Parallelization of Genetic Programming for Solving Mobile Robot Navigation Problem on NOWs

Shisanu Tongchim[1] and Prabhas Chongstitvatana[2]
Department of Computer Engineering, Faculty of Engineering,
Chulalongkorn University, Bangkok 10330, Thailand
Tel: (662) 218-6982, Fax: (662) 218-6955
E-mail: g41stc@cp.eng.chula.ac.th[1], prabhas@chula.ac.th[2]

Abstract: Genetic Programming (GP) has been used to solve a large number of complex problems in various application domains. However, it is known that GP may take the substantial processing time to solve a given problem depending the time required for the fitness evaluation. In this paper, the parallel implementations of GP for solving the mobile robot navigation problem on NOWs (Networks of Workstations) were proposed. Our objective was to reduce the execution time by distributing the task of the serial GP to different processors. Two parallel implementations were examnied. The first implemented a conventional coarse-grained model while the second used a heterogeneous coarse-grained model. The results showed that the second implementation achieved superlinear speedup with the same quality of the answers.
Key words: Parallel Genetic Programming, Mobile Robot Navigation Problem

1. Introduction

The idea of using the perturbation to improve robustness of the robot programs generated by genetic programming (GP) was proposed in the previous study [1]. Despite the success of this technique, the obvious drawback is the substantial amount of processing time. However, the genetic programming process can be easily implemented as a parallel algorithm since the fitness evaluation of a population of candidate solutions can be performed independently.

This work presented two parallel implementations of the previous GP program used in [1] on a dedicated cluster of workstations. The first implementation was based on a conventional coarse-grained model for parallelization. The second implementation adopted the concept of a heterogeneous coarse-grained model by distributing the objective functions, in an attempt to further improve the speedup of the parallel algorithm.

The remaining sections are organized as follows: The next section discusses the related work. Section 3 is a description of the mobile robot navigation problem. Section 4 describes a problem representation in the serial GP. Section 5 shows the parallel solutions. Section 6 presents the experimental results. Finally, section 7 provides the conclusions of this work.

2. Related work

Cantú-Paz [2] 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 in this model.

Unlike genetic algorithm, there is little research on parallel GP. The earlier work of parallel GP was implemented on a network of transputers by Koza and Andre [3]. 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.

Dracopoulos and Kent [4] proposed the use of the Bulk Synchronous Parallel Programming (BSP) model to parallelize genetic programming. Two approaches of parallel GP were examined on a cluster of Sun workstations. The first was based on a master-slave model while the second was based on a coarse-grained model. This work used the Artificial Ant problem to evaluate the performance of two implementations. The results showed that the achieved speedup was close to linear. However, the implementations in this work were based on general models for parallelization, the further studies in order to improve the speedup were not investigated.

Oussaidène et al. [5] applied parallel genetic programming to evolve trading model strategies. The parallel scheme was based on a master-slave model. A non-preemptive dynamic scheduling algorithm was proposed to deal with uneven loads among processors. The results showed that the acceptable trading models can be acquired with near linear speedup.

A recent paper by Punch [6] presented the empirical study about some problem-specific factors which affect the effectiveness of parallel GP. Punch concluded that the achieved performance of parallel GP by using a

coarse-grained model may vary according to the nature of problems.

3. The mobile robot navigation problem

Our previous work [1], GP was used to generate a robot control program for the obstacle avoidance task. The task was to control a mobile robot from a starting point to a target point in a simulated environment. The mobile robot had a round shape with the ability to move forward, turn left and turn right. The robot had sensors for detecting the collision when the mobile robot crashed into any obstacle and indicating whether the robot was nearer to the target compared to its previous position. The size of the simulated environment was 600×400 units. The environment was filled with the obstacles which had several geometrical shapes (see Fig. 1).

The aim of the work was to generate *robust* control programs. The perturbation of the training environments was proposed in order to improve the robustness of the robot program. In the evolution process, each individual was evaluated under many environments that were different from the original one. The result showed that the robustness of the robot programs was improved by such an approach. However, the considerable execution time was required to evaluate the fitness of the population of the robot programs.

4. Serial algorithm

The terminal set is composed of three primitive movement controls {`move`, `left`, `right`} and one sensor information {`isnearer`}. The function set is composed of three functions {`if-and`, `if-or`, `if-not`} with 4, 4 and 3 arguments respectively. The `move` command moves the robot forward by 1 unit and returns 1 if the robot hits an obstacle and 0 if it does not hit any obstacles. The `left` and `right` command change the robot direction by 22.5° of its previous direction. The `isnearer` indicates whether the robot is close to the target in the previous move. The GP parameters are shown in Table 1.

In the fitness evaluation, each robot program is executed in a specific number of environments that are different from the initial environment. The execution in each environment continues until either the robot achieves the target point or reaches an iteration limit when the robot executes 10,000 terminals. The fitness function is a sum of the fitness value in each environment which is based on the distance of the final position and the number of moves. This fitness measurement scheme indicates that the smaller the value is, the more efficient the program will be.

$$f = \sum_{i=1}^{n} (10000 \times d_i + m_i) \qquad (1)$$

where,

n is the number of environments

Table 1: GP parameters

Total population	6000
Crossover probability	0.9
Mutation probability	0.1
Selected individual	5% of Total population
Maximum generation	200

d_i is the distance of the final position from the target position under the environment i

m_i is the number of moves under the environment i

As mentioned earlier, the fitness evaluation is carried out under several environments that are changed only slightly from the original one. We randomly select the obstacles and move them from their original positions by 5 units in a random direction. The difference between each environment and the original environment is defined as the percentage of disturbance (D).

$$D = \frac{N_m}{N_o} \times 100 \qquad (2)$$

where,

N_m is the number of obstacles that are moved

N_o is the total number of obstacles

In the evolution process, the percentage of disturbance is 20% and two experiments of the serial algorithm are examined with 5 and 8 as the number of training environments.

5. Parallel genetic programming

5.1 Concept

In a general coarse-grained model, 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. Each processor is responsible for selecting and mating in its own subpopulation. Every predetermined interval, some selected individuals are exchanged via a migration operator. The model is also known as *Island model* and the subpopulation is called *deme* [3].

Besides using the similar parameters on each node, a coarse-grained model allows the subpopulations to use different genetic parameters, coding, operators and objective functions since each subpopulation evolves independently from the others. This model is called a heterogeneous coarse-grained model.

5.2 Implementation

We investigate two different models of implementation. In the first model, each node uses the same genetic parameters and the environments for the fitness evaluation.

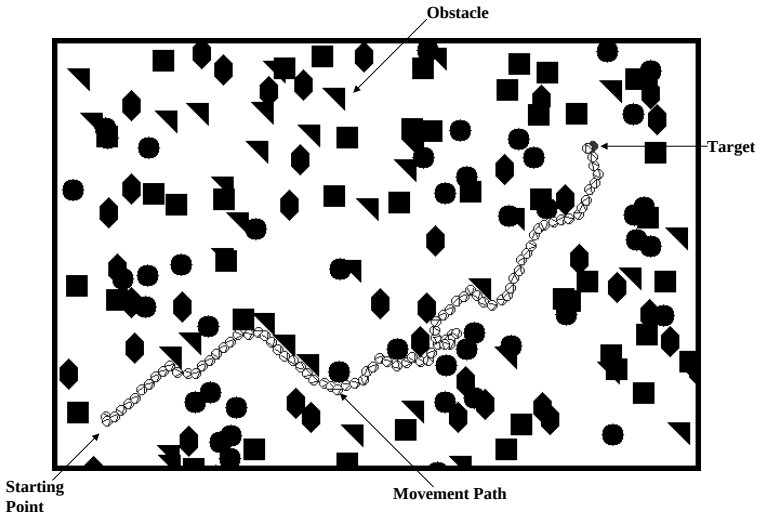

Figure 1: Simulated environment

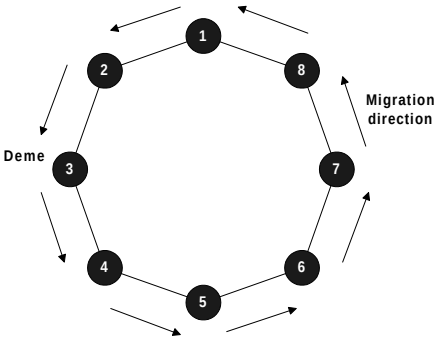

Figure 2: Ring topology

The connection is a ring topology (See Fig. 2). By synchronizing migration, the communication is separated into two steps. First, the nodes with odd number send the population to another even number. Then, the nodes with even number send the population to another odd number in the same direction. The number of environments for the fitness evaluation is 5, and the top 10 individuals of each subpopulation are exchanged during the migration phase which occurs every generation.

In the second model, we try another method to further improve the speedup by using the concept of a heterogeneous coarse-grained model. The environments are divided among the processing nodes. After a specific number of generations, every subpopulation is migrated between processors using a fully connected topology. This scheme causes the reduction of robustness since each individual has a shorter period in each training environment. To mend this problem, we increase the number of environments in each node. However, an important consideration is that the number of environments in each node should be less than the conventional model, hence the amount of work is reduced . Several trials are examined to find an appropriate value for the number of environments (see Table 2). The migration is carried out as follows: each node sends its subpopulation to all other nodes by the broadcast function, this is repeated for every node. The top 5% of individuals from each subpopulation are exchanged during the migration.

In both models, the total population is held constantly for the task and is divided equally among workstations. The number of selected individuals, crossover operation, mutation operation, reproduction is the percentage of the amount of the total population. The parallel efficiency is measured by varying the number of nodes and the results are averaged over 20 runs for each number of nodes.

6. Results and discussion

The widely used performance evaluation of the parallel algorithm is the parallel speedup. To make an adequate comparison between the serial algorithm and parallel algorithm, Cantú-Paz [7] suggests that the two must give the same quality of the solution. In this paper, the quality of the solution is defined in terms of *robustness*. The following section describes the robustness in more details.

6.1 Robustness

The robustness (R) is the percentage of the success of a robot program in the unseen environments.

$$R = \frac{N_s}{N_t} \times 100 \tag{3}$$

where,

N_s is the number of success runs
N_t is the number of total runs

Table 2: Experimental parameters of the second model

Processors	Population size (per node)	Environments (per node)	Migration interval (generation)
1	6000	8	NA
2	3000	7	100
4	1500	4	50
6	1000	3	34
10	600	2	20

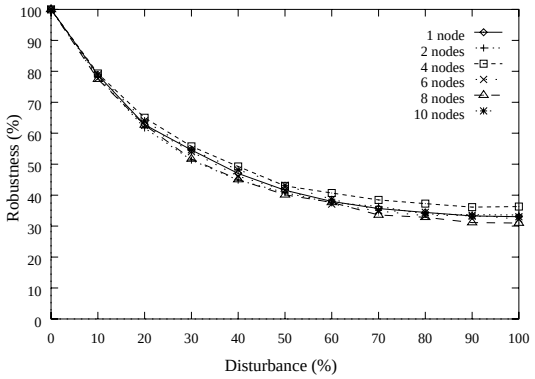

Figure 3: The first model robustness

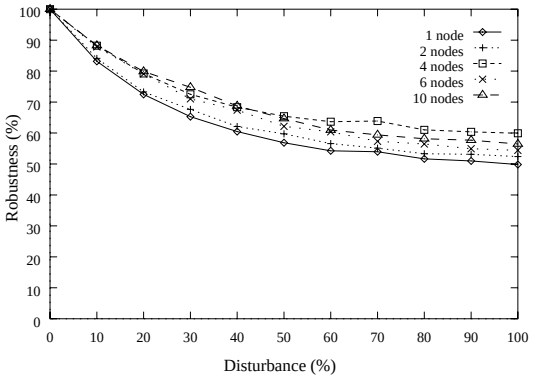

Figure 4: The second model robustness

We use the robustness graph which illustrates the robustness against the percentage of disturbance to compare the ability of the robot control programs from the serial and parallel algorithm. The robustness is averaged from the best individual from 20 runs in each algorithm, measured under 1000 new testing environments and the percentage of disturbance is varied from 0-100%. The robustness graph of the first model is shown in Fig. 3 while the robustness graph of the second model is depicted in Fig. 4. The robustness graphs of the first model and the second model are compared against the robustness from the serial algorithm with 5 and 8 training environments respectively. Both graphs show that the parallel solutions and serial solutions have the same robustness.

6.2 Speedup

The parallel speedup (S_p) is defined as the ratio of the serial execution time (T_s) to the parallel execution time on n processors (T_n).

$$S_p = \frac{T_s}{T_n} \tag{4}$$

The speedup of the first model is illustrated in Fig. 5. The graph shows that near linear speedup can be acquired with a small degradation in the speedup as the number of processors increases. The achieved speedup is caused from the benefit of the multiple populations distributed across separate processors.

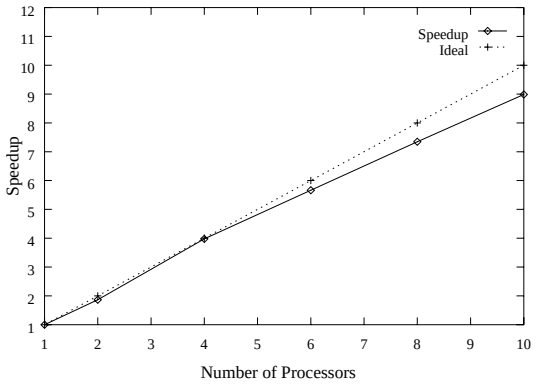

Figure 5: Speedup of the first model

The speedup of the second model is shown in Fig. 6. The graph shows that this model achieves superlinear speedup. This significant performance is caused by two factors; the speedup from the populations distributed across different processors and the speedup obtained by distributing the training environments. However, there is a degradation of the speedup for 10 processors. Thus, the next section investigates the further detailed analysis of the communication overhead.

6.3 Communication overhead

Figure 7 shows the percentage of the time spent in the communication overhead of the first implementation. Since the relative time spent in the communication over-

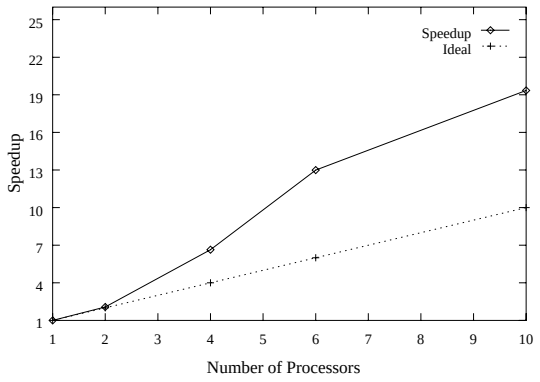

Figure 6: Speedup of the second model

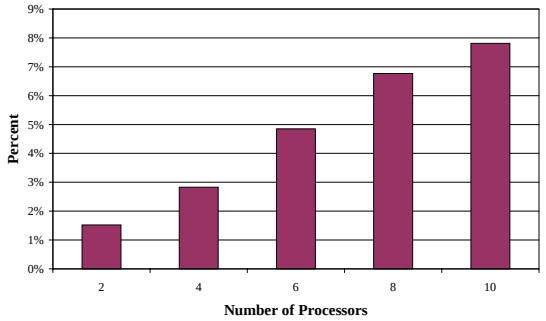

Figure 7: Percentage of time spent in the communication overhead (the first model)

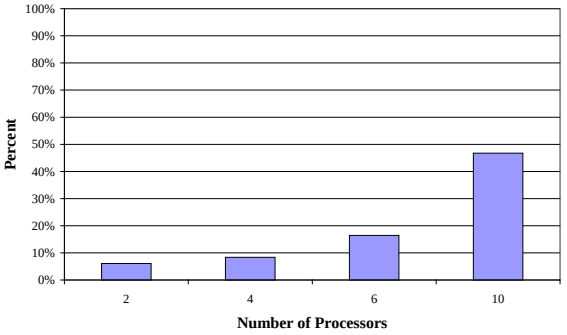

Figure 8: Percentage of time spent in the communication overhead (the second model)

head is relatively small – i.e. less than 8% for 10 processors, the achieved speedup of the first implementation is close to the number of processors used.

The percentage of the time spent in the communication overhead of the second implementation is illustrated in Fig. 8. The graph indicates a sharp increase in the communication overhead in 10 processors which obliterates the benefit of additional processors.

Figure 9 shows the absolute time spent in major communication functions of the second implementation. The communication overhead is the sum of the communication time and the barrier time. The communication time is the sum of the time spent on sending and on receiving the information among the processors. The barrier time is caused from uneven work loads among the processors due to the different time required for the evaluation of the robot programs. In case of a small number of processors, the barrier synchronization is identified to be the primary source of the overhead. On the contrary, the broadcast time is the major source of the overhead in case of a large number of processors.

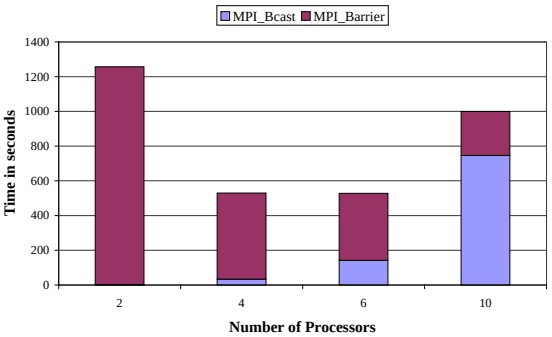

Figure 9: Absolute time spent in the communication overhead (the second model)

7. Conclusions and future work

This work investigated the use of parallel processing

in order to reduce the execution time of genetic programming. The problem chosen to examine the parallel implementations was the mobile robot navigation problem. Two parallel approaches were examined. The experiments compared the performance of the serial GP and parallel GP with varying the number of processing nodes. The performance analysis of the parallel algorithm was measured in terms of the parallel speedup. The experimental results showed that the speedup of the second method was greater than linear and the solutions from the serial and parallel method had the similar quality.

Based on the detailed analysis of the communication overhead, the direction of future work will focused on reducing the communication overhead. It is expected that the further study will enable the achieved speedup to be improved.

References

[1] Chongstitvatana, P., Improving robustness of robot programs generated by genetic programming for dynamic environments, *Proc. of IEEE Asia Pacific Conference on Circuits and Systems*, pp. 523–526, 1998.

[2] Cantú-Paz, E., A survey of parallel genetic algorithms, *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[3] Koza, J.R. and Andre, D., Parallel genetic programming on a network of transputers, *Proc. 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.

[4] Dracopoulos, D.C. and Kent, S., Bulk synchronous parallelisation of genetic programming, *Proc. of the Third International Workshop on Applied Parallel Computing in Industrial Problems and Optimization (PARA '96)*, Springer Verlag, Berlin, 1996.

[5] Oussaidène, M., Chopard, B., Pictet, O.V. and Tomassini, M., Parallel genetic programming: An application to trading models evolution, *Proc. of the First Annual Genetic Programming Conference*, MIT Press, Cambridge Massachusets, pp. 357–380, 1996.

[6] Punch, B., How effective are multiple populations in genetic programming, *Proc. of the Third Annual Genetic Programming Conference*, San Francisco, CA, Morgan Kaufmann, pp. 308–313, 1998.

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