Parallel solution of maze optimal path based on ant colony algorithm

Liu Yu
College of Mechanical and Control Engineering
Guilin University of Technology
Guilin, China
e-mail: lewis_5709@163.com

Xiao Yi
College of Information Science and Engineering
Guilin University of Technology
Guilin, China
e-mail: louisxcode@yahoo.com

***Abstract*—In order to solve efficiency of maze optimal path problem based on ant colony algorithm, we proposed a CUDA platform parallel programming model, executing the algorithm with GPU that significantly improve the calculation efficiency. A parallel matrix design method was introduced and the ant colony searching process was changed into parallel matrix operations. To reduce the access overhead and increase algorithm running speed, data were rationally organized and stored. Experiments of different scale maze matrix test show that speedup will be increased with the expansion of the matrix size. In our experiments, the maximum speedup is about 11.5.**

Keywords- ant colony algorithm; Maze problem; optimal path; parallel process;

I. INTRODUCTION

The maze problem is a typical search and traversal problem of graph. Many intelligent problems could be transformed into the maze optimal path problem, for example: chess game, strategy decision, robot path planning. The space and time complexity of the traditional algorithm will be growing exponentially with the expansion of the matrix size and complexity. So it's very hard using it to resolve the large scale cases. According to the character of maze optimal path problem, Reference [3] the application of ant colony algorithm to solve maze problem divide ants into two groups and set three life cycles, then get the solution by iterative cycle. But with the expansion of the matrix, the loops of iterative cycle will be greatly increased. Reference [4] an improvement to [3] algorithm set the distance to gate as basic parameter k, and then distributes ants at the position k. It will decrease the count of loops by ant colony algorithm. MMAS (Max-Min Ant System) algorithm is also applied to this process to avoid convergence too early. That algorithm avoids the search by every ant's taboo list, but it needs judgment to many conditions on each loop. This article wills analysis those algorithms mentioned before and post a parallel solution to solve the maze optimal path problem based on ant colony algorithm. It uses parallel computation for the data in each iterative cycle the processing speed of the algorithm mentioned in [4].

II. PARALLEL PROGRAMMING MODEL

In November 2006, NVIDIA introduced CUDA, a general purpose parallel computing architecture – with a new parallel programming model and instruction set architecture – that leverages the parallel compute engine in both CPU and GPU to solve many complex computational problems in a more efficient way than only CPU. In the CUDA environment, CPU is the main memory called host and GPU is a co-processor called device. In this model, CPU responsible for carrying out the logic transaction processing and serial computing, GPU is responsible for the implementation of highly threaded parallel processing tasks. CPU has the memory of the host side called host memory and GPU has the memory of the equipment side called device memory. CUDA provides specific API functions to operate the device memory, the operation includes the initialization of the memory space, open up and release, as well as host memory and device memory data transfer. Fig. 1 is CUDA parallel programming model. It shows the function running on the GPU is called the kernel function. A kernel function just a part of CUDA program which can be executed in parallel. The complete CUDA program is composed by the serial processing function on host and the parallel processing kernel function on device. The complete program execution in accordance with the order of the corresponding statement in turn.

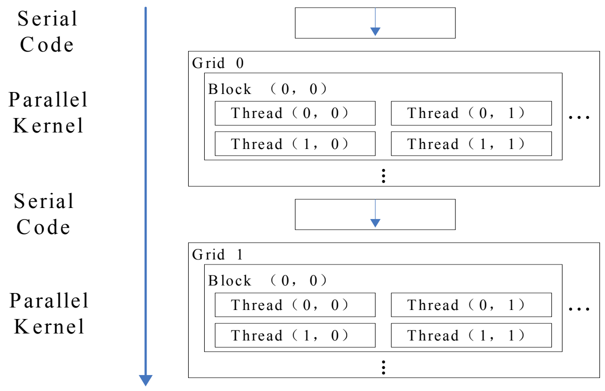

Figure 1. Parallel programming model

III. THE IMPLEMENTATION OF ALGORITHM OF PARALLEL COMPUTING

A. *The improvement to existing algorithm*

Reference [4] defining the following data structure for maze. It prevents the redundant visit to taboo list and the space cost for each path position visited by each ant.

struct{

int pheromones; //pheromone of this position, Initialized as p. If blocked, set as 0

int I_dist; //the distance from this position to entrance of maze, Initialized as ∞

int O_dist; //the distance from this position to exit of maze, Initialized as ∞

}Maze[M][N] //M × N maze

In order to improve the efficiency of the algorithm to solving maze optimal path problem and to make algorithm more suitable to parallel computing, we make such changes to the algorithm in [4]: 1. Add a "Wall" data structure to maze data structure. It has the same data structure with maze data. It is designed for reducing the judgment to boundary conditions by doing condition judgment to obstacle information. 2. The updating maze path information named "Reclamation". Execute the reclamation before ants moving, so that the amounts of path updating will quickly moving towards to a minimum value. 3. Add ants' pheromones matrix and moving position matrix. They're used to store the pheromones of 8 directions from every ant's current position and the next selected position. These matrices will be used at parallel computing.

B. *Algorithm parallelization solving*

According to Amdahl's law, the impact of the overall system performance depends on proportion and acceleration efficiency of the program which can be optimized. The reclamation and ants moving is the main part of original algorithm, collectively called ant searching. Ants move based on four judgment conditions for up to a position in around 8 directions. Experiment on 1024 order matrix, we test 6 single searching process to calculated the time of original serial algorithm program data initialization and ants' determining the start point, the time of single searching for the optimal path and the time of roulette method. The test results show in Tab. 1.

The test results show that searching process total spend 96.5% running time of the original serial algorithm. The roulette method accounted for 53.9% of the searching process time, 52.1% of the total running time. Parallelizing searching part of the algorithm can dramatically improve the speed of whole algorithm. According to Amdahl acceleration formula: $s = \frac{p}{1+f(p-1)}$, when $p \to \infty$ can be inferred $s = \frac{1}{f}$. That means acceleration limit of the algorithm in the search process is 20.

C. *Algorithm Parallel Implementation*

According to the original algorithm, each ant carries out the following activities in turn: reclamation, moving base on the rules, searching the optimal path. When the optimal path exists ant update the pheromone. Reclamation and optimal path pheromone updating process involves parallel writing issues. Other parts have parallelism in the whole process. Based on the [5], if we make ants as GPU threads, will get lower efficiency.

Each ant in the reclamation process need to get 8 around direction pheromone from the maze matrix and to store in the ants' pheromone matrix: Round matrix. Ants' next location information was stored in the moving position matrix: Result matrix. In order to improve the efficiency and reduce the GPU parallel access memory delay, Round matrix design were designed with following rules.

The maze Matrix is a matrix of order M. We assume the rows of the Round matrix is m, the column is n. According to original algorithm, the number of ants is 2M. Introducing the parameter χ, we can get $2M/2^{\chi} = 2^{\chi} \times 9$. Compute the parameters $\chi = \left\lfloor \frac{1}{2}\log_2(2M/9) + 0.5 \right\rfloor$,so $m = \lceil 2^{\chi} \times 9 \rceil$ and $n = \lceil 2M/2^{\chi} \rceil$.

Each ant has 9 spaces in the matrix, called ant information space. The information store the random number and ant's 8 direction pheromone around order by column. The random number also has the function of a flag to indicate whether the ant needs the next location information. Every GPU thread based on the thread number corresponds to an ant space. Threads parallel read Round matrix ant's information space and use Roulette method to calculate the probability of each direction. Threads based on the random number determine the next location of the moving and write to the corresponding positions in Result matrix. Fig. 2 is program structure diagram. The entire parallel computation process is shown in Fig. 3.

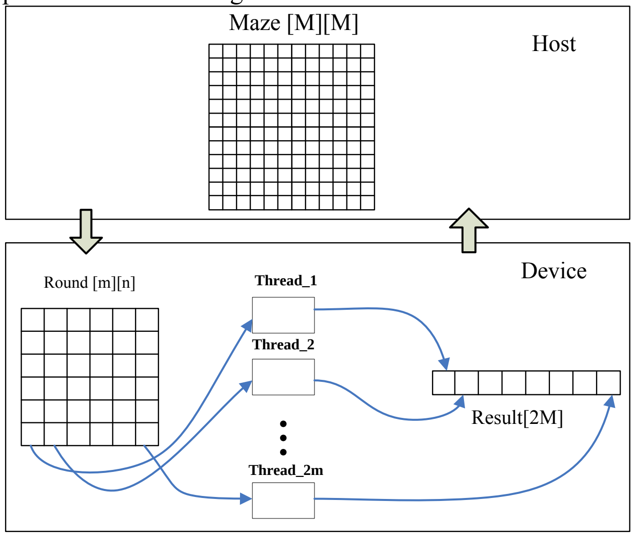

Figure 2. Program structure diagram

D. *Algorithm Description*

Step 1: Data initialization. Host initialize maze matrix, counter and constant coefficients. Ants initialize starting point based on the value of k. Device initialize and allocate space for the Round matrix and Result matrix.

Step 2: The Ant Colony reclaim 8 around directions and update maze matrix. Then save the Round matrix in corresponding ant information space.

Step 3: Ant Colony moves base on the rules. When one of the first three conditions is met, the flag will be marked on Round matrix in corresponding ant information space.

Step 4: Transfer the data from host to device.

Step 5: Create GPU multiple threads to parallel compute the next location information based on Round matrix. Information is stored in the Result matrix.

Step 6: Transfer the data from device to host. Ant Colony move base on the information on Result matrix.

Step 7: Search the optimal path. If optimal path exist, update the pheromone on the optimal path.

Step 8: To determine whether the iteration parameters meet preset value. If it's true, output the optimal path. Otherwise, reset the Ant Colony information and go to Step 2.

IV. Experimental Results

We used a PC with Windows XP operation system that has one Intel Core i3 3.3 GHz processor and one NVDIA GeForce 405. According to the original algorithm parameter settings, Tab. 2 shows the experimental results of single search action on 1024×1024 matrix size.

Speedup refers to the ratio of the time required by the application running on the CPU time and on the GPU. From the data in Tab. 2, parallel algorithm running in 1024 × 1024 order matrix get the speedup of 11.5. Although still a gap with the theoretical value, but greatly reduced the searching time. The data in Fig. 4 shows searching time of the optimal path based on different maze matrix size from 64 × 64 to 1024 × 1024. The serial and parallel algorithms were used 20 times, respectively, for each data test. Original algorithm as the matrix order increases, program run time of rapid growth. The proposed algorithm is its control of linear growth and faster computing speed.

V. Conclusion

This paper post a parallel solution to solve the maze optimal path based on ant colony algorithm. Compared with algorithm used in the [4], it has the following advantages: 1) adding the maze boundary to reduce the judgment of the boundary of the ant colony can improve the efficiency of ant colony searching; 2) individually processing the ant colony reclamation can improve the convergence rate of a single searching; 3) making use of GPU for parallel computation can improve the operating speed and get higher speedup, and remain the cost negligible at the same time.

Acknowledgment

This work has been supported by The National Natural Science Foundation of China under research project 41264005 and also supported by The Guangxi department of education under the research project 201102ZD018.

References

[1] Colorni A, Dorigo M, and Maniezzo V, "An investigation of some properties of an ant algorithm," Of the Parallel Problem Solving from Nature Conference (PPSNp92). Brussels. Belgium. Elsevier Publishing, pp. 509–520, 1922.

[2] Dorigo M, Maniezzo V, and Colorni A, "The ant system: Optimization by a colony of cooperating agents," IEEE Transactions on System, Man and Cybernetics Part B, vol. 26, pp. 29–41, 1996.

[3] HU Xiao-bing and HUANG Xi-yue, "Ant colony algorithm in maze optimal path problem," Computer Simulation, vol. 22, pp. 114–116, 2005. *(references)*

[4] ZHANG Gong-jing and XU xi-jun, "The application of ant colony algorithm to solve the maze optimal path," Journal of Qingdao University (Natural Science), vol. 21, pp. 61–65, 2008. *(references)*

[5] LI Jian-ming, HU Xiang-pei, PANG Zhan-long and QIAN Kun-ming, "A fine-grained parallel GPU accelerated ant colony algorithm," Control and decision-making, vol. 24, pp. 1132–1136, 2009.

TABLE I. Different parts of the running time of the original algorithm

Number of searching	1	2	3	4	5	6	Average
Initialization	0.656	0.594	0.609	0.641	0.594	0.578	0.612
Searching	18.219	15.734	17.891	17.922	19.547	13.110	17.071
Roulette	10.014	10.014	9.582	10.126	9.421	6.810	9.205
Program running	18.875	16.328	18.500	18.563	20.141	13.688	17.683

TABLE II. time of the Parallel algorithm

Number of searching	1	2	3	4	5	6	Average
Initialization	0.75	0.656	0.641	0.547	0.593	0.75	0.656
Searching	0.875	0.922	0.859	0.859	0.875	0.875	0.878
Program running	1.625	1.578	1.500	1.406	1.468	1.625	1.534

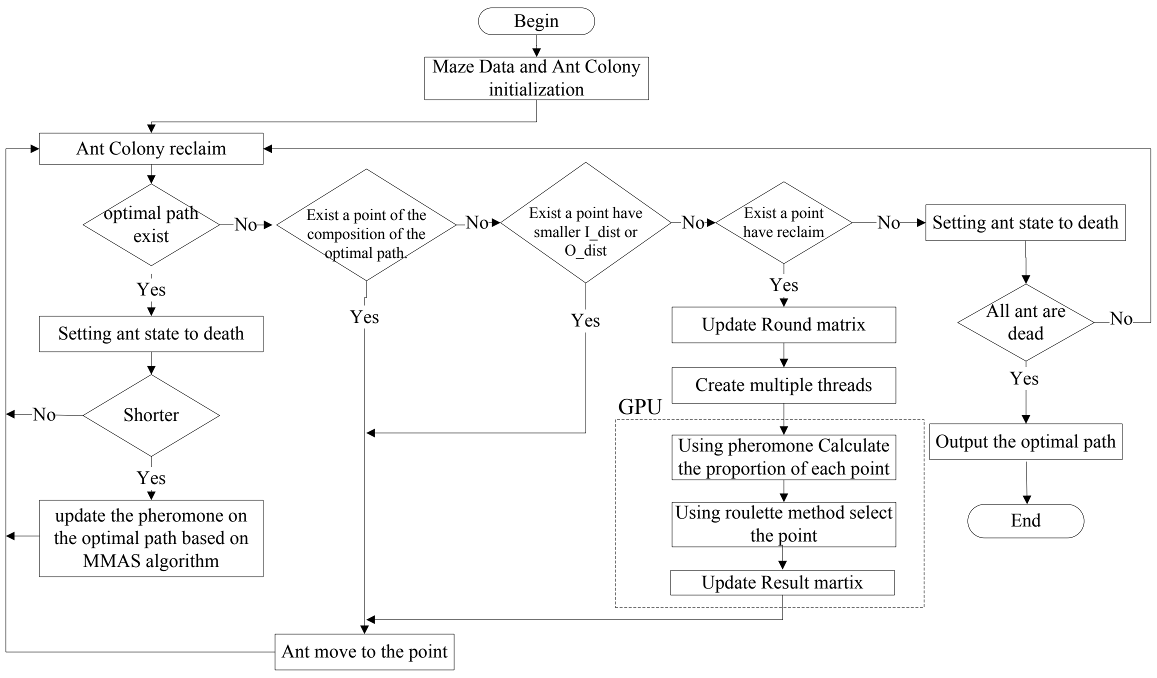
Begin
Maze Data and Ant Colony initialization
Ant Colony reclaim
optimal path exist
No
Exist a point of the composition of the optimal path.
No
Exist a point have smaller I_dist or O_dist
No
Exist a point have reclaim
No
Setting ant state to death
Yes
Setting ant state to death
Yes
Yes
Yes
Update Round matrix
All ant are dead
No
Create multiple threads
Yes
GPU
No
Shorter
Using pheromone Calculate the proportion of each point
Output the optimal path
Yes
Using roulette method select the point
End
update the pheromone on the optimal path based on MMAS algorithm
Update Result martix
Ant move to the point

Figure 3. Algorithm parallel flow chart

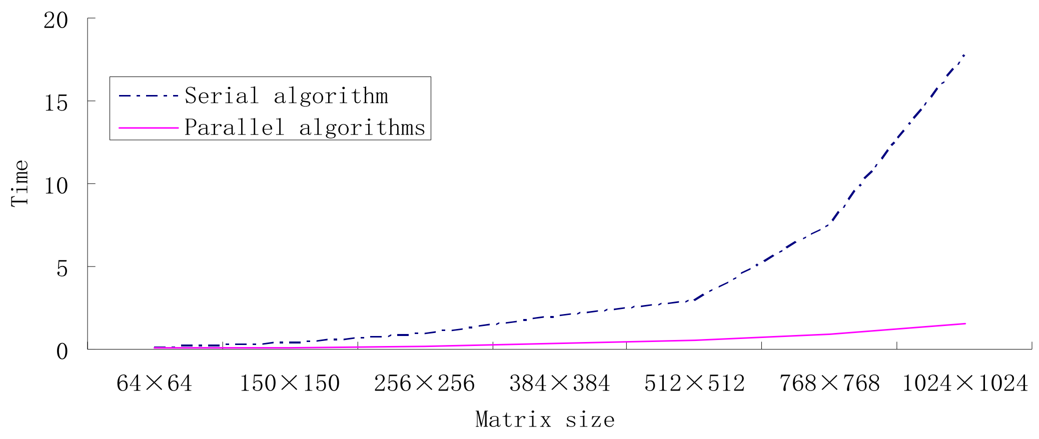

Figure 4. Contrast of the algorithm running time