ICOT Technical Memorandom: TM-1262

TM-1262

An Asynchronous Fine-Grained Parallel Genetic Algorithm

by
T. Maruyama, A. Konagaya
& K. Konishi (NEC)

May, 1993

An Asynchronous Fine-Grained Parallel Genetic Algorithm

Tsutomu Maruyama, Akihiko Konagaya and Koichi Konishi
C&C System Research Laboratories, NEC Corporation
4-1-1 Miyazaki Miyamae-ku Kawasaki Kanagawa 216 JAPAN
maruyama@csl.cl.nec.co.jp

Abstract

In this paper, we propose a new asynchronous fine-grained parallel genetic algorithm. This algorithm has two important features, (i) all synchronizations during recombination and selection are eliminated by applying the operators to an individual in a processor and the youngest ancestors in other processors, and (ii) the values of the youngest ancestors are broadcast among processors in order to maintain wide-ranging comparison in the selection. These features make it possible to exploit a great deal of parallelism without increasing the probability of falling into locally optimal solutions.

Experiments on graph partitioning show good results. The quality of the partitions found by this algorithm is almost the same as a sequential genetic algorithm with the same genetic operators. The speedup of this algorithm is about 14~18 times using 15 processors. This high performance gain makes it possible to find good solutions much faster than heuristic algorithms for graph partitioning.

1. INTRODUCTION

Genetic algorithms show especially good performance when they are combined with heuristic algorithms, because heuristic algorithms climb up local hills efficiently and genetic algorithms lead the entire search towards optimum solutions. There are two important issues in order to achieve high performance gain by parallel processing of these hybrid approaches.

One is to eliminate all synchronizations in genetic algorithms. The time required for applying heuristic algorithms to each individual can vary considerably. Therefore, if there are synchronizations in the genetic algorithm, the speedup by parallel processing is limited by the slowest individual.

Another important issue is to maintain as good quality of solutions as sequential genetic algorithms. Local selection methods are widely used in asynchronous and fine-grained parallel genetic algorithms[1–4] in order to reduce overheads for parallel processing such as synchronization and inter-processor communications. In these methods, an individual in a processor is compared with its neighborhood, and replaced by one of the neighborhood or its parent. However, the small size of neighborhood in the local selection methods increases the probability of falling into locally optimal solutions.

In this paper, we propose a new asynchronous fine-grained parallel genetic algorithm. This algorithm has two important features. First, all synchronization in recombination and selection are eliminated by applying the operators to an individual in a processor and the youngest ancestors in other processors. Second, the selection operator compares an individual in a processor with all the youngest ancestors in other processors using the values of the ancestors broadcasted among processors. These features make it possible to exploit a great deal of parallelism without increasing the probability of falling into locally optimal solutions.

This paper is organized as follows. Section 2 briefly describes genetic algorithms. Section 3 presents the asynchronous fine-grained parallel genetic algorithm. Section 4 describes the graph partitioning problem and heuristic algorithms. Section 5 introduces a parallel algorithm for graph partitioning which combines the proposed algorithm and the mincut algorithm. Section 6 presents the experimental results. In Section 7, conclusions are given.

2. GENETIC ALGORITHMS

Genetic algorithms are stochastic search algorithms based on ideas from genetic and evolutionary theory Genetic algorithms simulate the survival of the fittest of a number of individuals, which represent points in a search space. A step of genetic algorithm, called a generation, consists of the following operators: recombination (crossover), mutation and selection. The recombination operator produces two descendants by exchanging a part of two individuals, called ancestors. This operator causes a long jump in the search space. To the contrary, the mutation operator changes only a small part of an individual. This operator searches close to a point in the search space. The selection operator simulates the survival of the fittest principal. This operator first calculates the relative fitness of all individuals. Then, several worse individuals are discarded and the same number of better individuals are replicated according to their relative fitness.

3. AN ASYNCHRONOUS FINE-GRAINED PARALLEL GENETIC ALGORITHM

In this section, we introduce a new asynchronous fine-grained parallel genetic algorithm. In order to achieve high performance, the algorithm has to satisfy the following two requirements.

- Exploiting maximum parallelism
- Good quality of solutions

The proposed algorithm has two important features. First, the recombination and selection operators are applied to an individual in a processor and to the youngest ancestors in other processors. This feature makes it possible to exploit maximum parallelism by eliminating all synchronization in the algorithm. Second, all individuals are compared with all the youngest ancestors in other processors using the values broadcasted among them. This wide-ranging comparison reduces the probability of falling into locally optimal solutions.

3.1. Asynchronous Genetic Operators

There are two kinds of synchronizations in genetic algorithms. One is in recombination, and the other is in selection. In our algorithm, these synchronizations are eliminated by using buffers to store a few ancestors and by applying the recombination and selection operators to an individual in a processor and the youngest ancestors in other processors.

Each processor has

- an individual,
- buffers which store a few ancestors and
- values of the youngest ancestors in other processors

The recombination operator produces only one offspring by replacing a part of an individual with one of the youngest ancestors in other processors. The selection operator first calculates the relative fitness of an individual in the processor and all the youngest ancestors in other processors using the values broadcasted among processors. If the individual can survive (the first selection of the individual N-1 in figure 1), the value of the individual is broadcast to all processors, and then it is copied into a free buffer. This copy becomes the youngest ancestor of the processor. Ancestors which are not being accessed by other processors are discarded. If the individual can not survive (the first selection of the individual 1 in figure 1), one of the youngest ancestors in another processor is selected according to the relative fitness and copied into the processor as a new individual of the processor. Before starting the remote copy, two values are broadcast from the processor. One is the decrement for the value of the selected ancestor. The value is decremented in order to prevent the ancestor from being copied repeatedly. The other is the value of the new individual. A bad value is assigned to the individual, because this individual is a copy of an ancestor and should not be copied again.

3.2. Critical Sections

In this algorithm, there are three critical sections. One is the update of the buffers. If the buffers are being accessed during the update, an incorrect ancestor may be copied by other processors. In the implementation, the buffers are locked during the update, because only several pointers are changed during the update.

The other two are the race conditions in the update of the value of the selected ancestor. First, suppose that PEi selects the youngest ancestor in PEj as a new individual of PEi. If PEj just finishes its selection and updates the value of the youngest ancestor during the selection in PEi, PEi might decrement the value incorrectly when it finishes its selection. In the implementation, the broadcast of the decrement by PEi is canceled, if the value of the youngest ancestors is updated during its selection.

Second, suppose that PEi selects the youngest ancestor in PEj as the new individual of PEi. If PEk starts its selection before the value of the selected ancestor is decremented, PEk may select the same ancestor as its new individual. This excessive copy of ancestors increases the probability of falling into locally optimal solutions. However, the computation time for the selection is very small, and can be ignored if the broadcasting on the parallel machines is fast enough, which is the common case in current parallel machines.

3.3. Scalability

In this algorithm, the number of the individuals is limited by the speed of the broadcasting. However, by using a multi-population strategy [5, 6], we can exploit more parallelism. In this strategy, each population has a number of individuals in it, and exchange a few individuals between populations from time to time. In [5, 6], it was reported that this strategy can find as good solutions as the single population strategy.

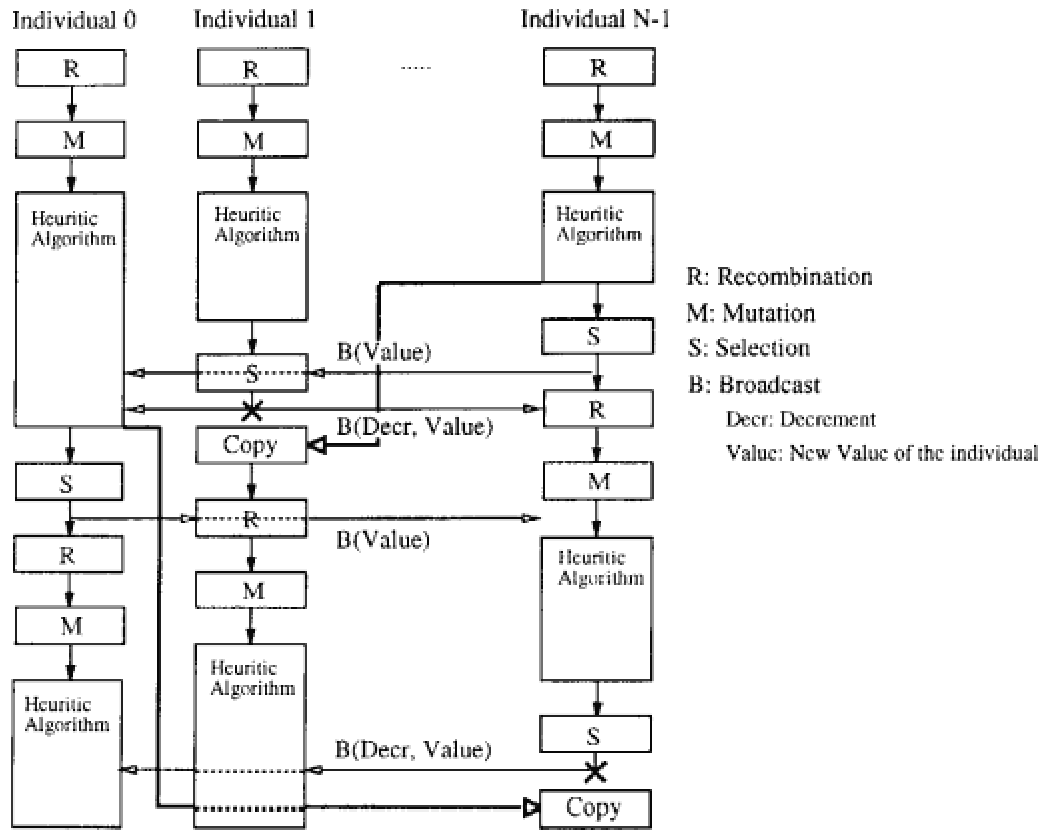

Figure 1. Asynchronous Genetic Operators

4. THE GRAPH PARTITIONING PROBLEM AND HEURISTIC ALGORITHMS

The graph partitioning problem is to decompose a graph into two subgraphs so that the size of each subgraph is bounded and the cut size (the number of edges that connect to cells in both subgraphs) is minimized. Graph partitioning is an important process in many areas of computers (e.g. design of circuits, mapping). The recursive decomposition of the problems can dramatically reduce the complexity of the problems.

Mincut-based partitioning[7, 8] is a technique to decompose a graph into two subgraphs efficiently. This technique, however, often falls into locally optimal solutions. In order to get a good partition, we have to apply the algorithm repeatedly varying the initial partition.

The hierarchical clustering with mincut exchange methods (HCME)[9] is a heuristic algorithm which aims to produce a good initial partition for the mincut algorithm. First, the HCME recursively merges clusters until only two clusters are left, starting from clusters which have only one cell in them. In this step, cluster pairs with high connectivity are selected as clusters to be merged. Then, it recursively applies the 1-opt mincut algorithm (see [7, 8]) on clusters, decomposing the clusters in the inverse order. The HCME can find better partitions than the 1-opt mincut algorithm, though it requires 5~7 times of computation time.

5. AN ASYNCHRONOUS FINE-GRAINED PARALLEL GENETIC ALGORITHM FOR GRAPH PARTITIONING

In this section, we introduce a parallel graph partitioning algorithm which combines the 1-opt mincut algorithm and the asynchronous fine-grained parallel genetic algorithm.

Two kinds of recombination operators are used in this algorithm. Both recombination operators replace a part of the partition with a part of a different partition first. In general, this replacement destroys the constraint of partition sizes. Then, one operator moves the cells with the highest gain (the reduction in the number of edges when the cell is moved) between subgraphs, until the constraint of partition sizes is satisfied. On the other hand, another operator moves the cells at random.

The mutation operator simply exchanges a certain amount of cells between two subgraphs. In general, the mutation operators changes only a small part of individuals. However, small changes of the individuals are immediately restored by the mincut algorithm. In order to exit from local optimum by mutation operators, a greater part of individuals have to be changed. In the implementation, one of the recombination operators or the mutation operator is applied in each generation.

6. RESULTS

We partitioned six examples from real gate array designs on a Sequent Symmetry which is a shared memory parallel machine with 15 processors. The results described below are the average of 100 runs. In all evaluations, the time for loading the graph data is not included. The parameters of the genetic algorithm are not tuned for each graph partitioning. In all experiments, the recombination operators replace about 40% of a partition with those of different partition. The ratio of applying the recombination and mutation operators is fixed to 7:3. The ratio of applying two kinds of recombination operators is fixed to 1:1.

We evaluated the performance of four algorithms below. All algorithms are implemented in C.

1. APGA – the asynchronous fine-grained parallel algorithm (the proposed algorithm)
2. SPGA – a synchronous parallel genetic algorithm
3. The 1-opt mincut algorithm
4. The HCME algorithm

In the synchronous parallel genetic algorithm, all the genetic operators are applied after applying the mincut algorithm to each partition. Therefore, the quality of solutions found by this algorithm is almost the same as a sequential genetic algorithm with the same operators.

Table 1
Cut size after the same number of the mincut algorithm is applied (average of 100 runs)

		APGA		SPGA	
Examples	#cell	Average	Best(Probability)	Average	Best(Probability)
test1	249	36.0	36(1.00)	36.0	36(1.00)
test2	615	71.0	71(1.00)	71.0	71(1.00)
test3	1861	254.0	250(0.04)	254.8	251(0.18)
test4	1869	243.4	241(0.29)	243.9	241(0.24)
test5	3096	302.9	292(0.01)	304.1	292(0.01)
test6	3373	186.2	186(0.86)	186.2	186(0.84)

Quality of Partitions

Table 1 compares the cut size by the two genetic algorithms after the same number of the mincut algorithm have been applied (960 times in total = 64 generations in average). The cut sizes found by the proposed algorithm (APGA) are almost the same as the synchronous algorithm (SPGA). This means that the asynchronous operators in the proposed algorithm do not degrade the quality of solutions.

Convergence Speed

Figure 2 and 3 show how the cut size improves as the searches progress. The horizontal axes of the figures show the average of the generation of the individuals. The asynchronous genetic algorithm (APGA) converges as fast as the synchronous genetic algorithm (SPGA). This means that the asynchronous operators don't make the convergence speed slower at all. Figure 4 and 5 compare the search speed of the genetic algorithms and the heuristic algorithms. The search by the asynchronous algorithm converges to good partitions within the time for 20 runs of the mincut algorithm. This is much faster than other algorithms. Because of this fast convergence, the proposed algorithm shows the best results from the beginning to the end of the search.

Parallelism

Figure 6 compares the search speed of the proposed algorithm (APGA) and the synchronous parallel algorithm (SPGA) with a sequential genetic algorithm. The speedup of the proposed algorithm by parallel processing is about 14 ~18 times compared with the sequential algorithm, and about 2 times compared with the synchronous algorithm. This very good performance comes from the increase of the total cache size by using 15 processors, and the difference of the number of the mincut applied for each solution. Figure 7 shows the time for applying the mincut algorithm on each individual. The horizontal axis is the cut size after the mincut algorithm is applied, and the vertical axis is the average execution time of the mincut algorithm (normalized by the average time from initial partition). The execution time becomes shorter as partitions are improved. Therefore, within the same amount of time, the mincut algorithm is applied to better partitions more often, and better partitions become much better as the search progresses. By repeating this process, the proposed asynchronous algorithm can find a good solution very quickly.

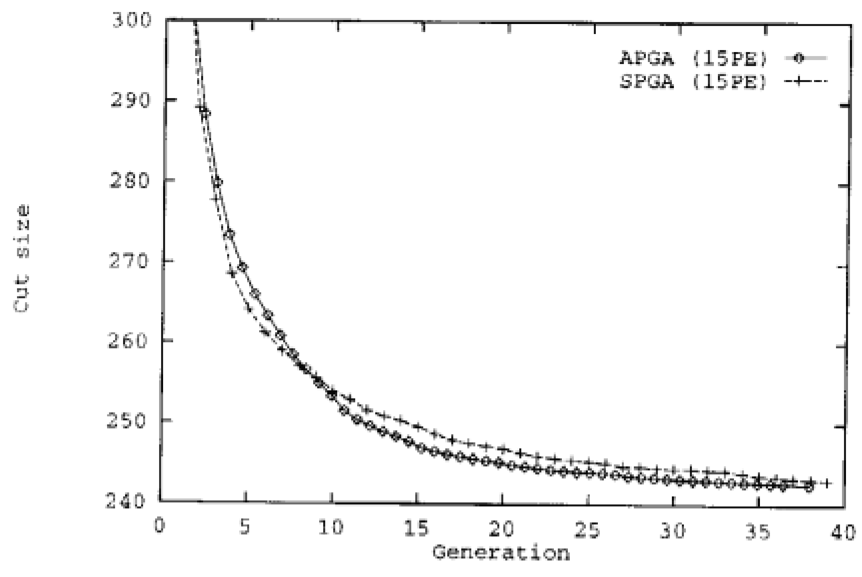

Figure 2. Cut size (test4)

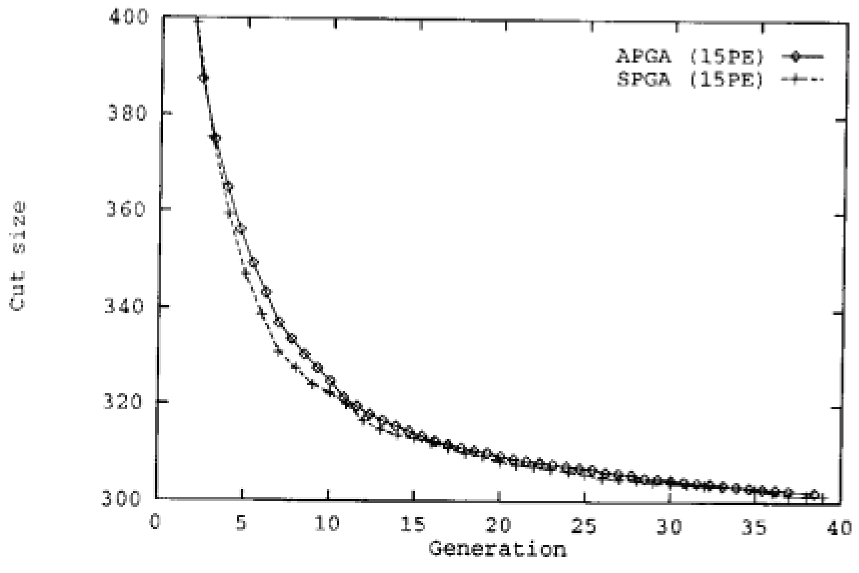

Figure 3. Cut size (test5)

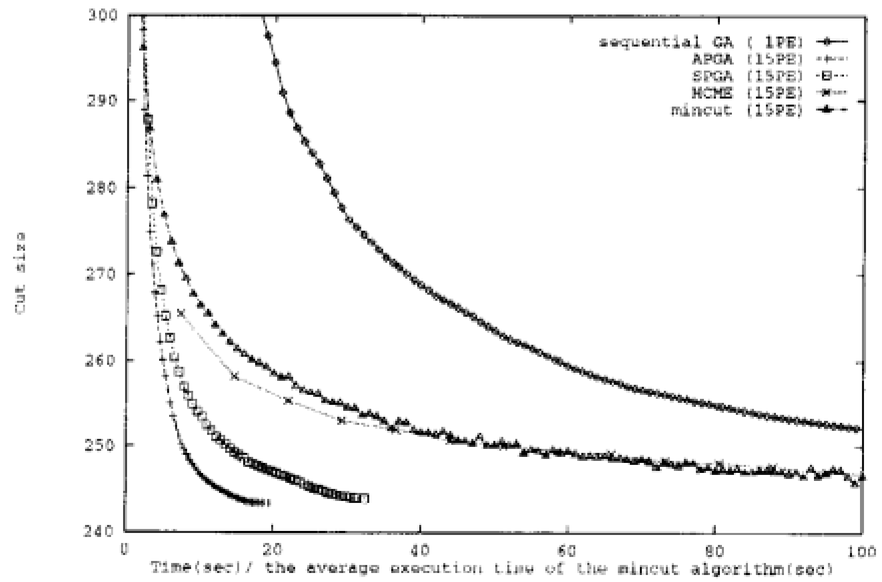

Figure 4. Cut size (test4)

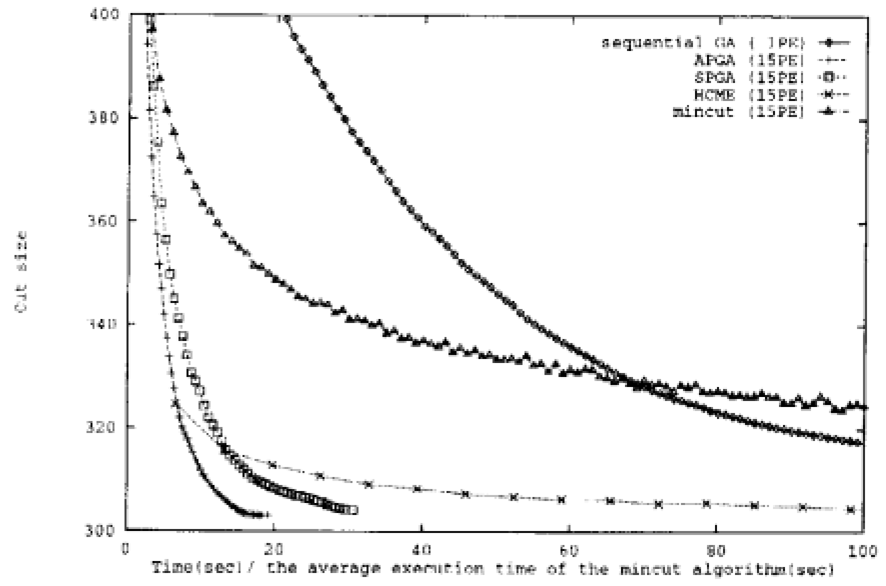

Figure 5. Cut size (test5)

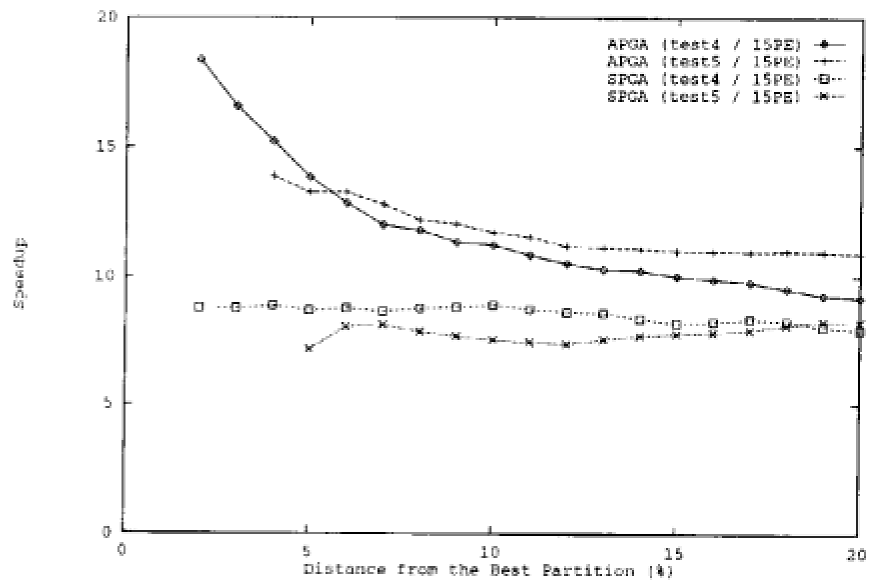

Figure 6. Speedup by parallel processing (test4 and test5)

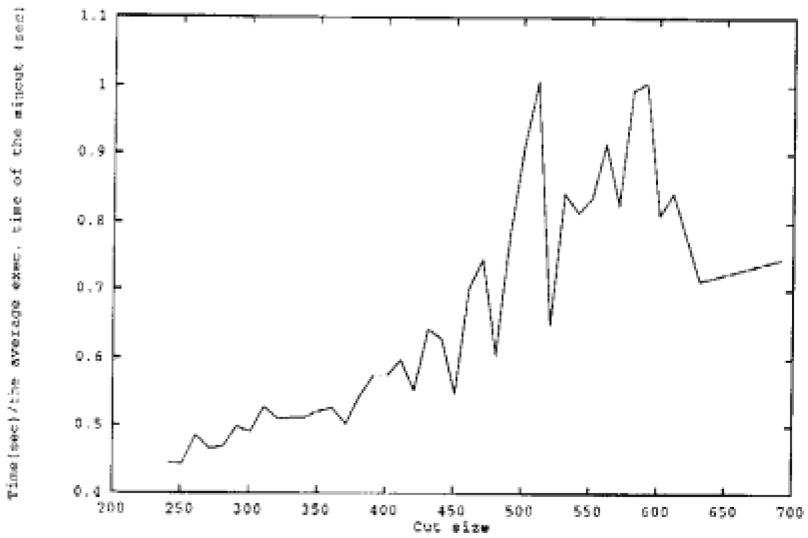

Figure 7. The average time required for applying the mincut algorithms in test4

7. CONCLUSIONS

In this paper, we have proposed a new asynchronous fine-grained parallel genetic algorithm. This algorithm has two important features. First, all synchronization during recombination and selection are eliminated in order to exploit maximum parallelism. Second, each individual is compared with all other individuals in the selection using the values broadcast among processors in order to reduce the probability of falling into locally optimal solutions.

Experiments on graph partitioning show good results. The quality of the partitions found by this algorithm is almost the same as a sequential genetic algorithm with the same genetic operators. The speedup of this algorithm is about 14~18 times using 15 processors. This high performance gain makes it possible to find good solutions much faster than heuristic algorithms.

The scalability of this algorithm is limited by the speed of the broadcasting. In order to exploit more parallelism, we can combine the algorithm with the multi-population strategies. Then, the algorithm achieves high performance in each population.

ACKNOWLEDGEMENT

This material is based upon work supported by the Fifth Generation Computer Project. We would like to thank Dr. Takashi Chikayama for giving us a chance to develop parallel genetic algorithms as a part of the evaluation of the concurrent object-oriented language A'UM 90[10].

REFERENCES

1 M. Gorges-Schleuter, "ASPARAGAS An Asynchronous Parallel Genetic Optimization Strategy", Proc. of Intl. Conf. on Genetic Algorithm, 1989, pp 422-427.

2 P. Spiessen and B. Manderick, "A Massively Parallel Genetic Algorithm - implementation and First Analysis", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 279-286.

3 G. von Laszewski, "Intelligent Structural Operators for the k-way Graph Partitioning Problem", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 45-52.

4 R. J. Collins and D. R. Jefferson, "Selection in Massively Parallel Genetic Algorithms", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 249-256.

5 H. Muhlenbein, M. Schomish and J. Born, "The Parallel Genetic Algorithm as Function Optimizer", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 271-278.

6 J. P. Cohoon, W. N. Martin and D. S. Richards, "A Multi-population Genetic Algorithm for Solving the K-Partitioning Problem on Hyper-cubes", Proc. of ICGA'91, pp 244-248.

7 B. W .Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell Systems Technical Journal, 1970, pp.291-307.

8 C. M. Fiduccia and R. M. Mattheyses "A linear Time Heuristics for Improving Network Partitions", ACM/IEEE design Automation Conf., 1982, pp.175-181.

9 M. Edahiro and T. Yoshimura, "New Placement and Global Routing Algorithms for Standard Cell Layouts", Proc. of 27th DAC, 1990, pp 642-645.

10 K. Konishi, T. Maruyama, A. Konagaya, K. Yoshida and T. Chikayama, "Implementing Streams on Parallel Machines with Distributed Memory", Proc. of Intl. Conf. on FGCS'92.