A Path Planning Algorithm Based on Parallel Particle Swarm Optimization

Weitao Dang[1], Kai Xu[1], Quanjun Yin[1], and Qixin Zhang[2]

[1] College of Information System and Management
National Univ. of Defense Technology
Changsha, 410073, China
dangwt08@163.com
[2] Nanchang Military Academy
Nanchang, 330103, China

Abstract. A novel path planning algorithm based on Parallel Particle Swarm Optimization (PSO) is proposed in this paper to solve the real-time path planning problem in dynamic multi-agent environment. This paper first describes the advantages of PSO algorithm in real time search problems, i.e. path finding problems. Then considering the development trend of multiprocessors, the parallel PSO (PPSO) was proposed to speed up the search process. Due to the above mentioned advantages, we in this paper adopt the PPSO to distribute particles onto different processors. By exchanging data upon the shared memory, these processors collaborate to work out optimal paths in complicated environment. The resulting simulation experiments show that when compared with traditional PSO, using PPSO could considerably reduce the searching time of path finding in multi-agent environment.

Keywords: Path Planning, Particle Swarm Optimization, Parallelism, Multi-processor, Multi-agent environment.

1 Introduction

Path planning has become an unelectable issue in multiple domains, including computer games, robotics and simulation. The algorithm of path planning greatly limits the scale and development of training simulation and computer games which have relatively high real-time requirements. Researches from different directions have been done to enhance the efficiency of path planning.

A*[1] is one of the most widely used algorithms for path planning. It is a kind of best-first heuristic search method, which guarantees optimal path return when the heuristic is admissible. A* could always find the optimal path by expanding lots of nodes. However, it is time consuming and unpractical in the real-time applications.

Finding an optimal path is normally time consuming, thus people work on finding a sub-optimal path to decrease the computation time to an acceptable level. Many optimal algorithms which based on Evolutionary Computation (i.e., Genetic Algorithms (GA) and Particle Swarm Optimization (PSO)) are proposed in path planning. These algorithms can fleetly find a feasible path via stochastic searches, but still needs exponential time for finding an optimal one.

With the development of multi-core and relevant techniques, more and more attentions have been paid to the idea of parallel computing, which can extremely decrease the computing time across multiprocessors working collaboratively. Parallel computing have been quickly developed and widely applied. The former fundamental path planning algorithm associated with parallelism is based on A*, such as Parallel Bidirectional Search (PBS)[2], Parallel Retracting A* (PRA*)[3], Parallel Local A* (PLA*)[4] and so on.

Although these works could improve planning efficiency, there is too much synchronization overhead in the parallelism algorithms based on A*, which leads to the fact that efficiency couldn't improve linearly with the increase of processors. The overhead couldn't be eliminated because A* has to maintain an Open and a Close list and access the Open and Close List continually during the whole search process. Therefore, the synchronization is virulent for the improvement of the efficiency across parallelism.

PSO[5] is an evolutionary computation technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behaviors of bird flocking or fish schooling. PSO optimizes an objective function by conducting population-based search. The population consists of potential solutions, called particles, similar to birds in a flock. The particles are randomly initialized and then freely fly across the multi-dimensional search space. While flying, every particle updates its velocity and position based on its own best experience and that of the entire population.

Compared to A*, there is no data exchange between the particles of PSO, the only shared data is the best position of the entire particles. There will be little synchronization overhead to maintain the position by all the particles in the parallelism of PSO[6-8]. Because the computation of every particle is the same, we can distribute the particles among the cores averagely to obtain load balance. The characteristics of PSO make it suit for parallelism, and the speedup will increase along with the number of processors which is hardly for other algorithm like A*.

The organization of this paper is as follows. Firstly, we give a brief introduction about the PSO and how to apply it to path planning. Then we discussed the parallelism of the algorithm. Next, the implementation of the parallelism of PSO is presented and compared with the serial PSO to analyze about the optimization, speedup and the scalability of the parallel algorithm.

2 PSO and Path Planning

2.1 PSO

Particle Swarm Optimization is based on the social behavior that a population of individuals adapts to its environment by returning to promising regions that were previously discovered. This adaptation to the environment is a stochastic process that depends on both the memory of each individual as well as the knowledge gained by the population as a whole[9].

Each particle in the PSO represents a solution which has a function to estimate its fitness value, and the solution with the less fitness value is closer to the optimal solution. Each particle is initialized with a random position as well as the velocity. The algorithm then repeatedly updates the particles' position.

The position of each particle at iteration k+1 is calculated from Equation (1):

$$x_{k+1}^{i} = x_{k}^{i} + v_{k+1}^{i}\Delta t \tag{1}$$

where x_{k+1}^{i} is the position of particle i at iteration k+1, v_{k+1}^{i} is the corresponding velocity, and Δt is the time step of an iteration.

The velocity of the particle is calculated from Equation (2):

$$v_{k+1}^{i} = \omega v_{k}^{i} + c_1 r_1 \frac{(p^{i} - x_{t}^{i})}{\Delta t} + c_2 r_2 \frac{(p_{k}^{g} - x_{t}^{i})}{\Delta t} \tag{2}$$

Here, p^{i} is the best experience of particle i, where p_{k}^{g} is the best position amongst all the particles at iteration k. r_1 and r_2 are random values between 0 and 1. c_1 and c_2 are constant values, representing the degree of how v_{k+1}^{i} affected by p^{i} and p_{k}^{g} respectively. ω is the inertia coefficient of the particle, with a larger value facilitating a more global search and a smaller value facilitating a more local search.

2.2 Path Planning Using PSO

Path planning is a search problem in a map from start location to the goal location. Path planning using PSO[10] means searching a path without crossing the obstacles. Based on this, the path should be as far as possible optimal, namely to be as shorter or less cost.

Fig. 1 shows the path planning environment in which the black polygons represent the obstacles, and the point S is the start location while point G is the target location. Fig. 2 explains how to search a path using PSO.

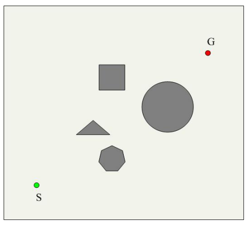

Fig. 1. The environment for path planning

To the path planning problem, one m-dimensions particle which consists of m points represents a path from the start point S to the goal point G. The fitness of the particle can be evaluated by the length of the path.

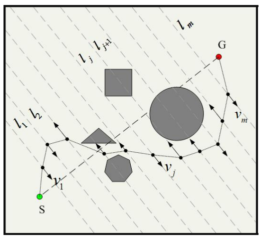

Fig. 2. Path Planning using PSO

As Fig. 2 shows, the segment $\overrightarrow{SG}$ is divided into m+1 parts. The search space of these dimensions is constrained by the m vertical lines (i.e. l_1 to l_m) respectively. Then connect S to the first point and G to the last point, we can obtain the path from S to G(the solid line in Fig. 2). The particle i for the iteration k can be represented as:

$$x_k^i = (x_k^i(1), x_k^i(2), \ldots, x_k^i(m)) \tag{3}$$

$x_k^i(1)$ to $x_k^i(m)$ in Equation (3) is the point on the vertical lines l_1 to l_m respectively.

The velocity of the particle i for the iteration k can be represented as:

$$v_k^i = (k_1 * \overline{e_1}, k_2 * \overline{e_2}, \ldots, k_m * \overline{e_m}) \tag{4}$$

$\overline{e_1}$ to $\overline{e_m}$ is the unit vector of the lines l_1 to l_m which indicate the direction of the velocity and k_1 to k_m is the coefficient to indicate the value. This makes every dimension of particles can only move on the vertical lines.

Based on above mentioned preconditions, the search process can be described as follow:

First, the process generates positions and velocities of the particles randomly. Then it iterates the updating process using Equation (1) and Equation (2). At every iteration, the best experience of every particle and that of all particles (p^i and p_k^g in Equation (2)) are also updated bases on the fitness value. The fitness of a particle is calculated by the evaluation function as Equation (5) :

$$f(x_k^i) = L(x_k^i) + k * L'(x_k^i) \tag{5}$$

The evaluation function consists of two parts: the length of path which the particle represented and the additional castigatory value. $L(x_k^i)$ in Equation (5) is the length of path which the particle represented, and $L'(x_k^i)$ is the total length of the path cross the obstacles, k is the castigatory coefficient.

When the terminate condition has been satisfied, the iteration process will be finished and the best experience of all particles generates the final selected path.

2.3 Parallelism of the PSO for Path Planning

The Parallelism of the PSO mainly distributes the particles to multi-cores so as to improve the efficiency of the algorithm. The particles in multiprocessors search simultaneously and they maintain a global best path together. For each iteration step, all of the particles compare their path with the global one. If one of the local paths is better, the global one will be updated. The detail steps of the algorithm are as follows:

- Step 1: coordinate transformation

For the convenience of computation, we need a coordinate transformation to transform the segment of S to G into X axis. Origin of the new coordinate is start point. Fig. 3 shows the transformed coordinate of Fig. 2.

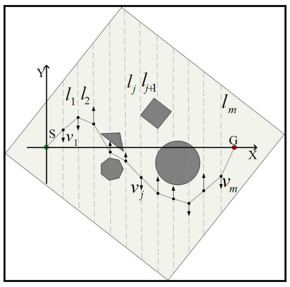

Fig. 3. The transformed coordinate

- Step 2: calculate the search space

The range of value in each dimension is calculated. The segment $\overrightarrow{SG}$ is divided into m+1 equal parts. Then the vertical lines (i.e. l_1 to l_m) which pass the dividing points constraints the range of possible values for these dimensions.

- Step 3: initialization

Initialize position and velocity for each particle. Generate the position and velocity for all dimensions of each particle. Then calculate the fitness value of the particles.
Choose the particle with the least fitness as global best and regard the initial position of each particle as their own local best, then start iterate.

- Step 4: parallel search

Distribute particles to multiprocessors averagely and search parallel. Each iterate process contains several parts: calculate the velocity using the Equation (2), update the positions using the Equation (1) and evaluate the fitness and select the local best and global best at last.

- Step 5: terminate planning process

Detect if terminating conditions are satisfied. Had been satisfied, transform the global best in new coordinate to the original coordinate and the result is the path which has been found.

The process of the algorithm could be described with the following pseudo code:

```
Procedure Parallel_Path_Palnning(Map, startPnt, endPnt)
      PSOInit()
      {
        CoordinateConversion( startPnt, endPnt )
        CalculateYRange()
        InitParticlePos()
        InitParticleVel()
        InitPBest()
        InitGBest()
      }
      ParallelPSO( CoreNum )
      {
        for i = 1 to CoreNum
          BeginThread(i)
        end
        For every Thread
          while( isRunning )
            CalculateVel()
            UpdatePos()
            AssessParticleFitness()
            UpdatePBest()
            UpdateGBest()
          end
      }
```

3 Implementation and Experimental Results

This section presents the results of the parallel PSO path planning algorithm described in section 2 and compared it with the serial algorithm for performance analysis. The test system used for the experiments is a 12-core Intel E5620(2.40 GHz with 12 MB Cache) processor, 16 GB RAM and Windows Server 32-bit OS.

We implement the algorithm with Visual C++, regard the whole window frame as the map, one pixel in the window represents one meter in reality. Polygons in the window simulate obstacles. We build a map with fixed width and height, set the start location and target location and add different number of obstacles to the map so as to simulate the environment with different complexities.

The parameters of functions for experiments are set as follow: the maximum number of iterations was 300, the value of c_1 and c_2 in Equation (2) are both set to be 2, ω in Equation (2) decrease from 0.9 to 0.4 with the function:

$$\omega = \omega_{\max} - (\omega_{\max} - \omega_{\min}) * i / n$$

in which $\omega_{\max}$ is 0.9 and $\omega_{\min}$ is 0.4, n is the max number of iterations while i is the current number of iterations. The number of dimensions is set to be 20 and the number of particles is set to be 40.

We will compare the parallel and serial algorithm in several aspects to evaluate performance of the parallel algorithm based on the parameters above.

3.1 Length of Path

Firstly, we compare the average length of the resulting paths obtained by our algorithm and the serial one respectively. As shown in Fig. 4, the number of the obstacles located randomly in the map ranged from 3 to 30. For each scenario both the two algorithms have been executed for 20 times, taking the average length as the result. X axis is the number of obstacles and Y axis is length of the path found by the two algorithms respectively. The blue line represents the length of parallel algorithm while the red is the serial one.

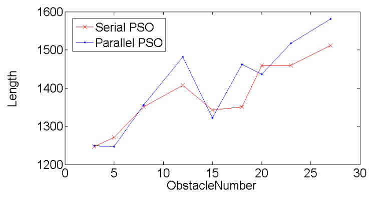

Fig. 4. Average length of the two algorithms

From the line chart, we could see that the average lengths of the resulting paths are almost the same. As a whole, the paths found by parallel are a little longer than which found by serial algorithm. But the longer is less than 5%.

3.2 Speedup

We research on the efficiency by comparing the speedup of parallel algorithm. Fig. 5 shows the searching time of two algorithms in each scenario. The red line represents the time spent for searching using serial algorithm while the blue one represents the time using parallel algorithm implementing on two threads. X axis is the number of obstacles and Y axis is the time spent for searching.

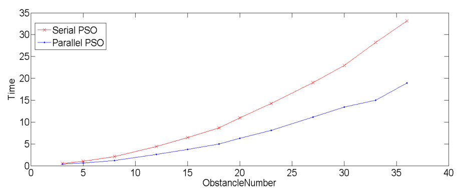

Fig. 5. The speedup of parallel algorithm with two threads

To have an intuitionistic understand of the speedup, Table 1 shows the speedup of the parallel algorithm implement with two threads. Speedup represents the time ratio of time consuming of serial algorithm and that of parallel algorithm.

Table 1. The speedup of the parallel algorithm

Obstacle	5	8	12	15	20	23	27	33	36
Time ratio	1.70	1.73	1.69	1.74	1.74	1.76	1.70	1.80	1.75

3.3 Scalability

We implement the parallel with different number of threads to study the scalability. The experiments are implemented repeatedly in different scenarios. Fig. 6 is the speedup of parallel algorithm with different number of threads. X axis is the number of cores the algorithm implements in, Y axis is the speedup compared with the serial algorithm. The red line represents the speedup of parallel algorithm. The blue dashed is an ideal linear speedup.

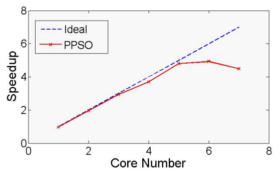

Fig. 6. The speedup of parallel algorithm with different threads

Fig. 6 shows that the parallel algorithm has a near ideal speedup when the number of cores is two or three. While with the increase of the number of cores, the speedup decreases obviously. So much as the speedup of 7 cores is worse than that of 6 cores. Synchronization overhead that all threads have to access the shared memory makes the decrease of speedup with the increase of core number.

3.4 Summary

Three aspects of comparison are presented in this section. The length of path found by parallel PSO algorithm is a little longer than that of the serial one. And the parallel algorithm can obtain a near linear speedup when the number of cores is less than 5. Because of the synchronization overhead, the speedup will be worse with the increase of the number of cores.

4 Conclusions and the Future Work

In this paper, we present the parallelism of PSO used in path planning. General experiments are presented to analyze the parallel PSO algorithm in several aspects, showing that the parallel PSO could obtain a near linear speedup and the final path found by parallel algorithm is only a little longer than that of the serial one. The parallel algorithm has a good scalability while the number of cores should not be large.

We expect to using parallelism in the whole path planning process. Firstly, parallel construct the environment, we plan to partition the map average and generate the waypoint in each area. Secondly, search the key node of path among waypoints. Finally, a refined planning between the key nodes is executed parallel.

References

1. Rich, E.: Artificial intelligence. McGraw-Hill (1983)
2. Brand, S.: Efficient obstacle avoidance using autonomously generated navigation meshes (Doctoral dissertation, MSc Thesis, Delft University of Technology, The Netherlands) (2009)
3. Evett, M., Hendler, J., Mahanti, A., Nau, D.: PRA*: Massively parallel heuristic search. Journal of Parallel and Distributed Computing 25(2), 133–143 (1995)
4. Dutt, S., Mahapatra, N.R.: Parallel A* algorithms and their performance on hypercube multiprocessors. In: Proceedings of Seventh International Parallel Processing Symposium, pp. 797–803. IEEE (1993)
5. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, vol. 1, pp. 39–43 (1995)
6. Venter, G., Sobieszczanski-Sobieski, J.: Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. Journal of Aerospace Computing, Information, and Communication 3(3), 123–137 (2006)
7. Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T., George, A.D.: Parallel global optimization with the particle swarm algorithm. International Journal for Numerical Methods in Engineering 61(13), 2296–2315 (2004)
8. Chu, S.C., Roddick, J.F., Pan, J.S.: Parallel particle swarm optimization algorithm with communication strategies. Submitted to IEEE Transactions on Evolutionary Computation (2003)
9. Kennedy, J., Spears, W.M.: Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In: Proceedings of the IEEE International Conference on Evolutionary Computation, Anchorage, AK, USA, pp. 78–83 (1998)
10. Raja, P., Pugazhenthi, S.: Path planning for mobile robots in dynamic environments using particle swarm optimization. In: International Conference on Advances in Recent Technologies in Communication and Computing, ARTCom 2009, pp. 401–405. IEEE (2009)