An Island-Based GA Implementation for VLSI Standard-Cell Placement

Guangfa Lu and Shawki Areibi

School of Engineering
University of Guelph
Ontario, Canada, N1G 2W1
Tel: (519) 824-4120, X53819
Fax:(519) 836-0227
sareibi@uoguelph.ca

Abstract. Genetic algorithms require relatively large computation time to solve optimization problems, especially in VLSI CAD such as module placement. Therefore, island-based parallel GAs are used to speed up this procedure. The migration schemes that most researchers proposed in the past have migration near or after the demes converged [1,2]. However, for the placement of medium or large standard-cell circuits, the time required for convergence is extremely long, which makes the above migration schemes non practical. In this paper, we propose a novel migration scheme for synchronous island-based GA. Compared to the widely used ring topology that usually produces worse solutions at the beginning of the search but better solutions at later generations, the proposed migration scheme enables a parallel GA to outperform its serial version most of the time. Near linear speedup is obtained. We also implemented an asynchronous model that can achieve super-linear speedup.

1 Introduction

Physical design of VLSI circuits is a process of determining the location and connection of devices inside a chip. The main stages of the physical design phase include partitioning, placement, and routing [3]. Our work focuses on the placement of standard-cell in the physical design phase.

Given a circuit consisting of a set of modules, the standard-cell placement problem attempts to find a layout indicating the positions of the modules in parallel rows such that the total interconnection length, placement area, or some other placement metrics (eg. congestion, circuit delay) are minimized. As shown in Figure 1, a standard cell is a small block of predefined sub-circuit. Cells are restricted to having equal height, but variable width, and are placed in parallel rows that are separated by routing channels, as illustrated in Figure 2.

The VLSI standard cell placement is an NP-hard problem [4]. Various heuristic optimization techniques have been applied to this problem in the past. Genetic algorithms are proved to be able to produce high quality placement solutions for standard-cell circuits as competitive as that of other sophisticated algorithms such as Simulated Annealing and force-directed algorithms [5]. However,

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1138–1150, 2004.

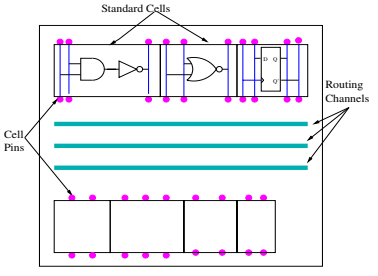

Fig. 1. Standard Cells

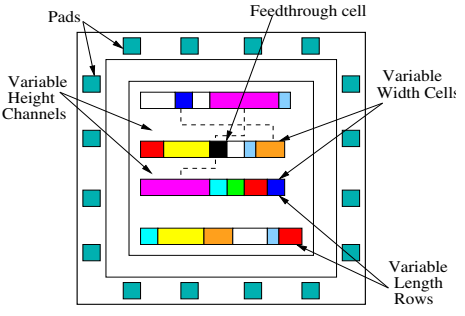

Fig. 2. Standard Cell Placement

the runtime of genetic algorithms are relatively longer than other algorithms [6, 7] and are becoming less competitive in the real world. Therefore, researchers are seeking parallel GA implementation for better performance.

Island-based GAs are coarse grain parallel models, and can be easily mapped to existing distributed parallel systems like a cluster of networked workstations. Since a fairly good speedup can be easily achieved, the Island-based GA is becoming the most popular parallel method. In this method the whole population is divided into subpopulations (also known as demes) and distributed among multiprocessors. These semi-isolated subpopulations are executed as normal genetic algorithms, except that they would swap a few strings occasionally. By introducing migration, parallel island models have often been reported to display better search quality than that of a serial GA [8,9].

Migration is very important to search quality and parallel efficiency. Usually considered to be a good migration scheme, "Delay" migration schemes are algorithms where migration occurs after the demes are near convergence or completely converged. However, in standard-cell placement, this approach is not suitable since large circuits require extremely long time to converge. We propose a practical migration scheme for placement in this paper, and describe the successful synchronous implementation. Besides the novel migration scheme, our contribution also includes an implementation of an asynchronous model for placement that can achieve super-linear speedup.

The rest of the paper is organized as follows: Section 2 presents a brief overview of previous work on evolutionary based placement algorithms and the challenge of parallel genetic algorithms for standard-cell placement. In section 3, we present a novel migration scheme to meet the challenge and describe the successful synchronous parallel GA implementation for placement. In section 4, we describe an asynchronous parallel implementation that can achieve super-linear speedup. The experimental results are presented in section 5, and the paper concludes in section 6.

2 Background

In this section, we describe a serial genetic algorithm implementation for standard-cell placement and highlight the challenge of parallelizing GAs for such a problem.

There are three primary objectives in the placement problem: minimizing chip area, achieving routable designs, and improving circuit performance. Our work in this paper mainly focuses on the objective of minimizing chip area. Since minimizing the wire-length is approximately equivalent to minimizing the total chip area for the standard-cell layout style [10], the total cost of a placement layout can be estimated by the sum of wire length over all nets:

$$\phi(x,y) = \sum_{1 \le i < j \le N} w_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{1}$$

(x_i, y_i) denotes the location of cell i; w_{ij} is a non-negative weight of the edge connecting cell i and cell j. Equation (1) can be rewritten in matrix form as:

$$\phi(x,y) = \frac{1}{2}\mathbf{x}^T\mathbf{C}\mathbf{x} + \mathbf{d}_x^T\mathbf{x} + \frac{1}{2}\mathbf{y}^T\mathbf{C}\mathbf{y} + \mathbf{d}_y^T\mathbf{y} + \mathbf{t} \tag{2}$$

Vectors $\mathbf{x}$ and $\mathbf{y}$ denote the coordinates of the N movable cells; matrix $\mathbf{C}$ is the Hessian matrix; vectors $\mathbf{d}_x^T$ and $\mathbf{d}_y^T$ and the constant term $\mathbf{t}$ result from the contributions of the fixed cells. In the placement stage, this *Semi-perimeter method* is commonly used to approximately estimate the total wire-length.

Previously, [11] has developed a standard-cell placer called *SC3* for academic research, using various optimization techniques. A steady-state genetic algorithm(Figure 3) is one of the selectable algorithms that can be used to optimize initial placement solutions. In this algorithm, a standard-cell placement solution string was represented by a set of alleles (the number of alleles equal to the number of cells). Each allele indicates the cell index, the X- coordinates and row number of the cell. Figure 4a illustrates the string encoding of the standard-cell placement given in Figure 4b. Individuals are evaluated to determine their fitness through a scoring function F, using the *Semi-perimeter method:*

$$F = \frac{1}{\sum_{i=1}^{n} HPWL_i} \tag{3}$$

where $HPWL_i$ is the estimated wire-length of net i, and n is the number of nets. Thus $\sum_{i=1}^{n} HPWL_i$ is the sum of the half perimeter of the smallest bounding rectangle for each net. In the implementation, cell overlaps are removed and row lengths are adjusted before evaluating the chromosome. Initial solutions can be constructed randomly, or by Cluster-Seed method. The GA starts its evolution by applying stochastic operators such as selection, crossover, and mutation to the individuals in each generations. The main drawback of genetic algorithms is the high computational demand, therefore we apply island-based genetic algorithms to speedup this procedure.

For island-based GAs, an appropriate migration scheme is critical. The methods of migration schemes determined by convergence are very popular. [12] proposed algorithms where migration begins only after the subpopulations are near

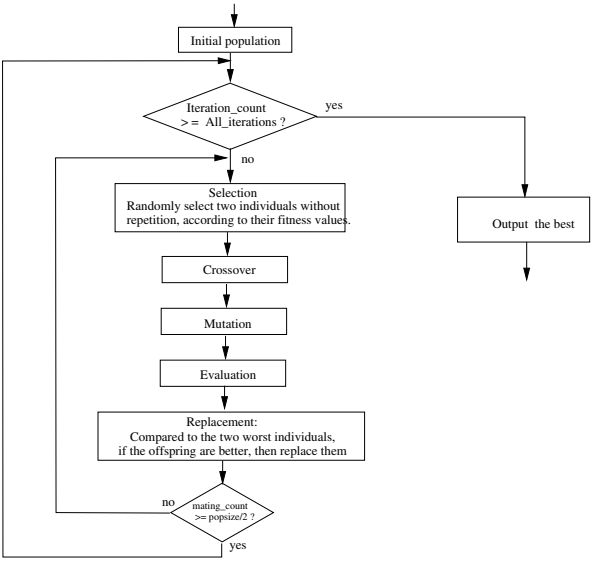

Fig. 3. A steady-state genetic algorithm for placement

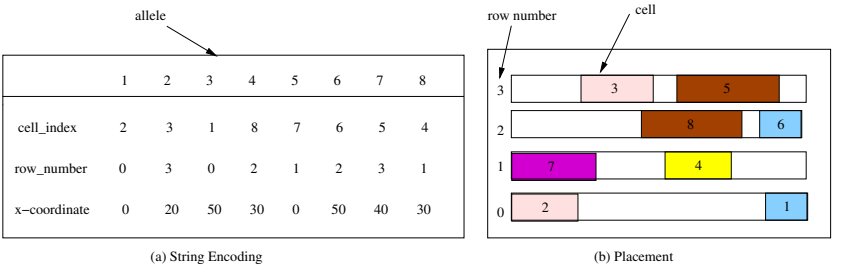

Fig. 4. String Encoding

convergence. In [1], migration occurs after the subpopulations completely converged. The study claimed that if migration is introduced too early before search converged, good original schemata in the demes may be destroyed by incoming migrants and thus the demes may lose their diversity.

However, previous studies show that large circuits require extremely long time to converge. Our experimental work shows that for small circuits convergence takes several hours even if we parallelize the GA using 7 processors. Since using convergence to trigger migration is not practical for placement, we proposed a better migration scheme, and successfully implemented a synchronous island-based GA for standard-cell placement. Besides, we also implemented an

asynchronous model that can achieve super-linear speedup. The two island-based models are described in the next two sections.

3 Synchronous Island-Based GA

In the synchronous island-based GA, the total population is divided equally into several subpopulations, and each processor is assigned a single subpopulation to form an island. As depicted in Figure 5, the sub-algorithm in each island is simply a regular serial steady-state GA plus migration.

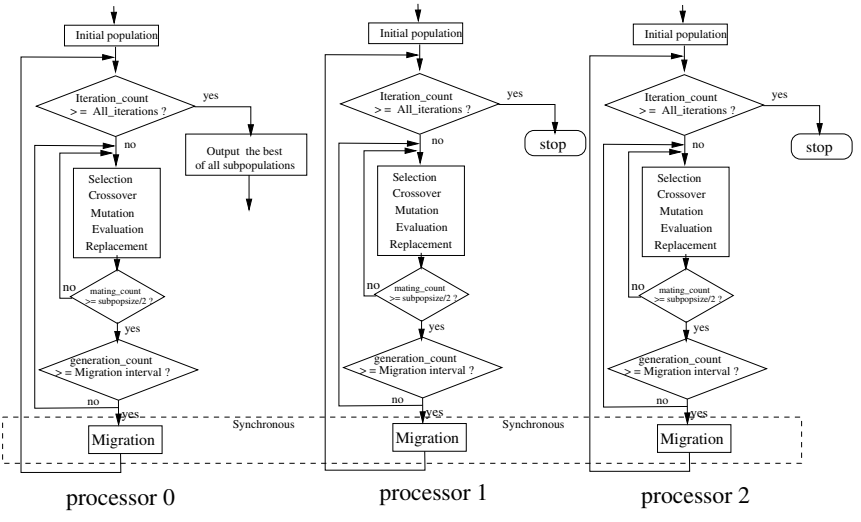

Fig. 5. A synchronous model with 3 processors

Each sub-algorithm includes an extra phase of periodicly exchange of some of their individuals. More specifically, for every certain number of generations (known as migration interval), all islands exchange a candidate of good individuals (migrants) with each other, based on the communication topology. For each island, if an incoming migrant is better than the worst existing individual, this migrant is then inserted into the subpopulation. All islands perform their migrations at the same time.

The mechanism of island models is complex, since many parameters affect the quality of search as well as efficiency of parallelism. Besides the basic GA parameters, other parameters are involved such as deme size, migrants selection method, migration interval, migration rate, communication topology, and replacement rules.

3.1 The Deme Size and the Number of Islands

For a specified population size, the deme (subpopulation) size is determined by the number of processors used. We can imagine that, the more processors we throw into the parallel system, the smaller the size of subpopulations, and the shorter the computation time. However, this is not always the case. The execution time of a parallel GA includes computation time and communication time. As more processors are used, the average available bandwidth for each processor decreases, but more communication is required for each processor. The latter grows especially fast in a low bandwidth parallel system. On the other hand, a very small subpopulation might prematurely converge and leads to poor solutions. Therefore, when we scale up a parallel GA system to pursue better performance, the parallel efficiency usually diminishes.

3.2 Migration

Migration is the process where subpopulations occasionally exchange some individuals. It is a very important parameter that determines the quality of the search as well as the efficiency of the parallelism. If the migration is low, the sub-algorithms on different small subpopulations work independently and the final result may be worse than a serial GA. If the migration is high, the parallel GA behaves very similarily to a serial GA [8,9]. There are many ingredient of migration, and they all affect the behaviors of the parallel genetic search:

- **Migrants Selection Methods:** Among the chromosomes in an island, high quality individuals are chosen to be sent out as migrants. There are several techniques that can be used to select such individuals. One way is just simply picking the best individuals. Other techniques involve selecting the best individuals probabilistically, such as Roulette-wheel selection, Stochastic universal selection, and Binary tournament selection. The higher the pressure, the faster the algorithm reaches equilibrium while sacrificing more genetic diversity of those migrants [13]. In our implementation, the outgoing migrants are from the best individuals, with the restriction that each individual being sent once. In addition, our algorithm restricts an incoming immigrant to be sent out directly but allows such a mechanism for children. In the 'to-all' communication scheme, only the best unsent individual of a deme can be sent out as migrant in each migration.
- **Migration Interval:** Migration intervals specify the migration plan. Most island-based GAs start migrations after subpopulations are converged. But for standard-cell placement, we use a fixed epoch in our synchronous model, and start migrations at very early generations. The reasons are: (i) Without extra computation, it's simple; (ii) A string encodes a placement solution of a VLSI chip, for medium or large circuits, the time of waiting for convergence could be too long in practice. We also believe that if we use such a fixed migration interval and migration scheme we could make distributed demes behave like a single panmictic population (serial GA) that would outperform the serial GA in all generations.

– **Communication Topologies:** The communication topology determines the pattern of communications, or the connections between subpopulations. The ring topology is especially popular, since it has the longest diameter and demes are more isolated than other topologies. With a ring topology, the parallel search may find a better solution than that of a serial GA in late generations. However, because the demes are very much isolated, the small deme size usually produces solutions worse than that of a serial GA with a single population in early generations. To solve this problem, we use a complete graph topology called 'to-all' topology. With such a topology, all demes are fully connected. During migration, each island contributes the best individual to the global migrant pool. Since the connection between demes is very strong in this topology, the frequency of migration must be controlled in some way and the number of migrants must be carefully set, such that the strength of migration will not be excessive. As shown in Figure 6, only a portion of the best individuals in the pool are merged back into the subpopulations. With this migration scheme, a global superfit individual can move across the overall population in every migration interval, while some degree of diversity between the subpopulations are maintained. Therefore, the parallel GA may have the chance to outperform its serial version in a very short time. Also, due to the high connectivity among islands, this synchronous model would search in a trajectory similar to a serial GA.

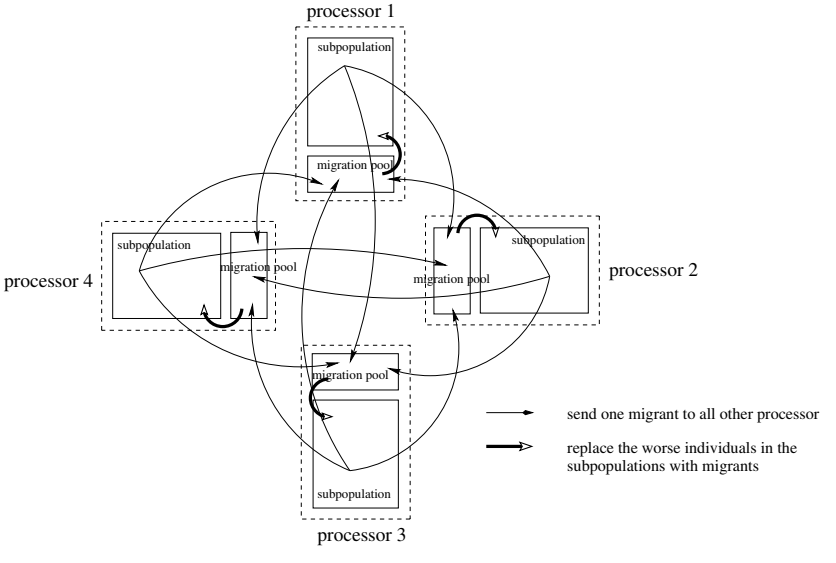

Fig. 6. The migration scheme of the 'to-all' topology

– **Migration Rate:** In our algorithm, if the ring topology is used, the migration rate is defined as the percentage of the individuals in a subpopulation

that need to be sent out as migrants; If the 'to-all' topology is used, the migration rate specifies the percentage of migrants in the migration pool that will spread to all subpopulations. A very high migration rate often leads to two problems: first, high communication costs; secondly, superfit migrants may take over the receiving subpopulation too soon. Search quality usually benefits from an appropriate migration rate.

- **Migrants Replacement Rule:** After an island receives some migrants, it replaces the worst individuals with these incoming migrants. For the 'to-all' migration scheme, although each island sends out only one migrant, it may receive more than one incoming individual to replace the worst members in its subpopulation.

The main disadvantage of synchronous models is that the migration of all islands occurs at the same time and the faster processors have to wait for the slower ones.

4 Asynchronous Island-Based GA

Since our goal aims at performance related issues, an asynchronous island-based GA was also implemented for standard-cell placement. In an asynchronous model, islands are free to evolve and to migrate their individuals without waiting for others. As illustrated in Figure 7, the workstations involved are configured as a master/slaves style, where each slave is a processor holding a subpopulation and the only acting master is a central controller.

4.1 The Master

The master brings two functions to the asynchronous model:

1. **Migration controller:** Controls the overall migration activities and dynamic load balancing including routing migrants, communication topology, migration plan, and dynamic load balancing. Figure 7 illustrates that the slaves report their current states to the master periodically and the master sends control signals to command these slaves.
2. **High speed communication router:** In our current implementation, there is no direct communication between slaves, and migration is performed through the master. Since the master keeps listening to the communication channels, it is able to respond to the requests from slaves almost immediately. After the master receives migrants sent from a slave, it buffers these messages, and controls the migration scheme. Incorporating a master in the system eliminates virtually all delay that might occur between processors. The master can easily choose a communication topology. Since our 'to-all' topology is synchronization in nature, we use an alternative topology in the form of a ring.

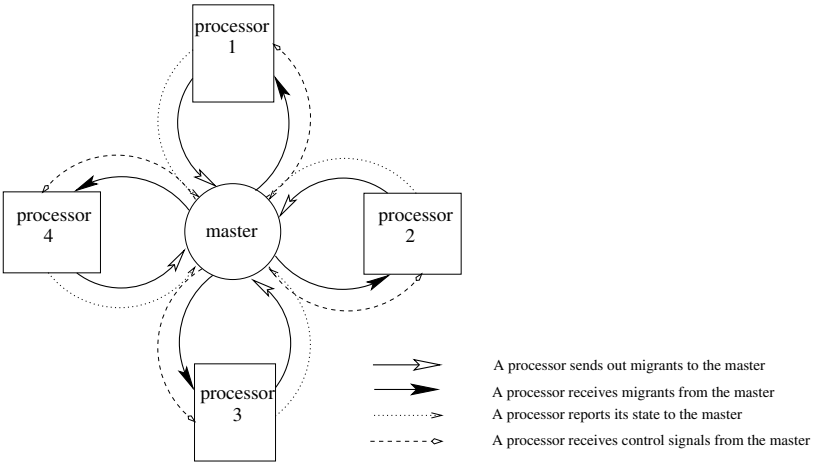

Fig. 7. The migration scheme of the asynchronous model with ring topology

4.2 Dynamic Load Balancing

Since subpopulations are free to evolve, some processors might finish their task much earlier than others. This leads to two problems: (i) First, if a migration occurs between an evolving island and an idle island, the migration becomes meaningless; (ii) Secondly, the parallel system still has to wait for the slowest processor to finish its work. Therefore, a load balancing mechanism is needed to remove some load from slower processors and transfer it to faster processors.

In Matthem's algorithm [14], if a processor completes its work early, it becomes idle and the algorithm reassigns some work to this processor from the other processors. However, we suspect a possible problem in this algorithm: the migration is now between two portions of the same subpopulation instead of two different demes. Therefore, we use different strategies for dynamic load balancing. Instead of taking part of the work from a slower processor and replacing the whole subpopulation of faster processor, we periodically detect the evolution speed on varied demes, and dynamically change the sizes of the subpopulations to match the speed.

4.3 Distributed Random Migration Plans

In the natural world, *islands* are independent and semi-isolated. Since subpopulations are free to evolve in our asynchronous model, we have a chance to give more independence to these subpopulations by (1)applying distributed random migration plan created in each island; and (2)using random number of migrants (with some control) for each migration. With such distributed random plans, the subpopulations are allowed to evolve totally asynchronously. The strategy attempts to simulate the fact that the number of migrants in each migration is not constant in the real world.

5 Experimental Results

All of the test runs were conducted on a cluster of networked Sun workstations, using the MPI (Message Passing Interface) v.1.2.5 based on the 'C' programming language. Many parameters affect an island-based parallel GA, thus we carried out a large number of runs. Table 1 presents the benchmarks that were used in our experiments. Two to seven islands were applied for the parallel algorithms in the tests. Each test was conducted 20 times and the average values were recorded.

Table 1. Benchmarks used as test cases

Circuit	ckt1_52	fract	struct	prim2	avq_large
Nodes	64	149	1952	3121	25178
Nets	55	147	1920	3136	25384
Pins	278	876	10814	22371	165374

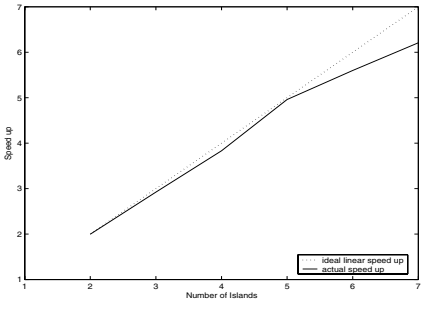

Fig. 8. Synchronous model: 'to-all' topology

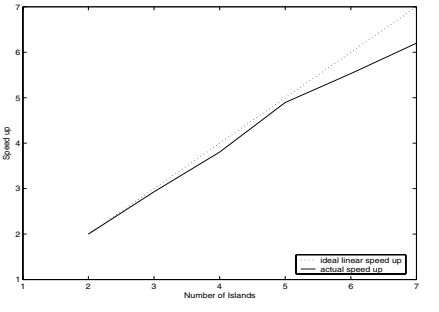

Fig. 9. Synchronous model: ring topology

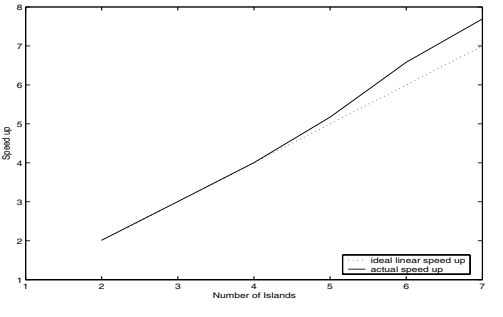

Fig. 10. Asynchronous model: ring topology

5.1 Performance: Speedup Versus Processors

The performance of the two synchronous migration schemes is shown in Figure 8 and Figure 9 respectively. The parallel systems were configured from two to seven processors, and circuit *struct* was tested. The experiment shows both the ring migration scheme and the 'to-all' migration scheme obtained near ideal linear speed up. The latter is even slightly better than the former. As expected, when the number of processors involved increased, the communication costs for synchronization grew, and a decrease of the parallel efficiency was observed.

With asynchronous migration and dynamic load balancing, the asynchronous model obtained super linear speedup (as seen in Figure 10). For example, 7 islands achieved a speedup of 7.6. [15] explains why super linear speedup is possible for asynchronous models, and [16] acquired similar results with their implementation.

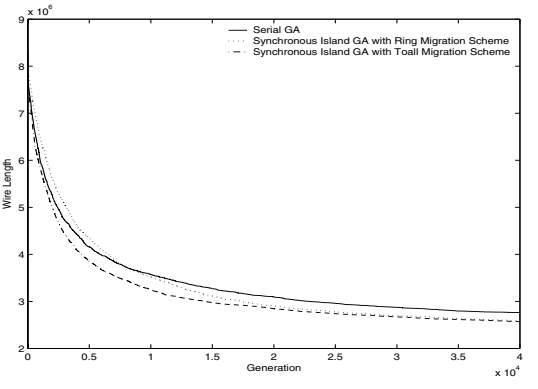

Fig. 11. Placement quality: Serial GA, Ring PGA, and 'to-all' PGA

5.2 Solution Quality

In order to compare the search quality of the proposed 'to-all' migration scheme with that of the ring migration scheme, we plotted their search for 40000 generations as seen in Figure 11 (a serial GA was also plotted as baseline). In this experiment, seven processors were used. To assure the fairness, we maintain the same basic GA parameters for all of them.

Like most implementations in the literature, the parallel search with ring topology outperforms the serial GA in the medium (> 8000 generations, in this case) and later generations. However, no implementation with ring topology (including ours) is able to surpass its serial version in early generations. Usually, to obtain a better solution than a serial GA, the ring topology parallel GA requires a large enough number of generations. The proposed 'to-all' migration scheme, on the other hand, works better than a serial GA almost all the time!

The advantage of the new migration scheme over ring topology is that, depending on how much time available, we can run the parallel GA for 200 generations or 20000 generations, and still get better solution than that of a serial GA. It is important to note that, other experiments show that the execution time for both migration schemes are pretty much the same.

We also compared the search quality of a synchronous island-based GA (with proposed 'to-all' migration scheme) to a serial based implementation in the early generations for different circuits, as shown in Table 2. The algorithms were run for only 100 generations, and 7 processors were used for the PGA (parallel GA). Each test was conducted 20 times, and we recorded the average values. The results show that the parallel implementation achieved speedups from 6.06 to 6.38 with 7 processors, and the proposed migration scheme yielded better solutions than the serial GA from very early generations (the 100th generation).

Table 2. Search results in the 100th generation (7 processors were used for PGA)

Benchmark	**Execution time (SGA)**	**Execution time (PGA)**	**Speedup**
ckt1_52.yal	2.8844803	0.4522641	6.38
struct.yal	475.7	76.139	6.25
prim2.yal	1180.28	188.41	6.264
avq_large.yal	94778.5	15631.4	6.063
Benchmark	**Wire length (SGA)**	**Wire length (PGA)**	**Improvement**
ckt1_52.yal	20177.6	19591.5	2.9%
struct.yal	7.5124×10^6	7.4958×10^6	0.22%
prim2.yal	6.800114×10^7	6.79839×10^7	0.024%
avq_large.yal	1.57018×10^9	1.568035×10^9	0.1366%

6 Conclusion

This paper proposed a practical migration scheme for synchronous island-based GA. Experimental results show that our algorithms can achieve near linear or even super-linear speedup. In addition the proposed migration scheme yields better quality of solutions than a serial GA from an early stage of evolution. Although we carried out some experiments with a large number of generations, we are especially interested in the search quality of initial phase of the search. Running a circuit placement with a GA for 10000 generations is usually not practical. This emphasizes the advantage of the proposed migration scheme in the application of standard-cell placement. The implemented asynchronous model was able to achieve super-linear speedup as expected.

References

1. Braun, H.: On solving travelling salesman problems by genetic algorithm. In Schwefel, H.P., Manner, R., eds.: Parallel Problem Solving from Nature, Springer-Verlag (1990) 129–133
2. Horii, H., Kunifuji, S., Matsuzawa, T.: Asynchronous island parallel GA using multiform subpopulations. Lecture Notes in Computer Science **1585** (1999) 122–129
3. Sherwani, N.A.: Algorithms for VLSI Physical Design Automation. Kluwer Academic Publishers, Intel Corporation, Hillsboro, OR, USA (1998)
4. Chang, H., Cooks, L., Hunt, M.: Surviving the SOC Revolution. Kluwer Academic Publishers, London (1999)
5. Kling, R.M., Banerjee, P.: Optimization by simulated evolution with applications to standard cell placement. In: Conference proceedings on 27th ACM/IEEE design automation conference, ACM Press (1990) 20–25
6. Kilng, R., Banerjee, P.: Esp: Placement by simulated evolution. IEEE Trans. on Computer Aided Design, 8(3) (1989) 245–256
7. Shahookar, K., Mazumder, P.: A genetic approach to standard cell placement using metagenetic parameter optimization. IEEE Trans. on CAD , vol. 9 (1990) 500–511
8. Whitley, D., Rana, S.B., Heckendorn, R.B.: Island model genetic algorithms and linearly separable problems. In: Evolutionary Computing, AISB Workshop. (1997) 109–125
9. Cantu-Paz, E., Goldberg, D.E.: Efficient parallel genetic algorithms: Theory and practice (2000)
10. Shahookar, K., Mazumder, P.: VLSI Cell Placement Techniques. ACM Computing Surveys **23** (1991) 143–220
11. Areibi, S.: Memetic algorithms for vlsi physical design: Implementation issues. In Hart, W., Krasnogor, N., Smith, J., eds.: Second Workshop on Memetic Algorithms (2nd WOMA). (July 2001) 140–145
12. Munetomo, M., Takai, Y., Sato, Y.: An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. In: Proceedings of the Fifth International Conference on Genetic Algorithms. (1993) 649
13. Cantu-Paz, E.: Migration policies, selection pressure, and parallel evolutionary algorithms. In Brave, S., Wu, A.S., eds.: Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference, Orlando, Florida, USA (1999) 65–73
14. Mcmahon, M.T., Watson, L.T.: A distributed genetic algorithm with migration for the design of composite laminate structures (1998)
15. Alba, E., Troya, J.M.: An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands. In: IPPS/SPDP Workshops. (1999) 248–256
16. Andre, D., Koza, J.R.: A parallel implementation of genetic programming that achieves super-linear performance. In Arabnia, H.R., ed.: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. Volume III., Sunnyvale, CSREA (1996) 1163–1174