation - and the serial genetic algorithm used as the base for performance comparison are slight modifications of earlier work [Sait (2001)]. The serial GA follows an aggressive pseudo-diversity approach, wherein instead of preventing identical solution strings in the population, the fuzzy fitness is used as the unique attribute. No two solutions in the population are allowed to have the same fitness values. The Parallel Genetic Algorithm maintains this artificial population diversity within each processor.

2. Problem and Cost Function Modeling

In this section, we formulate our problem and the cost function used in our 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 [Sait (2001)].

2.2 Cost Functions

Now we formulate cost functions for our three said objectives.

Wirelength Cost: Interconnect wirelength of each net in the circuit is estimated and then total wirelength 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. 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.

The above rule is translated to *and-like* OWA fuzzy operator [Yager (1988)] and the membership $\mu(x)$ of a solution x in fuzzy set *GOOD solution* is given as:

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

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:

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

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

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

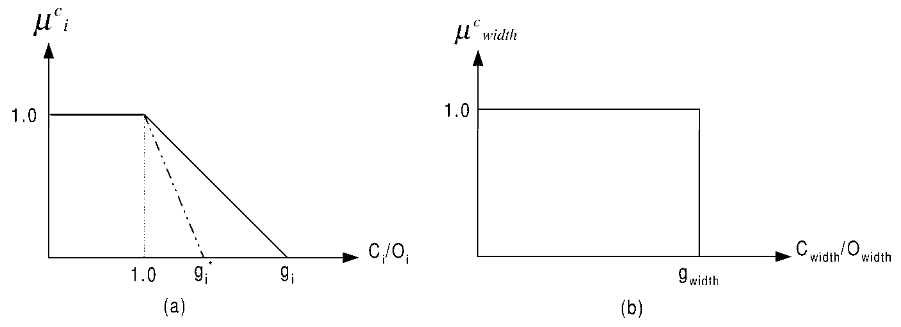


Figure 1. *Membership functions.*

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

where O_j for $j \in \{l, p, d, width\}$ are the optimal costs 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. Experimental Setup

The parallel architecture used in this work relies on distributing work load on a dedicated computing clutster with individual nodes connected via a low-latency network. Each of these nodes, which totalled eight in all is a general purpose stand-alone Pentium workstation running at 2.0GHz with 256MB memory and running the Red Hat Linux distribution. The cluster runs over a FastEthernet 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.

4. Distributed 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 process in local minima solution space. The encoding the solution and the application of genetic operators such as parent choice, crossover, mutation, and selection are taken from earlier work [Sait (2001)].

Before approaching plausible strategies of parallelization, a profiling of the serial GA is required to determine computation-intensive functions and routines. This is shown below in Table 1 where the percentage of runtime spent in the main components of the sequential GA - the fitness calculation and application of genetic operators is documented. These profiles generated for the ISCAS'89 benchmark circuits show that most time consuming for all circuits is the delay calculation and the genetic crossover. Other components of the fitness objective such as minimizing wirelength and power dissipation as well as the genetic selection operator consume less runtime.

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

Ckt	Delay (%)	Wire (%)	Power (%)	Fitness (%)	Crossover (%)	Selection (%)
s298	39.6	7.3	1.96	49.0	44.1	1.4
s386	44.8	7.0	1.70	53.4	39.3	1.3
s832	39.6	6.7	1.22	47.7	45.2	1.7
s641	78.4	2.1	0.36	81.0	14.2	0.4
s953	44.6	6.1	1.01	52.1	42.2	1.3
s1196	51.9	4.4	1.09	57.5	36.9	1.0
s1238	51.4	4.6	0.99	57.1	38.6	0.6
s1494	38.5	4.3	0.84	43.8	51.6	0.4
s1488	37.7	5.3	1.10	44.2	51.6	0.6

Given the above profile, it can be seen that the traditional model of Global Parallel GA, wherein only the fitness is distributed among processors, fails. The global parallel GA model, assumes that application of genetic operators is trivial, with most time spent in fitness calculations [Adamidis (1994), Cantu-Paz (1998)].

The distributed parallel GA as reported often in literature, is a more popular approach, wherein the population is distributed among processors. Each processor applies the genetic algorithm to its assigned subpopulation with periodic or condition-triggered migration of solutions between them [Cantu-Paz (1998), Cantu-Paz (1998b)].

In our model, which is a variant of the distributed parallel GA, the size of the initial population created by the root process increases linearly with the number of processors. This guarantees that the initial population per process stays constant even as the number of processes grow. Also, given that this initial population constructor is located at root, and controlled by our pseudo-diversity approach, no two solutions will be the same in terms of fitness function. These distinct populations have a high probability of following different evolutionary paths, and with effective migration policies can help explore a much wider search space.

Figure 2 illustrates this model in the form of a schematic diagram, while the algorithm is presented in pseudo-code in Figure 3. The initial population constructor on the master (root) processor creates the initial population which is a multiple of the number of processors. This 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

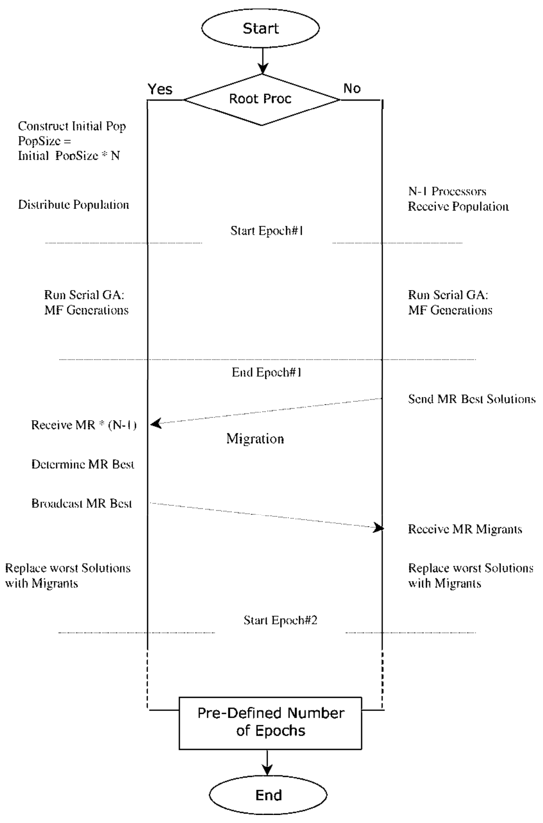

Start
Root Proc
Yes
No
Construct Initial Pop PopSize = Initial PopSize * N
Distribute Population
N-1 Processors Receive Population
Start Epoch#1
Run Serial GA: MF Generations
Run Serial GA: MF Generations
End Epoch#1
Send MR Best Solutions
Receive MR * (N-1)
Migration
Determine MR Best
Broadcast MR Best
Receive MR Migrants
Replace worst Solutions with Migrants
Replace worst Solutions with Migrants
Start Epoch#2
Pre-Defined Number of Epochs
End

Figure 2. *Schematic Diagram illustrating the working of the Distributed Search Space Parallel GA.*

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. Figure 3 shows the pseudo-code for this algorithm.

Although this model logically represents a non-hierarchical parallel architecture, physically it is implemented here as a Master-Slave model. The master processor is responsible for determining the source and destination of migrant solutions.

5. Experimental Results, Comparison, and Discussion

In this parallel GA model, the objective as mentioned earlier is to encourage each processor to potentially explore different areas of the search space. These processors communicate at intervals exchanging best solutions, thereby effectively communicating information about the search space to one another. This approach has the potential to both reduce runtime significantly, as well as reach higher quality solutions within the same number of iterations or time length. The population sizing, genetic operators and migration parameters for this multiple-population model are listed below.

- Population size of 32 per processor is always maintained regardless of number of processors. The initial population construction is done at the root processor, and as such, diversity can be enforced. The population is then distributed among all other processors.
- Regarding the genetic operators, Roulette wheel is used for parent selection, dual PMX crossover is used for offspring generation while an elitist roulette mechanism is employed

```
ALGORITHM $Distributed\_Parallel\_GA$
NOTATION
$RANK$ : 0= Root Processor 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
  FOR RANK:0
      Initial Population Constructor
      Distribute Initial Population
  ENDFOR RANK:0

    FOR RANK:ANY
  Receive Allocated Population
  ENDFOR RANK:ANY

    LOOP-A
    FOR RANK:ANY
      LOOP-B
        Serial GA on Allocation Population:
          Choice of Parents
          Crossover and Offspring Generation
          Fitness Calculation
          New Population Selection
      END LOOP-B IF [Num_Iterations >= MF]
    ENDFOR RANK:ANY

      FOR RANK:0
      Collect the MR*(N-1) 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 ($Distributed\_Parallel\_GA$)
```

Figure 3. *Structure of the Distributed Parallel GA.*

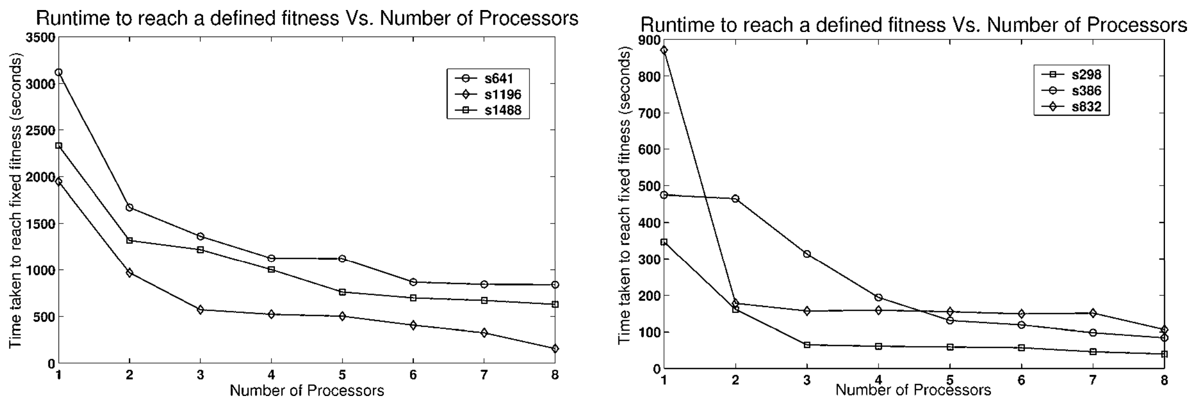


Figure 4. *The decrease in runtime to reach a pre-defined fitness objective with increasing number of processors.*

for new population selection, in which the best solution is always selected, while the other solutions are based on roulette wheel.

- A Migration frequency of 20 generations is empirically selected i.e., migration occurs after a static epoch length.
- Migration rate is only one i.e., only the best solution is sent to other processes.
- The migrant absorption policy requires the migrant to replace the worst quality solution in the processor's population.
- The run length was for a thousand epochs. With each epoch of twenty generations, each processor cumulatively runs 20000 generations.

Table 2. *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=3	P=5	P=7
s386	0.504	15	9.9	5.7	6.7
s641	0.616	793	307	390	289
s832	0.479	128	43	37	39
s953	0.511	309	136	91	108
s1196	0.484	988	327	262	205
s1488	0.482	1883	677	435	418
s1494	0.496	1405	847	638	479

Table 2 shows the amount of time taken to reach a predefined fitness objective with increasing number of processors. As seen, there is a constant decrease in runtime for all circuits with significant performance gains. The results for the smaller circuits show that the runtime levels off after approximately three to four processors, while in the case of larger circuits, the speedup continues even upto eight processors. This would be because of the less-expansive search space in the case of smaller circuits, which is effectively covered by 3-4 processors. Concurrently, with larger circuits, the increasing number of processors translates into a more expanded view of the search space and lesser runtimes to reach target fitnesses. A more clear view of this trend can be seen from the graphs shown in Figure 4.

Figure 5 shows a speedup comparison between the parallel Genetic Algorithm discussed and the parallel Tabu Search approach mentioned earlier. The figure shows that for the circuits s1488 and s1494, the parallel GA presents continuous gains in speedup, though for the latter circuit, the speedup after five processors tends to decrease. In comparison, the parallel TS approach shows minimal speedup with upto five processes, followed by an increase in runtime with additional processors. However, it should be noted that the absolute runtimes demonstrated by Tabu Search in general are significantly lesser than those for Genetic Algorithms. This is simply because Tabu Search works with a single solution and simple moves, while Genetic Algorithms work with a population and more complex crossover schemes to navigate the search space.

The speedup gains illustrated by the above parallel GA model can be compared against a parallel Tabu Search approach applied to the same VLSI problem. Tabu Search is an artificial intelligence based heuristic that works on a single solution at a time. The heuristic navigates through the search space using single-move strategies to discover new solutions and a memory component that conditionally prevents returning to previously visited areas of the search space [Sait (1999)].

The parallel Tabu Search strategy used here is reported in literature as a synchronous master-slave model with the following characteristics: P-control (each process is responsible for its search), Rigid Synchronization (all processes must communicate and synchronize at specified points), and Multi-Point Single Strategy (All processes start with different initial solution, but follow identical search strategies) [Toulouse (1996), Crainic (1997)]. This parallelization approach is similar to our Parallel GA approach, in terms of the synchronous behavior, each process conducting its own search and following identical search strategies.

Figure 5 shows a speedup comparison between the parallel Ge netic Algorithm discussed and the parallel Tabu Search approach. The figure shows that for the circuits s1488 and s1494, the parallel GA presents continuous gains in speedup, though for the latter circuit, the speedup after five processors tends to decrease. In comparison, the parallel TS approach shows minimal speedup with upto five pr ocesses, followed by an increase in runtime with additional processors. However, it should be noted that the absolute runtimes demonstrated by Tabu Search in general are significantly lesser than those for Genetic Algorithms. This is simply because Tabu Search works with a single s olution and simple moves, while Genetic Algorithms work with a population and more complex crossover schemes to navigate the search space.

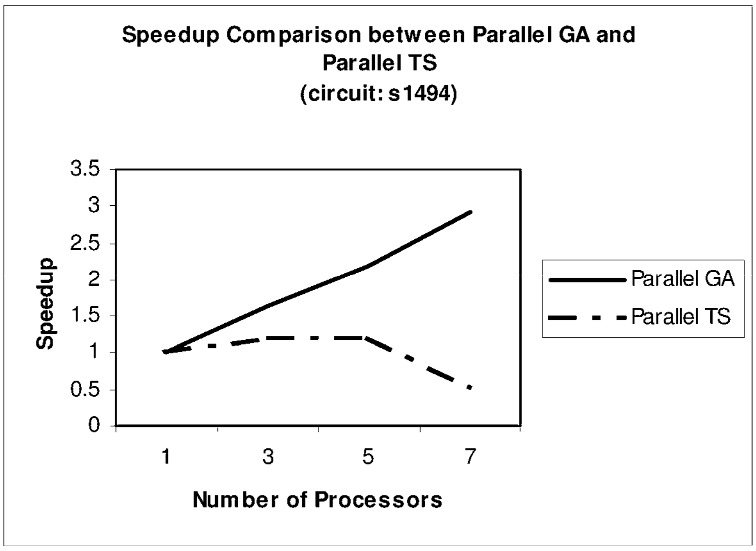
6. Future Work and Conclusion

This paper presented the application of a modified Distributed GA to a multi-objective VLSI cell placement problem. The algorithm focussed on distributing the search space among processors with an increasing initial population size instead of distributing the work load with a static population. Also, an appreciable degree of diversity is maintained by preventing solutions with the same fuzzy fitness value to exist within the population at the same time.

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 two lines. One would be to achieve a parallel GA that follows the evolutionary path of the serial GA, as close as possible with linear speedups. This would be a modification of the traditional Global Master-Slave parallel GA, wherein the computation-intensive crossover function is also distributed among the processors.

A second approach would be to seed the parallel Genetic Algorithm with solutions from low-runtime iterative heuristics such as Tabu Search. Most GA systems generate random initial



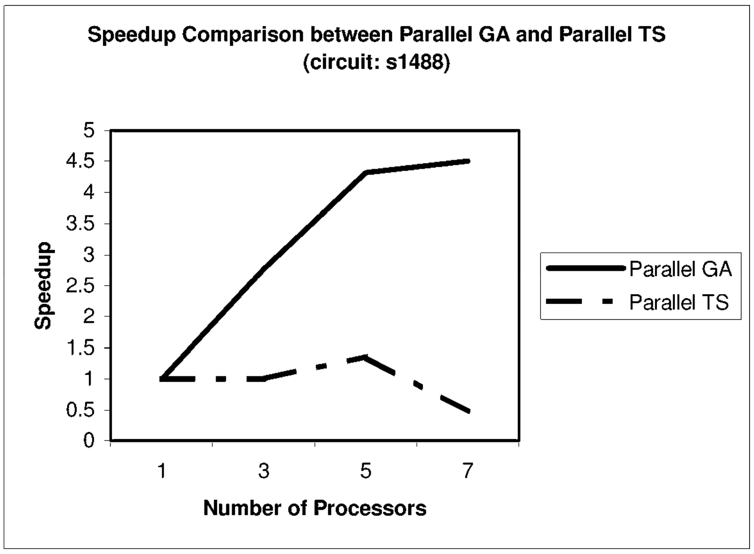


Figure 5. *Speedup comparison between Parallel GA and Parallel TS for circuits s1488 and s1494*

populations which then are used to navigate through the search space. However, including a high quality result from a low runtime heuristic into the initial population would likely accelerate the search process with the parallel GA.

Acknowledgment:

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

References

Erick Cantu-Paz. (1998). Designing efficient master-slave parallel genetic algorithms. *Genetic Programming*, 1998.

Erick Cantu-Paz. (1998b). A survey of parallel genetic algorithms. *Calculateurs Parallles, Reseaux et Systems Repartis*, 1998.

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

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

Adamidis, P. (1994). Review of genetic algorithms bibliography. *Technical Report, Aristotle University of Thessaloniki, Greece*, 1994.

Sait, S. M. and Youssef, H. (1999). *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, California, December 1999.

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

Sait, S. M., Youssef, H., El-Maleh, A. and Minhas, M. R. (2001). 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.

Toulouse, M., Crainic, T. G., and Gendreau, M. (1997). 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.

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

Crainic, T. G., Toulouse, M. and Gendreau, M. (1997). Towards a taxonomy of parallel tabu search heuristics. *INFORMS Journal of Computing*, 9(1):61–72, 1997.