SAND2012-1459 C

SAND2012-1459C

Parallel Particle Swarm Optimization (PPSO) on the Coverage Problem in Pursuit-Evasion Games

Shiyuan Jin[1], Damian Dechev[1,2], Zhihua Qu[1]
1: Department of EECS
University of Central Florida
Orlando, FL 32816
2: Sandia National Laboratories
Scalable and Secure Systems Department
Livermore, CA
sjin@knights.ucf.edu, ddechev@sandia.gov, qu@mail.ucf.edu

Keywords: Parallel computing, PSO, coverage optimization, pursuit-evasion games

Abstract

A Parallel Particle Swarm Optimization (PPSO) algorithm using MPI is implemented to solve the coverage problem of pursuit-evasion (PE) games where multiple pursuers need to cooperate to cover an agile evader's possible escape area within reasonable time. The area to be covered is complex and thus difficult to calculate analytically. With the use of PPSO, maximum coverage is achieved in less time, given the minimum number of pursuers. The computation time can be further reduced by optimizing the fitness function based on data locality. In addition, using variable length of communication data frame performs better than fixed length in reducing inter-process communication time when the number of processors increases (more than four in the test example). Simulation results show a comparison of the speedup, the computation time before and after optimizing the fitness function, and communication time between fixed and variable data frame. Pursuers' positions and orientations are also presented to show the effectiveness of the PPSO algorithm.

1. INTRODUCTION

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy [1] in 1995, inspired by social behavior of bird flocking in search for food. PSO has been successfully applied in many areas such as multi-objective optimization, wireless sensor network, base station coverage problem, and intelligent control. However, the major obstacle limiting the use of PSO in real-time execution is its long execution time [2]. Parallel PSO is a possible way to expand traditional PSO applications. [1]

Pursuit-evasion (PE) games are one of the challenging problems in which pursuers try to capture evaders as soon as possible whereas evaders attempt to avoid being captured. PE games have numerous applications such as military combat operations, agent cooperation and competition in multi-agent systems, and wireless sensor-actuator networks. The dynamics of the games requires that pursuers and evaders make strategies for the next step in real-time. If it takes seconds for pursuers to make a decision, it would give the evader enough time to escape, and vice versa. This paper focuses on one-step of PE games, i.e., given the evader's location and orientation, how pursuers find their positions and orientations such that the evader's possible escape area or reachable set (RS) is fully covered. Once the one-step coverage problem is solved, the PE game is just a sequence of many such steps.

Unlike most other optimization problems, the PE coverage problem has its unique features:

(1) The area to be covered is a complex polygon whose coverage is difficult to calculate analytically.

(2) The pursuers, whose reachable sets are similar to the evader's, can be regarded as a directional sensor–both location and orientation need to be optimized.

(3) Parallel processing is necessary. The problem needs to be solved within reasonable execution time.

The coverage problem is NP-hard [3]. To the best of the author's knowledge, no sequential or parallel PSO has been found to solve this problem. In this paper, a PPSO algorithm is implemented based on classic PSO. In PPSO, each processor or node (the two words will be used interchangeably hereafter) evaluates the fitness (or the percentage of coverage in this problem) values of particles. The master node collects data from the slave nodes and broadcasts the particle with the global fitness value. The slave nodes update their individual particles based on the global fitness value. Two approaches are tested to reduce the computation and communication time. The fitness evaluation function is optimized to reduce the evaluation time based on data locality information. Variable communication data length is utilized to eliminate unnecessary communication data in iterations when the

[1]Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

global fitness value is not updated.

The rest of the paper is organized as follows. Section 2. reviews related work regarding PSO coverage optimization and PPSO. Preliminaries of PSO and PE coverage problem are introduced in Section 3. and Section 4., respectively. The proposed PPSO approach is introduced in Section 5. Simulation results and discussions are presented and analyzed in Section 6. Conclusions and future work are offered in Section 7.

2. RELATED WORK

The problem considered in this paper is motivated by [4] where the authors use control theory and non-linear numerical optimization approach to solve the PE problem. The boundaries of reachable sets of the pursuers and evader were empirically approximated by a polynomial of an ellipse. However, their simulation shows that sometimes small regions of the evader's reachable set may become uncovered due to suboptimal control actions generated by the optimizer and the complexity of the problem. In addition, parallel processing and execution time were not the focus of the paper, although the author admitted that the execution time was an issue.

The coverage part of the PE problem may be solved by PSO. Although no PSO has been found in solving this specific problem, much research has been done regarding PSO coverage optimization. Kukunuru et al. [5] use PSO to solve a coverage problem in wireless sensor networks and achieve optimal solution. A centralized, off-line PSO-Voronoi algorithm [6] is used to minimize the area of coverage holes. The authors claim that PSO-Voronoi algorithm achieves close to ideal coverage, in spite of the time complexity of determining Voronoi polygons. Xu et al. [7] proposed a PSO algorithm to the coverage of a camera network in which the orientation of each camera can be freely adjusted while camera positions are fixed. Their results showed that the coverage can be greatly improved by adjusting the orientation of each individual camera.

However, most coverage problems are solved under the assumption that sensors have circular coverage so that sensor headings do not need to be considered. In addition, the execution time is not an issue to be considered for those problems. The computation intensive PSO algorithm requires parallel processing to reduce execution time. Tu et al. [8] use parallel computation models of PSO to solve a path planning problem. They compare the performance of the parallel computation models use multiple threads with regard to different communication capability among subgroups such as broadcast, star, migration and diffusion network topologies. Wang et al. [9] presented a parallel PSO algorithm using OpenMP to solve a facility location optimization problem. OpenMP is a library that is most suited for shared memory multi-core architecture. In their parallel algorithm experiment, two threads are executed, and results show that, in addition to less time, the parallel algorithm achieves better results than the serial algorithm. Similar PPSO research also has been done by [10] [11].

In this paper, the boundaries of the reachable sets are simplified (details of the reachable set are introduced in Section 4.). A one-step coverage problem is optimized by PPSO. The use of PPSO aims not only to reduce execution time but also to maximize the pursuers' coverage.

3. PRELIMINARIES OF PARTICLE SWARM OPTIMIZATION

PSO is initialized with a group of randomly generated potential solutions called particles. A particle is an encoding representation of a solution. All particles form a swarm. The goodness of a particle is evaluated by the fitness function. After evaluation, each particle stores "*pbest*"–the particle position with the best fitness value it has obtained so far. In addition, the global best particle, "*gbest*", i.e., the best of all particles, is also stored. In every iteration, each particle uses these two "best" particles to update its velocity and positions based on the following equations:

$$V_i(t+1) = w * V_i(t) + C_1 * rand() * (pbest_i - X_i(t)) + C_2 * rand() * (gbest - X_i(t)) \quad (1)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (2)$$

where $V_i(t)$ and $X_i(t)$ are the velocity and position of particle i at time t, respectively; $pbest_i$ is the particle with the best fitness value obtained so far by particle i; *gbest* is the global best particle with the best fitness value of all particles. $rand()$ is a random number between (0,1). C_1 and C_2 are constant, representing learning factors. w is the inertia weight representing the effects of previous velocity.

The equations can be explained geometrically in Fig. 1. Suppose that the current particle is at location $X_i(t)$, its next position $X_i(t+1)$ is the summation results of weighted vectors $V_i(t)$, $pbest_i$, and *gbest*, which function as inertia, self-consciousness, and collective consciousness, respectively. As a result, $X_i(t+1)$ is closer to the best position found so far.

Algorithm 1 is the pseudo code of the PSO algorithm.

4. PROBLEM DEFINITION

The pursuit-evasion coverage problem is defined under the following assumptions:

1. The evader runs slower but is more agile (small minimum turning radius) than pursuers.

2. All pursuers have the same-sized reachable set.

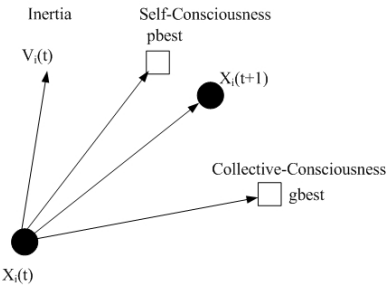


Figure 1. PSO particle motion

Algorithm 1 PSO algorithm

(1) Generate M number of initial particles
for $i = 1$ to MAX_LOOP **do**
  for $j = 1$ to M **do**
    (2) Calculate fitness value f_j of particle j
    (3) Update pbest and gbest
    (4) Update particle velocity according to equation (1)
    (5) Update particle position according to equation (2)
  end for
  (6) If the maximum iterations is reached or optimal solution is found, terminate the algorithm.
end for

3. Pursuers are not allowed to enter into $E's$ sensing range (within a radius of R_E) in the coverage optimization phase. Also suppose $R_E < V_p \Delta t$, where V_p is the pursuer's speed and Δt the pre-defined time interval.

4. Every pursuer knows the instantaneous location, orientation, and speed of the evader, and so does the evader.

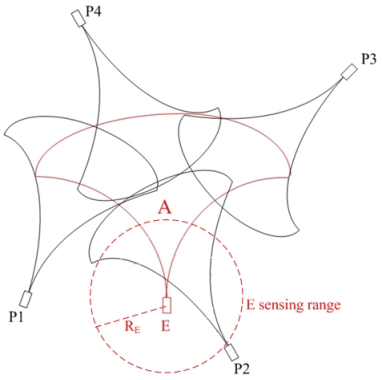


Figure 2. An example coverage problem with four pursuers ($P_1 - P_4$) vs. one evader (E)

Figure 2 is an example of four pursuers that successfully cover the area of A, the RS of evader E in Δt. Thus, E would be unable to find a "loophole" to escape.

The evader's RS is defined in Fig. 3, in which point B(x, y) is the farthest position the evader can reach after time Δt when the evader turns θ angle first, then goes straight.

$$x = R(1 - cos\theta) + (V_E \cdot \Delta t - R \cdot \theta) sin\theta \quad (3)$$

$$y = Rsin\theta + (V_E \cdot \Delta t - R \cdot \theta) cos\theta \quad (4)$$

where V_E is the maximum speed of the evader. R is the minimum turning radius. Detailed description of the reachable set can be found in [12]. Due to the evader's higher maneuverability, multiple pursuers need to cooperate to guarantee full coverage of the evader's RS before entering into the capture phase. The coverage problem can be represented as the following equations:

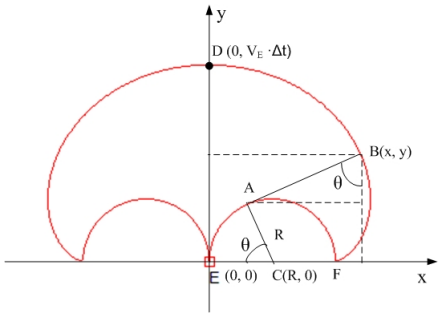


Figure 3. Definition of the evader's reachable set

$$\min_{X_{p_i}, Y_{p_i}, \theta_{p_i}} N \quad (5)$$

s.t.

$$S_E \bigcap \sum_{i=1}^{N} (\bigcup S_{p_i}) = S_E \quad (6)$$

where X_{p_i}, Y_{p_i}, θ_{p_i} are pursuer $p_i's$ x, y position and heading, respectively. N is the number of pursues to be minimized. S_E is the area of the $E's$ reachable set. S_{p_i} is the area of pursuer $p_i's$ reachable set. Equation (5) minimizes the number of pursuers needed for full coverage, subject to the condition of full coverage represented by equation (6).

Given the reachable set areas of the evader and pursuer, S_E and S_P, the minimum number of pursuers required for full coverage meets the following relationship:

$$N_p \geq \left\lceil \frac{S_E}{S_P} \right\rceil \quad (7)$$

The areas of the reachable sets can be calculated using integral calculus, but the calculation is omitted here due to space limitation.

5. THE PROPOSED APPROACH

A simple yet slow approach is to partition area A (Fig. 4) into tiny cells. A complete coverage means that any cell in

area A must be covered by the reachable set of one of the pursuers. Obviously, the smaller the cells, the more accurate the coverage calculation, but the more computational time it requires.

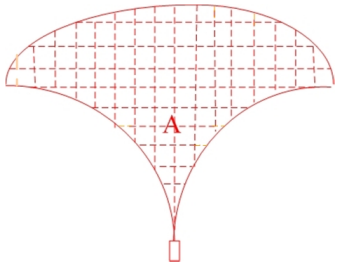


Figure 4. Area A is fragmented into small cells

5.1. Data Communication Topology

There are several kinds of communication network topology, but in this paper the master/slave model is used as it is more suitable for this problem. The particle with the global fitness value needs to be shared by all nodes, so it is more efficient to designate a master node to collect data and broadcast the global best. Particles in each node share the local best fitness value through shared memory; particles in different nodes share the global fitness value through inter-process communication.

Fig. 5 shows data flows and process functions. All processors evaluate a fraction number of particles. Each slave node sends the particle with local best fitness value (*pbest*) to the master node, which collects results and broadcasts the particle with the best fitness value (*gbest*) to slave nodes. Each slave node uses the *gbest* and *pbest* to update its position and velocity.

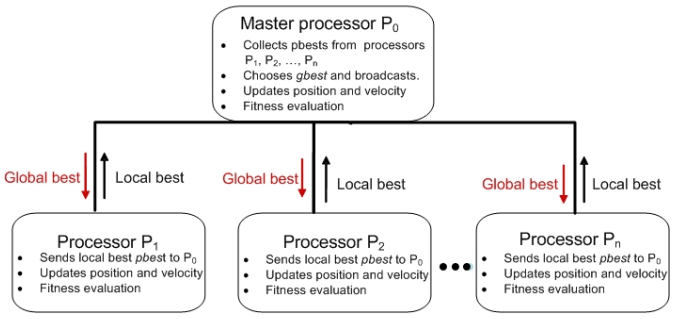


Figure 5. PPSO data flows and process functions

5.2. PPSO Task Partitioning

There are two data partitioning schemes to be considered before implementation.

1. Scheme A: The master node broadcasts every particle to slave nodes. Each slave node calculates the coverage of a subarea of E's RS, and sends back to the master its coverage result. The master totals the results for each particle.

2. Scheme B: Each processor/node processes a portion of total particles independently; each slave processor sends the particle with the best coverage to the master node.

Schedule B is used in the implementation of the PPSO. It has less communication overhead as the master node does not need to communicate with the slave nodes for each particle evaluation.

5.3. Swarm Initialization in PPSO

Swarm is usually initialized using uniform distribution approach over multidimensional search space, but it is not the efficient way for this problem. To prevent the algorithm from converging too slowly, it is necessary to discard those "unqualified" particle solutions in which at least one pursuer does not cover any of the E's RS or it is inside E's sensing range. The following approach can avoid many "unqualified" initial particles.

Suppose two pursuers are needed for a full coverage. For each particle, two points, c_1 and c_2, are randomly chosen from the area E's RS. The positions and headings of pursuers P_1 and P_2 can be randomly generated but are subject to two conditions: (1) c_1 and c_2 are inside at the RS of P_1 and P_2, respectively, and (2) P_1 and P_2 are outside of $E's$ sensing range. Fig. 6 is an example of how a particle is initially generated. In PPSO, each node is responsible for generating a sub-swarm, i.e., a portion of total particles. Suppose there are n pursuers, a

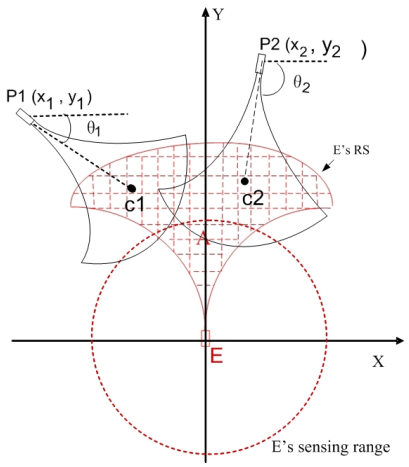


Figure 6. Initial particle generation

particle can be represented as an array shown in Fig. 7, where *fit* denotes the fitness value of the particle; x_i, y_i, and θ_i represent the x, y position and heading of pursuer i, respectively.

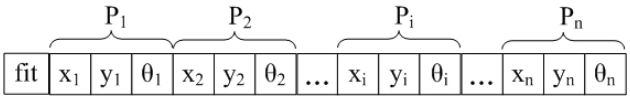


Figure 7. Particle representation

5.4. Fitness Function

The fitness value of a particle is the coverage defined as follows:

$$f = C * \frac{Number\ of\ covered\ cells}{Total\ cells} \tag{8}$$

$$C = \begin{cases} 0 & \text{if } \exists i,\ P_i \text{ is within } E\text{'s sensing range} \\ 1 & otherwise \end{cases} \tag{9}$$

Equation (9) penalizes a particle in which any pursuer gets into the evader's sensing range. The higher the fitness value is, the higher percentage the coverage, and the better the particle is.

Fitness evaluation is the most time-consuming part of the PSO. To check the percentage of coverage, all cells inside the reachable set A need to be checked. In each iteration of PPSO, different nodes evaluate their assigned particles independently. Only the global best fitness value needs to be transferred across different nodes.

5.5. The Proposed PPSO Algorithm

The PPSO is the parallelization of algorithm 1. Suppose there are M processors, p_0, p_1,...,p_{m-1}. p_0 is the master processor and p_1,..., p_{m-1} are slave processors. Suppose also that there are NUM_PT particles in each iteration. The algorithm does the following steps:

1. p_0 tries the minimum number of pursuers, N_p, which is calculated by equation (7).
2. Each node generates $\frac{NUM\_PT}{M}$ particles according to 5.3.
3. Each node including the master evaluates the fitness values of particles according to 5.4. Each slave node sends its *pbest* to p_0.
4. p_0 collects results from slave nodes and broadcasts *gbest*.
5. Each node updates its position and velocity according to equations (1)(2) .
6. If coverage=100%, then the algorithm terminates.
7. If loop $<$ Max_Loop, go to step 3.
8. If loop $>$ Max_Loop, then $N_p = N_p + 1$, go to step 2.

Fig. 8 is the flow chart of PPSO.

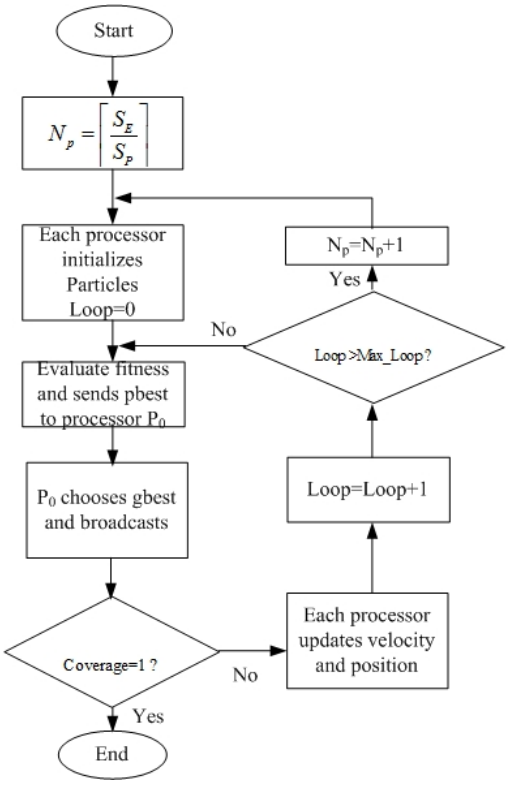


Figure 8. PPSO flow chart in master/slave mode

6. SIMULATION RESULTS AND DISCUSSIONS

The PPSO algorithm is implemented in C/C++ and MPI library. Two PE problems with different pursuer speed and minimum turning radius are tested in a cluster running in Redhat Linux with the following hardware configuration in each node:

Processor: Intel Xeon CPU 3.0GHz

Cache size: 2M

Memory size: 8M

Note that once the minimum number of pursuers is determined at the beginning, it will not be changed due to the fixed area of the reachable sets of both pursuers and the evader. For this reason, the execution time in the simulation is the time measured after the minimum number is determined. Also, due to the randomness mechanism of the PPSO algorithm and possible varying load on the cluster (which may slightly affect the execution time), all execution time is averaged over 10 runs.

6.1. Execution Time on a Two-Pursuer-One-Evader Coverage Problem

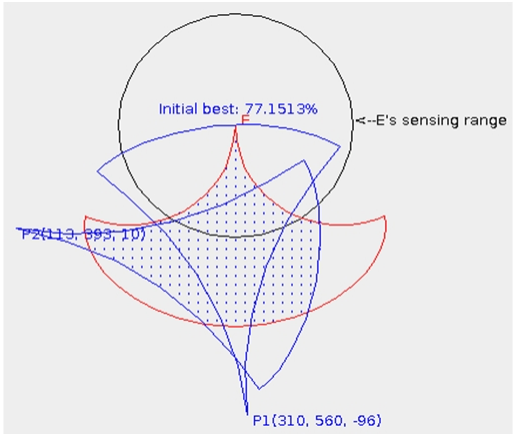
This PE coverage problem uses the following parameters:

E's position $(300, 200)$, E's heading: $PI/2$

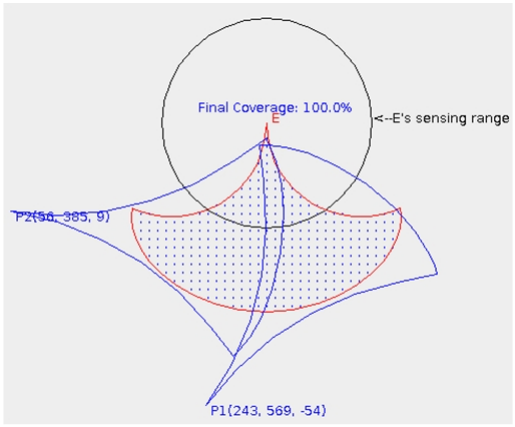
E's speed: 90, E's minimum turning radius: 90

E's sensing radius: 150
P's speed: 130, P's minimum turning radius: 300
P's sensing radius: unlimited
Swarm size: 20, Maximum iteration: 40
Time interval for the reachable set $\Delta t = 2$ seconds
Cell granularity: 8

Fig. 9 shows two graphic output of the coverage. Fig. 9(a) is the output of the best fitness particle chosen from the initially generated 20 particles. Fig. 9(b) is the result of the global best particle obtained by *PPSO* at iteration step 29. The data in the parentheses represents a pursuer's x, y position, and orientation.



(a) Best coverage of the initial 20 random particles



(b) Full coverage at iteration 29

Figure 9. An example of two pursuers and one evader

The progressive fitness values with the number of iterations is shown in Fig. 10. The algorithm terminates at step 29 when it achieves a full coverage, although the maximum number of iteration of PSO is 40.

As the swarm size is 20, the PPSO runs under the following number of processors: 1, 2, 4, 5, and 10. These numbers are chosen such that the load can be evenly partitioned into different processors, i.e., each processor is given the same

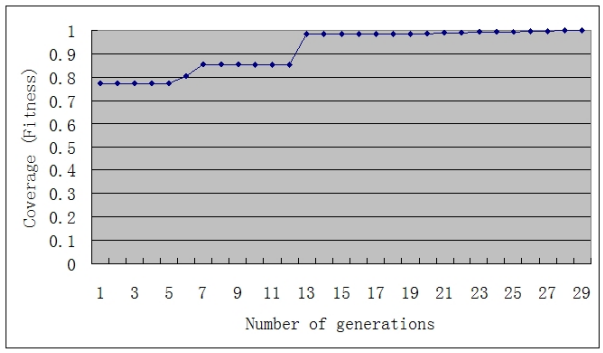


Figure 10. Fitness vs. number of generations

number of particles. The computation and communication time is shown in Fig. 11. The communication time increases and computation time decreases as the number of processors increases. Fig. 12 is the comparison of speedup. Given the granularity of cell size in this example, four processors is the "turning point" for the speedup. When the number of processors is greater than four, the communication overhead increases faster than the reduced computation time, causing the speedup to decrease. Generally, the smaller the cell granularity, the more processors are needed to perform fitness evaluation.

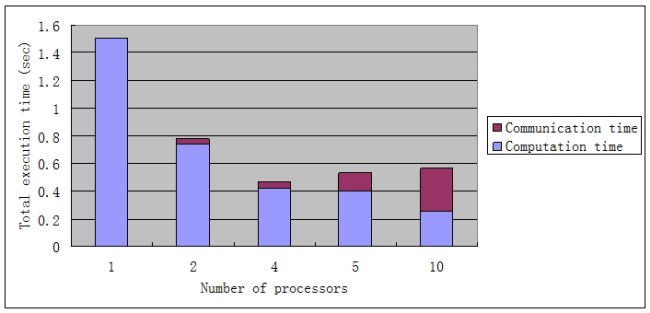


Figure 11. Computation time vs. communication time

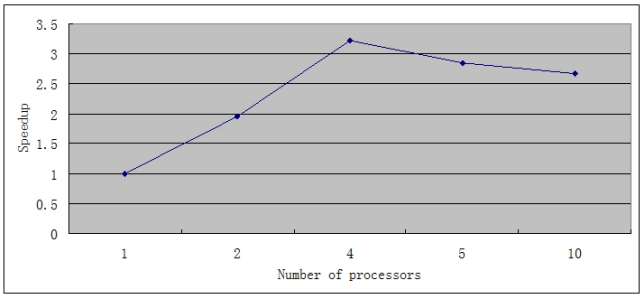


Figure 12. Speedup vs. the number of processors

6.2. Optimizing Fitness Function Using Data Locality Information

Tests show that it takes an average of 2*ms* to evaluate the fitness (or coverage) of a particle in the above example. The evader's reachable set is composed of 337 cells. To get the fitness value, each cell needs to be checked if it is covered by a pursuer's reachable set. A typical run of the sequential program shows that fitness evaluation takes up to 90% of the computation time.

The computation time can be reduced by changing the check sequence of pursuers based on cell locality - if a cell is covered by the reachable set of P_i, its next adjacent cell to be checked is more probably covered by P_i. Therefore, the next cell needs first to be checked with the reachable set of P_i. Doing so can eliminate many unnecessary checks done by fixed check sequence - from P_1, P_2,..., to P_n. Table 1 lists a comparison of the number of checks before and after using data locality. On average, the latter reduces 25.25% of total checks. The reason why the number of checks is greater than the total number of cells, 337, is that if a cell is not covered by one pursuer, it needs to be checked by another pursuer.

Table 1. Number of checks before and after using data locality (total cells: 337)

Particle No.	# checks (fixed sequence)	# checks (data locality)	Reduced checks
1	510	371	27.25%
2	502	397	20.92%
3	512	372	27.34%
4	513	369	28.07%
5	511	368	27.98%
6	512	372	27.34%
7	527	392	25.62%
8	509	370	27.31%
9	464	413	10.99%
10	518	372	28.19%
Average	508	380	25.25%

Fig. 13 shows that the fitness function based on data locality information can reduce computation time.

6.3. Communication Time Using Variable Length of Data Frame

At each iteration of the PPSO, the master node needs to broadcast the global best particle including a fitness value and all pursuers' locations and orientations. The length of the particle is $(3 * num\_p + 1)$ in float data type, where num_p is the minimum number of pursuers needed.

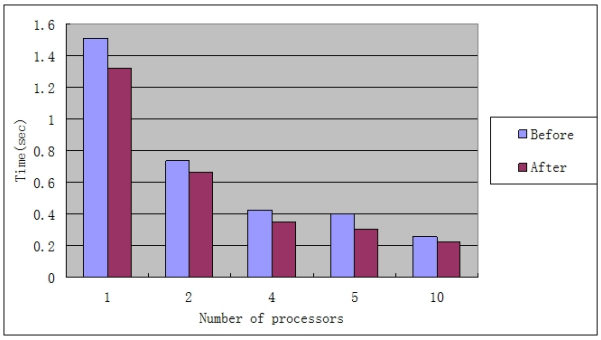


Figure 13. Computation time before and after using data locality

Actually, in many iterations the best fitness value is not updated (shown in Fig. 10). For this reason, instead of using fixed length of data frame, communication data can be reduced at iterations when the best fitness value collected from other processors is less than the existing global best. If so, the master node only needs to broadcast a tag (an integer) notifying all slave nodes that the global best is unchanged.

Nevertheless, using variable length of data causes extra overhead. Before broadcasting actual data, the master node has to add one more broadcast call notifying the length of the next incoming communication data. In contrast, this is not needed in the fixed-length data communication since all nodes assume that the length is $(3 * num\_p + 1)$.

To better test the algorithm, the second test case assumes slower pursuer speed, smaller minimum turning radius, and an increased number of maximum iterations listed below:

P's speed: 110

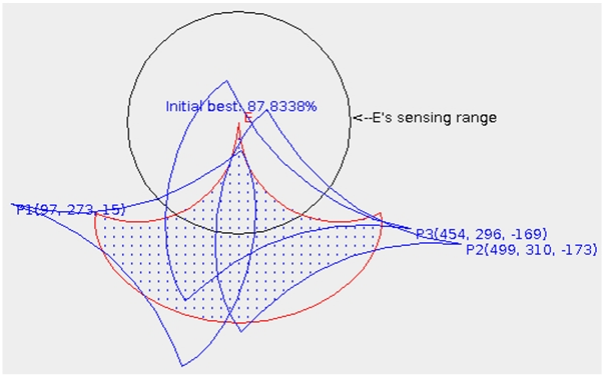
P's minimum turning radius: 220

Maximum number of iteration of PPSO: 60

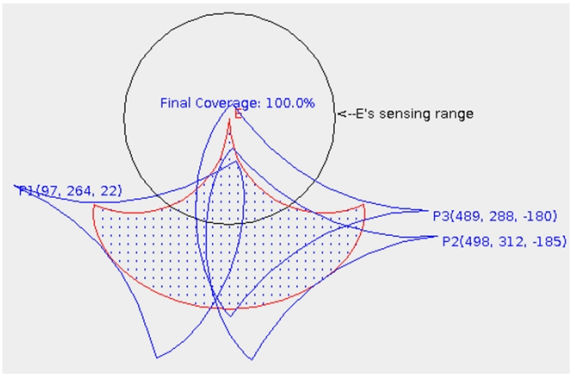
Other parameters remain the same as test case one. As the pursuers run slower, three pursuers are needed to fully cover the evader's RS shown in Fig. 14. In this example, the PPSO algorithm achieves a full coverage at iteration number 53. Figure 15 compares the communication time between fixed and variable communication data length. The variable length approach performs worse under 2 and 4 processors. The reason could be that the time saved for shorter message cannot offset the overhead caused by adding one more broadcast function call. However, as the number of processors continues increasing, sending shorter messages when the global best is not updated reduces overall communication time. Theoretically, under a given number of processors, the more pursuers are needed, the longer the length of the particle, the better the variable length approach performs.

7. CONCLUSIONS AND FUTURE WORK

A Parallel Particle Swarm Optimization approach is tested on a cluster to solve the PE coverage problem. Simulation re-



(a) Best coverage of the initial 20 random particles



(b) Full coverage at iteration 53

Figure 14. An example of three pursuers and one evader

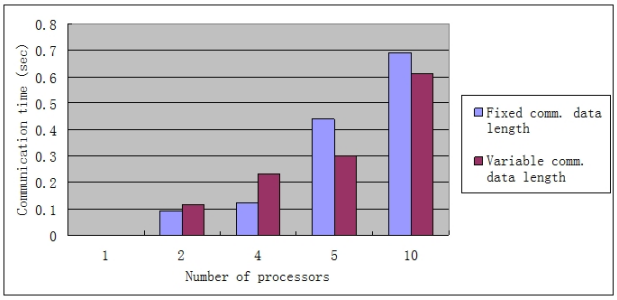


Figure 15. Communication time between fixed and variable data frame

sults show satisfactory speedup, the communication and computation time with the number of processors, as well as the graphic output of two test cases with different parameter values for the pursuers. In addition, the optimized fitness function by changing checking sequence based on cell locality reduces computation time. Compared with the fixed length of communication data, the variable length approach performs better in reducing communication time when the number of processor increases.

Our future work will focus on theoretical analysis of the algorithm in terms of time complexity, and an improved algorithm by adding the MPI multi-threading mode, that is, in addition to parallelizing the algorithm between processors, each processor can spawn multiple threads to do another layer of parallelization.

REFERENCES

[1] R. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory," in *Proceedings of the sixth international symposium on micromachine and human science*, 1995, vol. 1, pp. 39–43.

[2] A. Farmahini-Farahani, M. Laali, A. Moghimi, S. Fakhraie, and S. Safari, "Mesh Architecture for Hardware Implementation of Particle Swarm Optimization," in *International Conference on Intelligent and Advanced Systems*, November 2007, pp. 1300–1305.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[4] C. F. Chunga and T. Furukawa, "A Reachability-Based Strategy for the Time-Optimal Control of Autonomous Pursuers," *Engineering Optimization*, vol. 40, no. 1, pp. 67–93, January 2008.

[5] N. Kukunuru, B. R. Thella, and R. L. Davuluri, "Sensor Deployment Using Particle Swarm Optimization," *International Journal of Engineering Science and Technology*, vol. 2, 2010.

[6] N. A. B. A. Aziz, A. W. Mohemmed, and B. S. D. Sagar, "Particle Swarm Optimization and Voronoi Diagram for Wireless Sensor Networks Coverage Optimization," in *2007 International Conference on Intelligent and Advanced Systems*, 2007, pp. 961–965.

[7] Y. Xu, B. Lei, and E. A. Hendriks, "Camera Network Coverage Improving by Particle Swarm Optimization," *EURASIP J. Image and Video Processing*, vol. 2011, 2011.

[8] K.-Y. Tu and Z.-C. Liang, "Parallel Computation Models of Particle Swarm Optimization Implemented by Multiple Threads," *Expert Systems with Applications*, vol. 38, pp. 5858–5866, May 2011.

[9] D. Wang, C.-H. Wu, I. A., D. Wang, and Y. Yan, "Parallel Multi-Population Particle Swarm Optimization Algorithm for the Uncapacitated Facility Location Problem Using OpenMP," in *IEEE 2008 World Congress on Computational Intelligence*, June 2008, pp. 1214–1218.

[10] N. C. Chauhan, D. Aggarwal, R. Banga, A. Mittal, and M. V. Kartikeyan, "Parallelization of Particle

Swarm Optimization and Its Implementation on Scalable Multi-Core Architecture," in *IEEE International Advance Computing Conference (IACC 2009)*, Patiala, India, March 2009, pp. 392–397.

[11] B.-I. Koh, Byung-Il, George, A. D, Haftka, R. T, Fregly, and B. J, "Parallel Asynchronous Particle Swarm Optimization," *International Journal for Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578–595, 2006.

[12] P. J. Wong and A. J. Korsak, "Reachable Sets for Tracking," *Operations Research*, vol. 22, no. 3, pp. 497–509, May-Jun 1974.