Speedup Improvement on Automatic Robot Programming by Parallel Genetic Programming

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

Abstract

Genetic Programming has been successfully used to perform automatic generation of robot programs. However, to improve robustness of the generated robot programs, each candidate solution was evaluated under many environments which required the substantial processing time. This study proposed a parallel implementation to reduce the execution time. By using a coarse-grained model for parallelization, called Island Model, a near linear speedup was achieved with small communication overhead. In addition, the barrier synchronization was identified to be the primary source of the overhead.

1. Introduction

Genetic Programming (GP) [1] is an effective search algorithm which automatically generates programs to perform a given task. The major behavior of genetic programming is adopted from Genetic Algorithm (GA) [2]. The obvious distinction between genetic programming and genetic algorithm is the representation of the population of candidate solutions. In genetic programming, each individual in the population is a program which often appears in a tree structure, rather than a fixed-length string used in genetic algorithm.

In recent years, GP has been accepted as a promising method to undertake a large number of complex problems in various application domains. As there is an increase in the use of this method, the need for speeding up the GP process by means of parallel processing becomes important [3].

Parallel GP was earlier implemented on a network of transputers by Koza and Andre [4]. A related work in parallel GP implemented on a cluster of workstations used a master-slave model [5]. However, there is little research on parallel GP. In contrast, a lot of studies on parallel GA in various architectures were conducted, as shown in [6]. The reason for lack of studies may come from the problem-dependent of GP – the effectiveness of parallel GP varies according to the nature of problems [7]. We want to study the behavior of parallel GP on a realistic problem in contrast to some synthetic problems. Toward this goal, we implemented a parallel version of the previous work on GP [8] using a coarse-grained parallelization. The implementation was carried out on a dedicated cluster of PC workstations. The results showed that the speedup factor was close to linear. The quality of the solutions generated from the serial and parallel GP were also compared

The remaining sections are organized as follows: The next section is a brief introduction about genetic programming. Section 3 is a description of the obstacle avoidance problem. Section 4 describes a problem representation in the serial GP. Section 5 shows the parallel solution. Section 6 presents the research findings. Finally, section 7 provides the conclusions of this work.

2. Genetic Programming

The algorithm starts with the initial *population* of randomly generated programs. The programs in the population are expressed as parse trees, which are composed of *functions* and *terminals* appropriate to the given problem. After the algorithm creates the population, each individual in the population is ranked by means of a *fitness function* – a function which returns the fitness value according to the quality of the solution. Then the new population is created by applying the genetic operators such as *crossover*, *reproduction* and *mutation* on the individuals which are chosen with probabilities based on the fitness values. Then, the algorithm replaces the old population with the new population and iterates by using the new population. One cycle of these procedures is referred to as a *generation*. This continues until the fitness value indicates that the goal is achieved or an iteration limit is reached.

3. The Obstacle Avoidance Task

Our previous work [8], GP was used to generate a robot control program for the obstacle avoidance task. The task is to control a 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 of the obstacle and indicating whether the robot is nearer to the target compared to its previous position. The size of the simulated environment is 600x400 units. The environment is filled with the obstacles which have several geometrical shapes (see Fig. 1).

The aim of the work is 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 is required to evaluate the fitness of the population of the robot programs.

4. Serial Algorithm

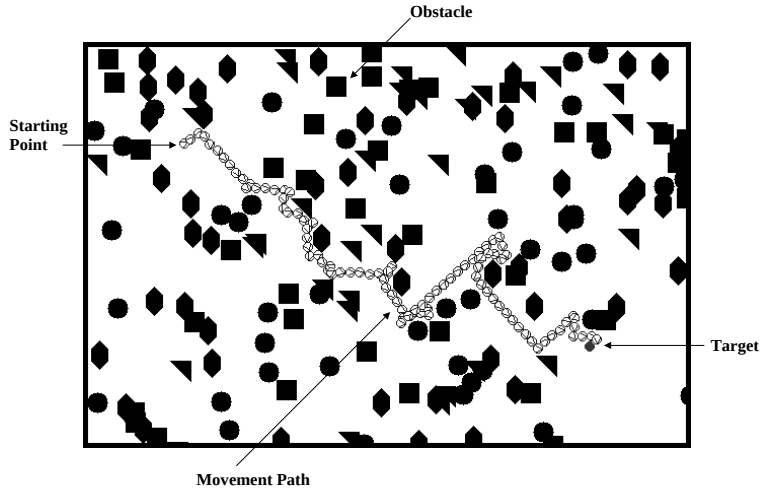

Fig. 1 Simulated Environment

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.

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

where,

n is the number of environments
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 obstacle and move it from its original position by 5 units in a random direction. The difference between each environment and the original environment is defined as the percent of disturbance (D).

Table 1 GP parameters

Total population	6000
Crossover probability	0.9
Mutation probability	0.1
Selected individual	5% of Total population
Migration size	10
Maximum generation	200

$$D = \frac{N_m}{N_o} \times 100 \tag{2}$$

where,

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

In the evolution process, the percent of disturbance is 20% and the number of training environments is 5.

5. Parallel Genetic Programming

Coarse-grained parallelization

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. 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* [4].

Implementation

We implement our parallel algorithm on the dedicated cluster of PC workstations with 133 MHz Pentium processors,

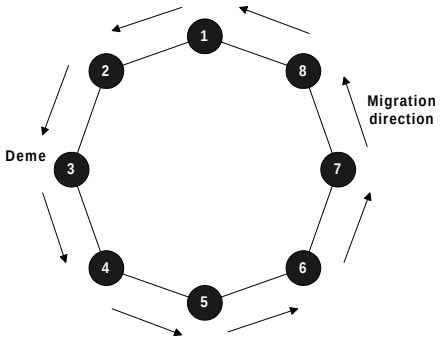

Fig. 2 Topology

each with 32 Mb of RAM, and running Linux as an operating system. These machines are connected via 10 Mbs ethernet cabling. We extend the program used in [8] to run under a clustered computer by using MPI as a message passing library. 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.

6. Results and Discussion

The parallel efficiency is measured by varying the number of nodes and the results are averaged over 20 runs for each number of nodes. 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 a percentage of the amount of total population.

The widely used performance evaluation of the parallel algorithm is the parallel speedup. However, in the field of parallel GA, there are some controversies about the speedup measurement, especially in the coarse-grained model [7,9]. This is due to the fact that the serial GA and parallel GA are not the same algorithm since the coarse-grained model changes the behavior of the traditional GA. To make an adequate comparison between the serial algorithm and parallel algorithm, E.Cantú-Paz [9] 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.

Robustness

The robustness (R) is the percent 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

The robustness is averaged from the best individual from 20 runs in each algorithm, measured under 1000 new testing environments and the percent of disturbance is varied from 0–100%. From the robustness graph (see Fig. 3), the robustness of the programs generated from the parallel GP is similar to the serial GP.

Speedup

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

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

The achieved speedup is depicted in Fig. 4. The graph shows that a near linear speedup can be acquired with a small degradation in the speedup as the number of processors increases.

Communication overhead

In this section, we investigate the source of the communication overhead to discern the cause of the small degradation in the speedup. Figure 5 shows the absolute time spent in the communication overhead. 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. From the graph shows that the communication time increases consistently as the number of processors increases.

The barrier time is caused from uneven work loads among the processors. Due to the fact that the robot performs the task until either the robot achieves the target point or reaches an iteration limit, the time required to complete the evaluation varies, with the least effective programs taking the longest period and the best programs taking the shortest period. The barrier time is the primary source of the overhead. The cause of the variation of the barrier time is due to the different random seeds in each number of processors.

The percentage of the time spent in the communication overhead is illustrated in Fig. 6. The relative time spent in the communication overhead increases slightly as the number of processors increases. Although the additional processors may not increase the absolute time (e.g. in 8 and 10 processors), this still increases the relative time due to the reduction of the computational time by the benefit of additional processors. In case of a small number of processors, the achieved speedup is close to the number of processors used since the percentage of time spent in the communication overhead is relatively small.

7. Conclusions

This paper proposed a success of speeding up the genetic programming process by means of parallel processing. The parallel implementation was done on a clustered computer by using a coarse-grained model. The study compared the performance of the serial GP and the coarse-grained parallel GP with varying the number of processing nodes. Performance analysis of the parallel algorithm was measured in terms of the parallel speedup.

The experimental results showed that the speedup was close to linear and the solutions from both methods had the similar quality. Furthermore, the overhead due to the communication

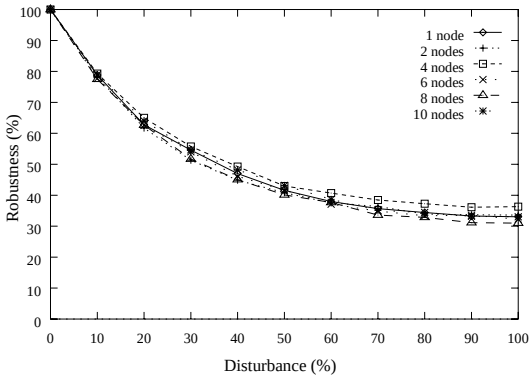

Fig. 3 Robustness

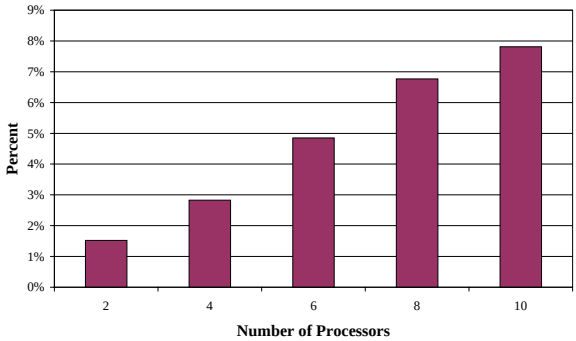

Fig. 6: Percentage of time spent in the communication overhead

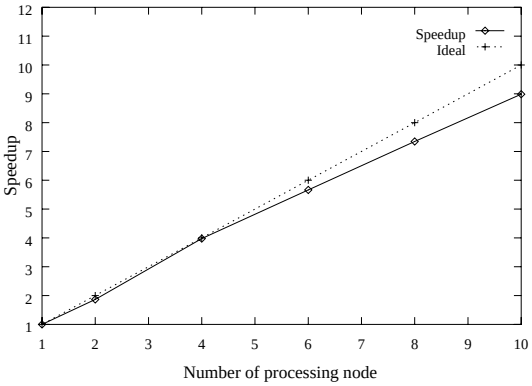

Fig. 4 Speedup

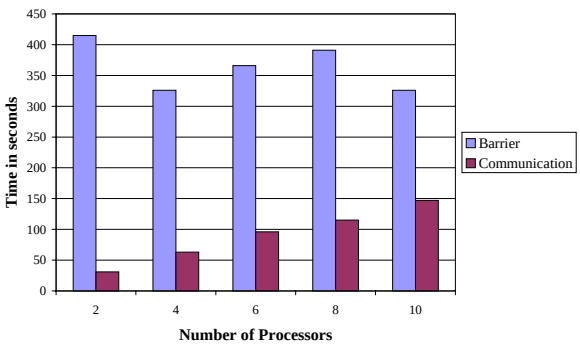

Fig. 5 Absolute time spent in the communication overhead

among the processing nodes was small and slightly increased toward the increment of processors.

References

[1] J.R.Koza, "Genetic Programming," MIT Press, Cambridge, Massachusetts, 1992.

[2] D.E.Goldberg, "Genetic Algorithm in Search, Optimization, and Machine Learning," Addison-Wesley, 1989.

[3] H.Juillé and J.B.Pollack, "Parallel Genetic Programming on Fine-Grained SIMD Architectures," Working Notes of the 1995 AAAI Fall Symposium on Genetic Programming, Cambridge, Massachusets, 1995.

[4] J.R.Koza and D.Andre, "Parallel genetic programming on a network of transputers," 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.

[5] D.Dracopoulos and S.Kent, "Speeding up genetic programming: A parallel BSP implementation," The First Annual Conference in Genetic Programming, MIT Press, Cambridge, Massachusets, 1996.

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

[7] B.Punch, "How effective are multiple poplulations in genetic programming," The Third Annual Conference in Genetic Programming, pp. 308–313, 1998.

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

[9] E.Cantú-Paz, "Designing Efficient and Accurate Parallel Genetic Algorithms," PhD thesis, University of Illinois at Urbana-Champaign, 1999.