Nearest Neighbor Migration in Parallel Genetic Programming for Automatic Robot Programming

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 presents a study of parallelization of genetic programming for automatically creating a robot control program in a mobile robot navigation problem. A nearest neighbor migration topology is proposed to reduce the communication time. This study compares the performance both in terms of the solution quality and the gain in execution time. The timing analysis is investigated to give insight into the behavior of parallel implementations. The results show that the parallel algorithm with asynchronous migration using 10 processors is 32 times faster than the serial algorithm.

Keywords : Parallel Genetic Programming, Mobile Robot Navigation Problem

1 Introduction

In the past few years, evolutionary techniques have been successfully applied to automatic design of the robot control program. These techniques have been accepted as promising methods to eliminate the difficulties of the human design for programming the complex behaviors of robots. From the difference in evolutionary procedure, these techniques can be classified into various methods, e.g. genetic algorithm (GA), genetic programming (GP), evolutionary strategies (ES) and evolutionary programs (EP).

In the previous work [1], GP was used to perform automatically generate mobile robot control programs. Despite the success of using GP in automatic robot programming, the programs generated by GP were found to lack robustness. The programs may fail from a change in the working environment. Hence, the use of multiple environments in the evolution process was proposed in order to improve the robustness of the robot programs. The obvious drawback of this method is that it uses a lot of computation time.

The limitation of the processing power from a single processor and the availability of low-cost multiprocessor system [3] has led to the investigation of parallelization of the GP process on a clustered computer. Our previous work [2] proposed the parallel implementations that reduced the processing time by using a coarse-grained model. However, the previous work showed that the reduction of the execution time was limited when using a large number of processors. The source of this limitation was the problem in the communication function that we used.

In this present study, we extend the previous work in an attempt to improve the reduction of the execution time. By changing the communication topology to a loosely connected topology, the ring topology, the communication is limited to occur between the neighboring nodes. This helps to reduce the communication overhead and improve the parallel performance. In addition, the behavior of the parallel program is observed in order to make the performance analysis.

The remaining sections are organized as follows: The next section provides a brief introduction about GP, the mobile robot navigation problem and parallel GP. Section 3 presents the experimental results and discussion. Finally, section 4 provides the conclusions of this work.

2 Background

2.1 Genetic Programming

Genetic programming (GP) works with a group of candidate solutions which are randomly generated at the beginning of the algorithm. These candidate solutions are computer programs. By simulating natural evolution, GP works as an iterative procedure which is referred to as a generation. Each solution is evaluated with the objective function to determine the quality of each solution, called the fitness value. The principle of the evolution is that the solution with the high fitness value has more chance to be selected and produce offspring.

After the evaluation is performed, some individuals are selected with the probability depending on their fitness values. Then, a set of genetic operators, i.e. crossover and mutation, transforms the selected individuals into the new population of candidate solutions. The algorithm replaces the old population with the new population and repeats the whole process with the new population. This continues until the fitness value indicates that the goal is achieved or repetition limit is reached.

2.2 Mobile Robot Navigation Problem

The problem is to control an autonomous mobile robot from a starting point to a target point in the simulated environment. The mobile robot has a round shape with the ability to move forward, turn left and turn right. The robot has sensors for detecting the collision with the obstacle and indicating whether the robot is nearer to the target compared to its previous position. The size of the simulated environment is 600×400 units. The environment is filled with obstacles which have several geometrical shapes (see figure 1).

In the previous work [1], GP was used to generate the robot control program. The aim of the work was to generate *robust* control programs. The use of multiple training environments was proposed in order to improve the robustness of the robot programs. In the evolution process, each individual was evaluated under many environments that were different from the original one. The findings showed that the robustness of the robot programs is improved as the number of environments is increased. Despite the success of improving the robustness by such a method, a substantial processing time was required to evaluate the fitness of the population of the robot programs.

2.3 Parallel Genetic Programming

As the idea of GP was developed from the GA foundation, the model of parallel GP was also adopted from the development of parallel GA. Several proposals on parallel implementations of GA in various designs have been extensively investigated. In the article [4], Cantú-Paz presented the classifications and descriptions of the most representative studies in the field of parallel genetic algorithm. This survey concentrated on a coarse-grained model since much research has focused their attention on this model.

In a general coarse-grained parallelization, the population is divided into a few large subpopulations and these subpopulations are maintained by different processors. When the algorithm starts, all processors create their own random subpopulations with different random seeds. After each processing node evaluates the fitness value, some selected individuals are exchanged via a migration operator. Generally speaking, each processor runs serial GA with the smaller population and some selected individuals migrate among the processors.

The migration can be implemented as synchronous and asynchronous. In synchronous migration, all nodes proceed at their own rates and synchronize when the migration occurs. The problem of synchronous migration is that it can cause uneven work loads among processors due to the different rate of evolution. In asynchronous migration, the migration occurs without relating to the state of all processors. Asynchronous migration can reduce the wait time required for all processors.

The earlier work of parallel GP using a coarse-grained model was implemented on a network of transputers by Koza and Andre [5]. The problem of symbolic regression of the Boolean even-5-parity function was used to make a comparison of the computational effort with several migration rates. Their result showed that the parallel speedup was greater than linear.

In our previous work [6], a parallel implementation of GP for solving the mobile robot navigation problem was conducted. The model of parallel GP was a coarse-grained model. All processing nodes used the same genetic parameters and training environments. The benefit of using multiple processors was measured in terms of parallel speedup. The results showed that the achieved speedup is close to linear while maintaining the solution quality.

From the experience in the parallel GP implementation, we found that the solution quality can be increased by using different environments in each processing node. This means that the number of environments on each node can be reduced while achieving the same level of the solution quality. Accordingly, the processing time required by the GP process will be further reduced. The later work [2] used this concept to improve the parallel performance. Both synchronous and asynchronous migrations were conducted. The results showed that asynchronous migration has a slightly better performance but the performance degrades dramatically as the number of processors increases.

In the work [2], the migration was carried out as follows: each node broadcast its subpopulation to all other nodes by `MPI_Bcast` function, this was repeated for every node. The top 5% of individuals from each subpopulation were exchanged during the migration. From the analysis of the communication operations, the broadcast function took substantial time to finish the communication at a large number of processors.

In this work, we further examine the parallel implementation by using another topology. The goal is to reduce the communication time and improve the parallel performance. The change in migration topology may effect the solution quality. Therefore, the quality of solutions are also considered as an important aspect.

3 Experiment

3.1 Experimental Design

In order to reduce the communication time, the migration is limited to occur between the neighboring nodes. By using the ring topology, migrants are sent in one direction. Asynchronous and synchronous migration are conducted as in the previous work [2].

The population size on each processor is determined by dividing the total population size by the number of processors. Due to the probabilistic na-

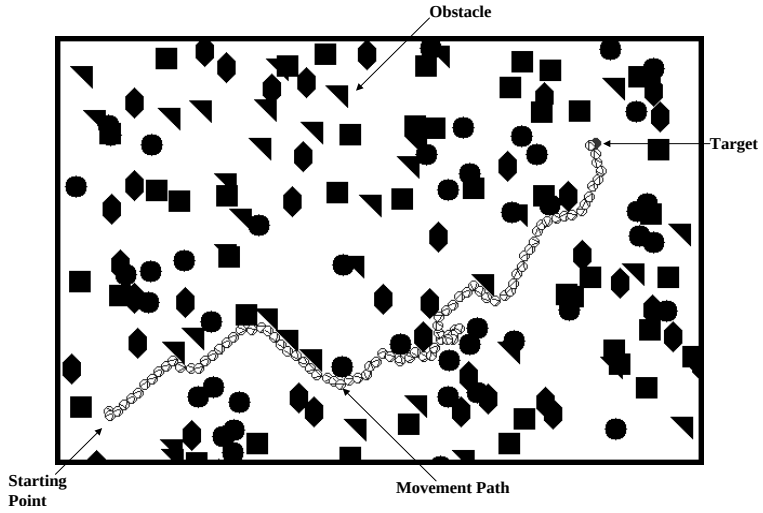

Figure 1: Simulated Environment of the Mobile Robot Navigation Problem

Table 1: Experimental Parameters

	Number of Processors				
	1	2	4	6	10
Population size*	6000	3000	1500	1000	600
Environments*	8	7	4	3	2
Migration interval**	NA	100	50	34	20

* per node
** generation

ture of GP, all results are averaged from 20 independent runs. As stated before, the use of different environments in each processor can improve the solution quality. Therefore, the number of environments can be reduced to give the same level of the solution quality (see table 1). The rest of the parameters are similar to the parameters that were used in [2].

The parallel algorithm is implemented on a dedicated cluster of PC workstations with 350 MHz Pentium II processors, each with 64 Mb of RAM, and running Linux as an operating system. These machines are connected via 10 Mbps ethernet cabling. For the programming environment, MPI is selected as a message passing library.

3.2 Experimental Results

3.2.1 Robustness

The robustness is defined as the quality of the solutions generated by GP. The robustness indicates the ability of the robot programs in the new environments that differ from the set of training environments. In the robustness measurement, the robot program is evaluated under the testing environments varying in the level of difficulty. The difficulty of each testing environment is defined as the percentage of disturbance (D).

$$D = \frac{N_m}{N_o} \times 100 \quad (1)$$

where N_m is the number of obstacles that are moved, N_o is total number of obstacles

The robustness is calculated by varying the percentage of disturbance from 0-100% and using 1000 testing environments. The robustness graph is depicted in figure 2. Most of the parallel experiments obtain higher robustness than the serial algorithm, while the robustness of the robot programs generated by using synchronous migration at 4 processors and asynchronous migration at 2 processors is close to the robustness from the serial GA.

3.2.2 Relative Time

In the previous work, the parallel speedup was used to measure the gain in execution time from the benefit of using multiple processors. The speedup is the ratio of the serial execution time to the parallel execution time. This definition relies on the assumption that both algorithms must do the same amount of work [7]. However, the use of parallel speedup has led to some controversies in a coarse-grained model [8, 9]. Many studies have claimed that the superlinear speedup [1] has been achieved. The source of this superlinear speedup is that the parallel algorithm performs less work than the serial algorithm. Thus, the superlinear speedup is not truly achieved.

In this study, we want to avoid this disputation. The ratio of the serial execution time and parallel execution time is defined as the relative time. This calculation is based on the comparison of the solution quality. The serial algorithm and parallel algorithm must give the same quality of solutions.

[1] Speedup is greater than the number of processors used.

The relative time is illustrated in figure 3. The graph shows a substantial increase of the relative time as a function of the number of processors used. The relative time is greater than the number of processors used. The reason is that the total amount of work in the parallel algorithm is less than the serial algorithm. The reduction of work is caused by the divided populations and the smaller number of environments in each node.

As compared with the relative time in the previous study (figure 4), the change in migration topology helps to further increase the relative time, especially for a large number of processors. In addition, the improvement of the asynchronous implementation over the synchronous implementation in the present study is better than the previous work.

3.3 Performance Analysis

Figure 5 illustrates the time spent in the communication section and computation section of the implementations. The percentage of communication slightly increases as the number of processors increases. Asynchronous migration helps to reduce the communication overhead. It is noted that when comparing asynchronous migration to synchronous migration, the communication overhead nearly vanishes. However, the 8.7% of communication overhead still appears in 10 processors.

To gain insight into the behavior of the parallel algorithm, the detailed analysis of the communication overhead is investigated. Figure 6 shows the absolute time spent in major functions of the communication. The barrier time is the time spent on waiting for all processes to reach the same point of execution. The sending and receiving time is the time used to send and receive migrants between neighboring processors.

Asynchronous migration can reduce only the barrier time. In 10 processors, the sending and receiving time increases dramatically compared with 6 processors. This is due to the fact that all nodes are connected to an ethernet hub. When increasing the number of nodes, it also increases packet collision probability. Thus, the percentage of communication overhead is not reduced as much as when using a small number of processors.

4 Conclusions

This study presents an improvement of parallel genetic programming for solving the mobile robot navigation problem. The parallel implementations are based on a coarse-grained model for parallelization. Asynchronous and synchronous parallelization approaches are examined on a clustered computer.

In order to avoid a controversy in the speedup definition, the relative time is used in the performance comparison. The change in migration topology results in an improvement of the relative time while keeping the level of solution quality. In asynchronous migration, the relative time using 10 processors increases to 32. This means that the parallel algorithm is 32 times faster than the serial algorithm by using 10 processors.

References

[1] Chongstitvatana, P., “Improving robustness of robot programs generated by genetic programming for dynamic environments”, Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, pp.523-526, Chaingmai, Thailand, 1998.

[2] Tongchim, S. and Chongstitvatana, P., “Comparison between synchronous and asynchronous implementation of parallel genetic programming”, Proceedings of the Fifth International Symposium on Artificial Life and Robotics (AROB), pp.251-254, Oita, Japan, 2000.

[3] Becker, D.J., Sterling, T., Savarese, D., Dorband, J.E., Ranawak, U.A. and Packer, C.V., “BEOWULF: A parallel workstation for scientific computation”, Proceedings of International Conference on Parallel Processing, 1995.

[4] Cantú-Paz, E., “A survey of parallel genetic algorithms”, Calculateurs Parallèles, Reseaux et Systems Repartis, Vol. 10, No. 2, pp.141-171, 1998.

[5] Koza, J.R. and Andre, D., “Parallel genetic programming on a network of transputers”, Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, University of Rochester, National Resource Laboratory for the Study of Brain and Behavior, Technical Report 95-2, pp.111-120, 1995.

[6] Tongchim, S. and Chongstitvatana, P., “Speedup improvement on automatic robot programming by parallel genetic programming”, Proceedings of IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp.77-80, Phuket, Thailand, 1999.

[7] Gustafson, J., “Fixed time, Tiered memory, and Superlinear speedup”, Proceedings of the Fifth Distributed Memory Computing Conference (DMCC5), 1990.

[8] Punch, B., “How effective are multiple poplulations in genetic programming”, Proceedings of the Third Annual Conference in Genetic Programming, pp.308-313, 1998.

[9] Cantú-Paz, E., Designing efficient and accurate parallel genetic algorithms, PhD thesis, University of Illinois at Urbana-Champaign, 1999.

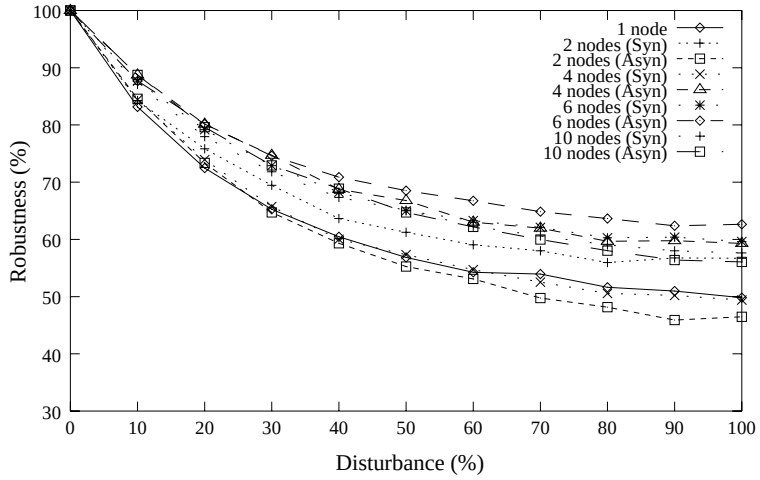

Figure 2: Robustness

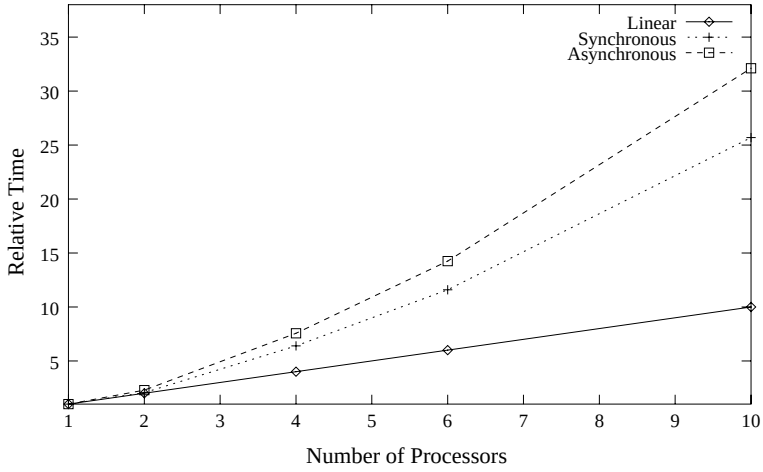

Figure 3: Relative Time

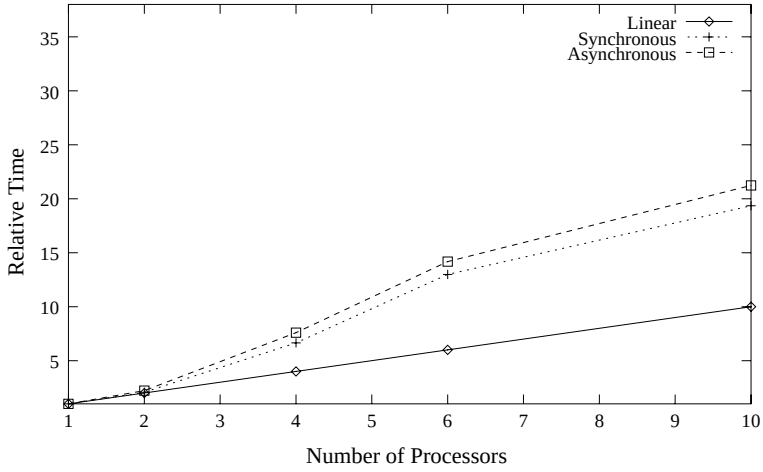

Figure 4: Relative Time of the Previous Work

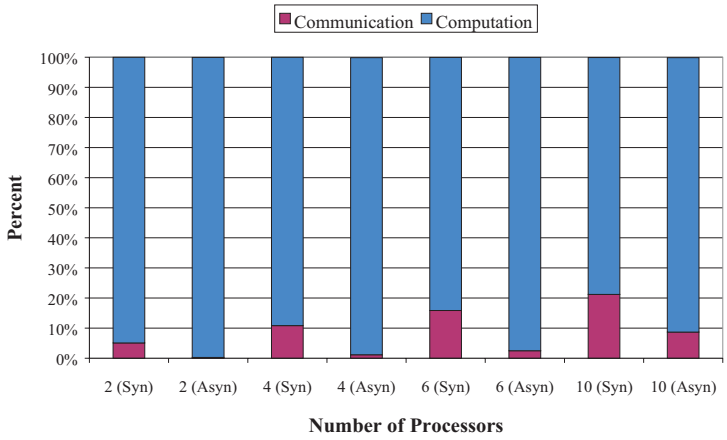

Figure 5: Percentage of Time Spent in Computation and Communication

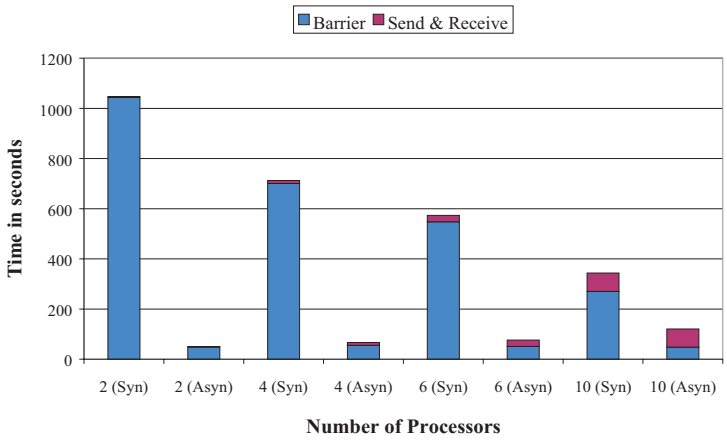

Figure 6: Absolute Time Spent in Communication