Asynchronous Migration in 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
g41stc@cp.eng.chula.ac.th
prabhas@chula.ac.th

Genetic Programming (GP) was used to generate robot control programs for an obstacle avoidance task [1]. The task was to control an autonomous mobile robot from a starting point to a target point in a simulated environment. The environment was filled with the obstacles which had several geometrical shapes. In order to improve the robustness of the program, each program was evaluated under many environments. As a result, the substantial processing time was required to evaluate the fitness of the population of the robot programs.

To reduce the processing time, this present study introduced a parallel implementation. In applying the parallel approach to the algorithm program by using a conventional coarse-grained model, the result achieved only linear speedup since the amount of work was fixed – the algorithm was terminated when it reached the maximum generation. Hence, the parallel algorithm did not exploit the probabilistic advantage that the answer may be obtained before the maximum generation.

We tried in this present study another method to further improve the speedup by dividing the *environments* among the processing nodes. After a specific number of generations, every subpopulation was migrated between processors using a fully connected topology. The parallel algorithm was implemented on the 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 Mbs ethernet cabling. We extended the program used in [1] to run under the clustered computer by using MPI as a message passing library.

In the first stage of the implementation, the migration was synchronized. The synchronizing migration resulted in uneven work loads among the processors. This was due to the fact that the robot performed the task until either the robot achieved the target point or reached an iteration limit. Hence, this migration scheme caused the evolution to wait for the slowest node.

In the second stage of the implementation, we attempted to further improve the speedup of the parallel algorithm by the *asynchronous migration*. When the fastest node reached predetermined generation numbers, the migration request was sent to all subpopulations. Therefore, this scheme caused the evolution of all subpopulations to proceed according to the *fastest* node.

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, E. Cantú-Paz [2] suggested that the two must give the

same quality of the solution. In this paper, the quality of the solution is defined in terms of the *robustness*. The robustness of the generated programs from the parallel algorithm was demonstrated to be better than the serial algorithm. Consequently, the amount of work from the parallel algorithm in this experiment was not less than the serial algorithm.

Figure 1 illustrates the speedup observed on the two implementations as a function of the number of processors used. Both implementations exhibit superlinear speedup. The speedup curves taper off for 10 processors and the performance of the asynchronous implementation is slightly better than the performance of the synchronous implementation.

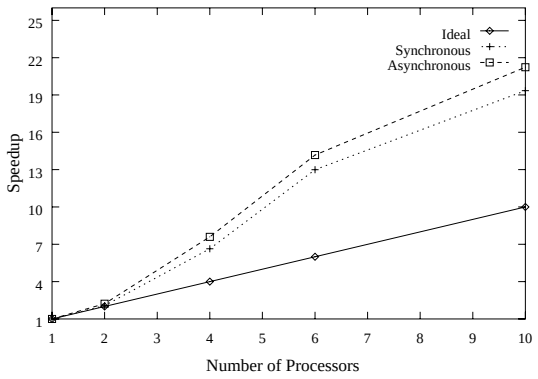

Fig. 1. Speedup

After obtaining some timing analyses, the results reveal the cause of the problem. The performance degradation in 10 processors is caused by the excessive communication time due to the broadcast function. Although the asynchronous migration reduces the barrier time effectively compared to the synchronous migration, the increase in the broadcast time in 10 processors obliterates this advantage. However, in case of a small number of processors (2,4,6), the reduction of the communication overhead from the asynchronous migration compared with the synchronous migration is considerable – i.e. the reduction in 2,4,6 nodes is 96.09%, 84.44% and 62.42% respectively. In terms of the wall-clock time, the asynchronous implementation in this work using 10 nodes is 21 times faster than the serial algorithm.

References

1. Chongstitvatana, P.: Improving Robustness of Robot Programs Generated by Genetic Programming for Dynamic Environments. In: Proc. of IEEE Asia-Pacific Conference on Circuits and Systems. (1998) 523–526
2. Cantú-Paz, E.: Designing Efficient and Accurate Parallel Genetic Algorithms. PhD thesis, University of Illinois at Urbana-Champaign (1999)