Parallel Genetic Algorithm for Finite-State Machine Synthesis from Input/Output Sequences

Shisanu Tongchim
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
`g41stc@cp.eng.chula.ac.th`

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
`prabhas@chula.ac.th`

Abstract

This work investigates the use of a parallel implementation to reduce the processing time required by Genetic Algorithm (GA). The concept of a coarse-grained model for parallelization is used to distribute the task on a cluster of workstations. The problem chosen to examine the parallel implementation is the finite-state machine synthesis from input/output sequences. The performance of the parallel algorithm was measured by recording the wall-clock time. The reduction of the execution time from the use of multiple processors was calculated in terms of relative time. The results show that the achieved relative time is linear and in some case is greater than linear while the solution quality is maintained.

1 INTRODUCTION

Genetic Algorithm (GA) was used to solve the problem of finite-state machine (FSM) synthesis from input/output sequences (Chongstitvatana, 1999). The main idea was to synthesize the target FSM from partial input/output sequences, not from a behavioral description. The result of synthesis was the FSM which produced the correct output sequences according to the given input/output sequences. Several circuits were chosen to examine the FSM synthesis.

In the previous work, the synthesizer used a small population and a large number of generations. The diversity was maintained in the selection process to prevent premature convergence. The selection algorithm was slightly adapted from (Winston, 1992). However, this selection scheme is very time-consuming since the running time is $O(n^2 \log n)$;[*] where n is the number of individuals. In addition, GA required a large number of generations to synthesize a large circuit. Due to the cost of the selection method which required sorting and a large number of generations, GA took several hours, sometime days to find the target FSM.

Since GA evaluates a population of candidate solutions in each iteration which can be viewed as a parallel process, it is easy to distribute the task of serial GA among the multiple processors. To date many proposals on parallel implementations in various architectures have been examined, as was shown by (Cantú-Paz, 1998).

The present study introduces a parallel implementation to reduce the processing time of the previous work (Chongstitvatana, 1999). The goal is to use the benefit of multiple processors by using a coarse-grained model for parallelization. Two circuits were chosen to compare the parallel performance. The first circuit is *reversible 8-counter* which represents a large circuit. The second circuit is *0101 detector* which represents a small circuit.

The remaining sections are organized as follows: The next section is a description of the previous work. Section 3 shows the parallel solution. Section 4 presents the research findings. Finally, section 5 provides the conclusions of this work.

2 PREVIOUS WORK

The idea of the FSM synthesis used in the previous work is depicted in figure 1. By observing the partial input/output sequences of the target FSM, the synthesizer eventually evolves into the target FSM.

In the prior work in FSM synthesis by GA (Manovit, 1998), the results were classified into two categories; complete solution and incomplete solution. A com-

[*]This was revised after the article was published.

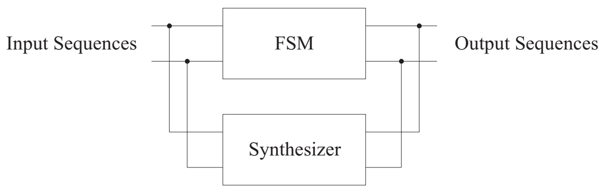

Figure 1: FSM Synthesis

plete solution performs correctly for all possible input/output sequences, whereas an incomplete solution operates correctly for input/output sequences used in the evolution process. In order to increase the percentage of complete solutions, the work (Chongstitvatana, 1999) proposed the use of multiple sequences in the fitness evaluation to improve the correctness of the results.

As mentioned earlier, the evolution process used a small population which was not suitable for dividing the population among the processors. Therefore, this work used the fixed population size on each processor. The population size on each processor was not determined by dividing the total population size by the number of processors used, as usually found in a conventional coarse-grained model. There is a related work which reported the comparison between the distributed population and the fixed population on each processor, called scaled parallel GA (Corcoran, 1994).

3 PARALLEL GENETIC ALGORITHM

In the first stage of the implementation, the population size was set at 400, as was used in the previous work (Chongstitvatana, 1999). The execution time is reduced due to the probabilistic advantage in the larger population size. In the second stage of the implementation, we reduced the population size from 400 to 200 and 100. The decrease of the population size helped to further reduce the execution time.

The experiment was implemented on a dedicated cluster of PC workstations with 350 MHz Pentium II processors, each with 32 Mb of RAM, and running Linux as an operating system. These machines were connected via 10 Mbps ethernet cabling. The program used in (Chongstitvatana, 1999) was extended to run under a clustered computer by using MPI as a message passing library.

All tests were set up to perform 10 runs. The migration between subpopulations was synchronized, and the top 5 of individuals from each subpopulation were exchanged. The migration interval was set to 10 generations. The topology used in the migration was a ring topology. Some selected individuals were exchanged between neighbors in the ring.

The crossover rate was 50% and the mutation rate was 25%. After creating the new population by crossover and mutation, the migrants were added to the new population. The remaining population was created by the reproduction.

The parameters of the circuits used in the experiment were similar to the previous work (Chongstitvatana, 1999). The number of input/output sequences was 100 for the experiment. This ensured that all obtained solutions were correct.

4 RESULTS AND DISCUSSION

To make a comparison between the serial algorithm and parallel algorithm, the quality of the solutions must be compared along with the parallel performance. Due to the fact that all obtained solutions are correct, the correctness of the solutions cannot be used as the quality of the solutions. The number of runs yielding solutions was used to compare the quality of the solutions instead. Table 1 shows the number of runs yielding solutions of the reversible 8-counter circuit. Table 2 shows the number of runs yielding solutions of the 0101 detector circuit. In the serial algorithm, the number of runs yielding solutions decreases as the size of the population reduces. In the parallel algorithm, the number of runs yielding solutions in the 0101 detector synthesis is not impacted by the reduction of the population size, whereas the number of runs yielding solutions in the reversible 8-counter synthesis decreases as the size of the population reduces.

The average execution time of the runs yielding solutions is shown in two graphs. The reversible 8-counter synthesis time is depicted in figure 2, and the 0101 detector synthesis time is illustrated in figure 3. The execution time of the parallel algorithm decreases as the number of processors increases.

The graph of the number of generations and the execution time of the runs yielding solutions for the reversible 8-counter synthesis is illustrated in figure 4. The graph of the number of generations and the execution time of the runs yielding solutions for the 0101 detector synthesis is shown in figure 5. The execution time increases linearly as the number of generations increases. In addition, the execution time reduces dramatically as the population size decreases.

The widely used performance evaluation of the paral-

Table 1: The number of runs yielding solutions (from 10 runs) : reversible 8-counter

Number of Processors	Population Size on Each Processor		
	400	200	100
1	8	6	1
4	10	10	7
6	10	9	5
8	10	10	2
10	10	10	6

Table 2: The number of runs yielding solutions (from 10 runs) : 0101 detector

Number of Processors	Population Size on Each Processor		
	400	200	100
1	10	9	9
4	10	10	10
6	10	10	10
8	10	10	10
10	10	10	10

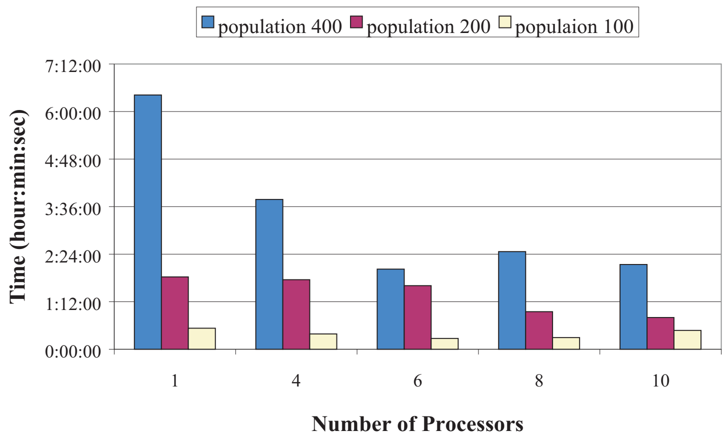

Figure 2: Time spent for synthesizing reversible 8-counter

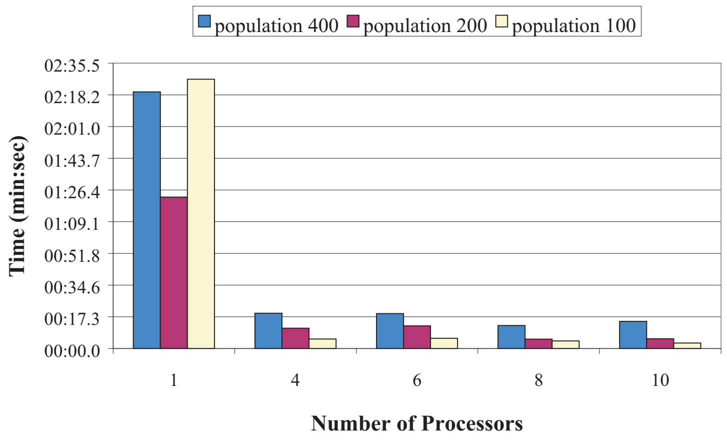

Figure 3: Time spent for synthesizing 0101 detector

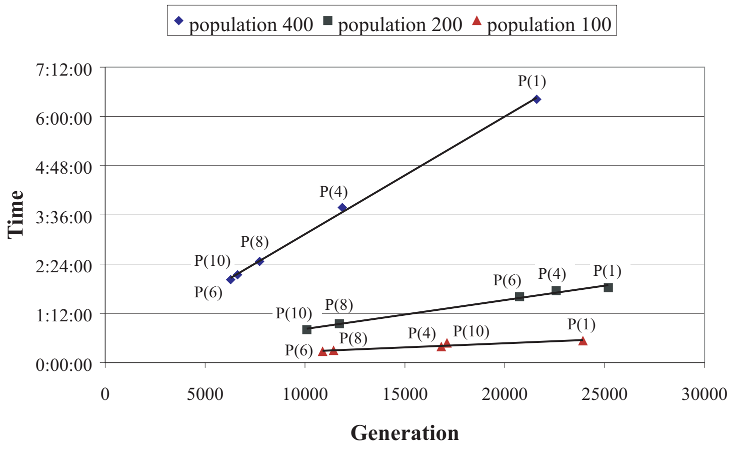

Figure 4: Time-Generation : reversible 8-counter ($P(n)$; n is the number of processors)

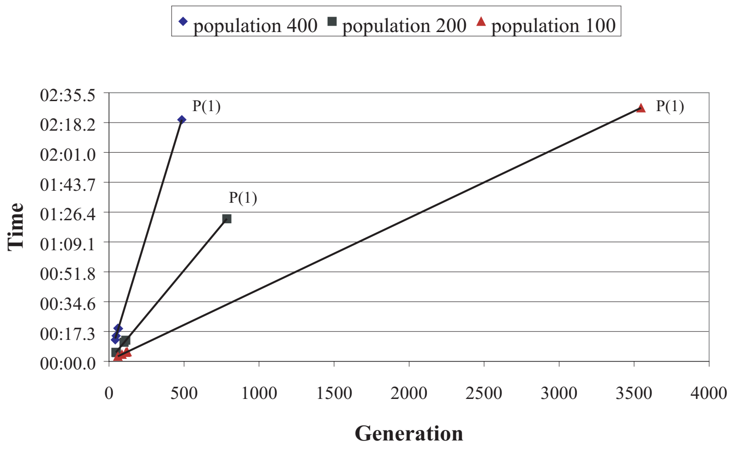

Figure 5: Time-Generation : 0101 detector ($P(n)$; n is the number of processors)

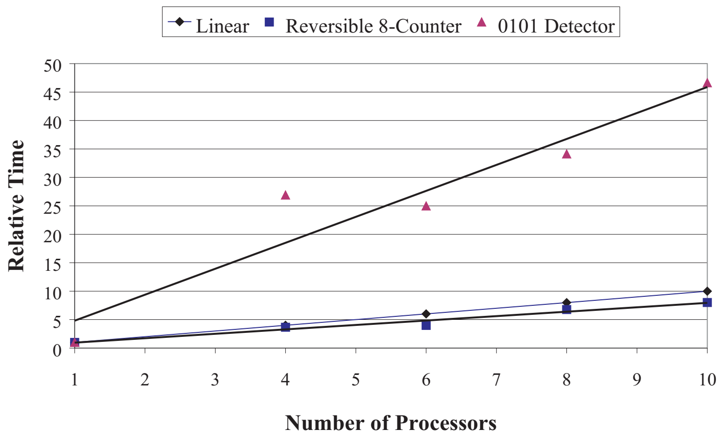

Figure 6: Relative Time

lel algorithm is the parallel speedup. However, there are some controversies about the speedup measurement in the coarse-grained model (Cantú-Paz, 1999). Many studies have claimed that superlinear speedup has been achieved. Some cases of superlinear speedup have been reported without explicitly comparing the quality of the solutions. Another source of superlinear speedup is that the parallel algorithm do less work than the serial algorithm (Punch, 1998).

In (Hart, 1996), the work avoided using the speedup measurement. The study claimed that the randomization in the GA process makes the distinction between the process of the serial implementation and the process of the parallel implementation. The execution time was used to compare the performance between two algorithms.

In this present study, we defined the performance comparison between the serial algorithm and parallel algorithm in terms of the relative time. The relative time is the ratio of the serial execution time to the parallel execution time. The serial algorithm and parallel algorithm must give the same quality of the solutions. This differs from the speedup calculation which two algorithms must do the same amount of work (Gustafson, 1990).

In comparing the performance, we want to measure the time used to find solutions of the same quality. The execution time is largely determined by the population size. We therefore chose the smallest population size that yielded solutions of the same quality.

The relative time is shown in figure 6. The relative time of the reversible 8-counter synthesis is compared between the parallel time with the population size of 200 and the serial time with the population size of 400. The relative time of the 0101 detector synthesis is calculated from the parallel time with the population size of 100 compared with the serial execution time with population size of 400. From the graph, the relative time of the reversible 8-counter synthesis is close to the number of processors used while the relative time of the 0101 detector synthesis is greater than the number of processors used.

5 CONCLUSIONS

This paper presents an empirical study of parallel genetic algorithm for solving the FSM synthesis. Since the size of the population is small, the population size in each processor is fixed, rather than dividing the total population size with the number of processors used.

The results show that the use of parallel processing increases the chance of finding solutions. Moreover, the parallel algorithm can help to reduce the population size in each process. Due to the method of selection, the reduction of the population size results in the significant gain in the execution time.

To avoid a controversy about the speedup calculation, we defined the ratio of the serial time and parallel time as the relative time. The serial algorithm and parallel algorithm must give the same solution quality, rather than executing the equal amount of work. The 0101 detector synthesis acquires the higher relative time compared with the relative time of the reversible 8-counter because the smaller population size can be used in the 0101 detector synthesis.

References

E. Cantú-Paz (1998). A survey of parallel genetic algorithms. *Calculateurs Parallèles, Reseaux et Systems Repartis* Vol. 10 No. 2. : 141-171.

E. Cantú-Paz (1999). *Designing Efficient and Accurate Parallel Genetic Algorithms.* PhD Thesis, University of Illinois at Urbana-Champaign.

P. Chongstitvatana, and C. Aporntewan (1999). Improving correctness of finite-state machine synthesis from multiple partial input/output sequences. In *Proceedings of the 1st NASA/DoD Workshop of Evolvable Hardware*, 262-266.

A. L. Corcoran, and R. L. Wainwright (1994). A parallel island model genetic algorithm for the multiprocessor scheduling problem. In *Proceedings of the 1994 ACM/SIGAPP Symposium on Applied Computing*, 483-487.

J. Gustafson (1990). Fixed Time, Tiered Memory, and Superlinear Speedup. In *Proceedings of the Fifth Distributed Memory Computing Conference (DMCC5).*

W. E. Hart, S. Baden, R. K. Belew, and S. Kohn (1996). Analysis of the numerical effects of parallelism on a parallel genetic algorithm. In *Proceedings of the 10th International Parallel Processing Symposium*, 606-612.

C. Manovit, C. Aporntewan, and P. Chongstitvatana (1998). Synthesis of synchronous sequential logic circuits from partial input/output sequence. In *Proceedings of International Conference on Evolvable Systems*, 98-105.

B. Punch (1998). How effective are multiple populations in genetic programming. In *Proceedings of the Third Annual Conference in Genetic Programming*, 308-313.

P. H. Winston (1992). *Artificial Intelligence*. MA: Addison-Wesley.