# *MAX-MIN* Ant System on GPU with CUDA

Hongtao Bai[a,b], Dantong OuYang[a,b], Ximing Li[a,b], Lili He[a,b], Haihong Yu[a,b]
*a. College of Computer Science and Technology, Jilin University, 130012, China;*
*b. Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, 130012, China)*
*helili@jlu.edu.cn*

## Abstract

*We propose a parallel MAX-MIN Ant System (MMAS) algorithm that is suitable for an implementation on graphics processing units (GPUs). Multi ant colonies with respective parameter settings are whole offloaded to the GPU in parallel. We have implemented this GPU-based MMAS on the GPU with compute unified device architecture (CUDA). Some performance optimization means for kernel program of GPU are introduced. Experimental results that are based on simulations for the traveling salesperson problem are presented to evaluate the proposed techniques.*

## 1. Introduction

*MAX-MIN* Ant System (*MMAS*), which is one of the most efficient Ant Colony Optimization (ACO) algorithms, has been introduced by Stutzle and Hoos [1]. It is widely used in solving traveling salesman problem (TSP), routing problem, scheduling problem and some other combinatorial optimization problems [2,3,4]. Recently, cluster analysis, mixed-integer programming, text feature selection and so on [5,6,7] benefit from *MMAS* or other ACO algorithms.

Unfortunately, all ACO algorithms could only approach but not guarantee to get the theoretic optimal solution for solving optimization problems, because of its random search techniques. Discovering a better solution in regular time is always meritorious work for the algorithm and its implementation. Performance of ACO will decrease rapidly with the growth in scale of the problem. What's more, it is highly sensitive to parameter settings. Aiming at these inherent shortages, many scholars have proposed a variety of measures in pheromone update and tours construction ways to improve the original algorithms [8,9]. Besides, Parallelization of ACO based on high-performance computing platform is a good choice to improve the solution quality and speed it up [10,11].

The rapid increase in the performance of graphics hardware, coupled with recent improvements in its programmability, have made graphics processing units (GPUs) a compelling platform for computationally demanding tasks in a wide variety of application domains [12]. In this paper, we propose a parallel *MMAS* algorithm that is suitable for GPU and implement it with the newest generation GPU computing circumstance compute unified device architecture (CUDA) [13]. Multi ant colonies with respective parameter settings synchronously evolve and each ant within every ant colony tours in parallel. Simulation experiments show its effectiveness and high efficiency.

## 2. Multi ant colonies on GPU

The behavior of *MMAS* suggests that good parameter settings are those that find a reasonable balance between a too narrow focus of the search process and a too weak guidance of the search, while the bad parameter setting is not able to focus the search on the most promising parts of the search space. However, the optimal parameter settings can't be accurately set by any mathematic method fitting to all kinds of problems. Dorigo and Stuzle [8] described appropriate setting scope of parameter $\alpha, \beta, \rho$ based on a lot of experiments. In this paper, we design the strategy that multi ant colonies with different parameter settings are offloaded to the GPU and ants in every ant colony are also parallelized. The best value of all sub ant colonies' optimal solutions is the final solution. To clearly illustrate our strategy, we discuss *MMAS* on standard TSP problem below. The distribution of multi ant colonies is shown as Fig.1.

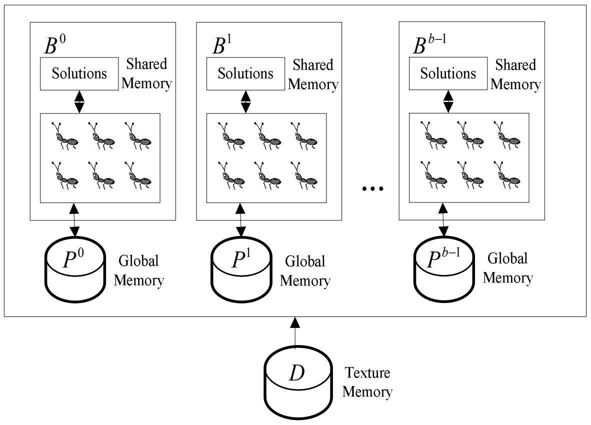

Fig.1 Distribution of multi ant colonies on GPU

Supposing there are $b$ sub ant colonies, each ant colony, corresponding to a thread block $B^i(i=0,1,...,b-1)$ of CUDA, has its private pheromone matrix $P^i(i=0,1,...,b-1)$ and shares one common distance matrix $D$. According to the memory architecture of CUDA and features of the TSP problem, every pheromone matrix $P^i$ is stored in Global Memory, while distance matrix D stay at read-only Texture Memory with cache. The current solution of every ant is put in Shared Memory, which is in favor of the reduction manipulation (Step 2 of infra).

GPUs are traditionally designed for gaming applications and remain many restricts for GPGPU. First, GPU can't self-start and I/O so that it must be a coprocessor of CPU. Next, the architecture and programming model are largely optimized for graphics processing, where data items are processed uniformly and control flows consist of a few rather fixed stages. Besides, a series of optimization means must be carefully considered to win high performance. Thus, we plan the framework of GPU-based *MMAS* (*GMMAS*), which is achieved by cooperation between CPU and GPU, as follows:

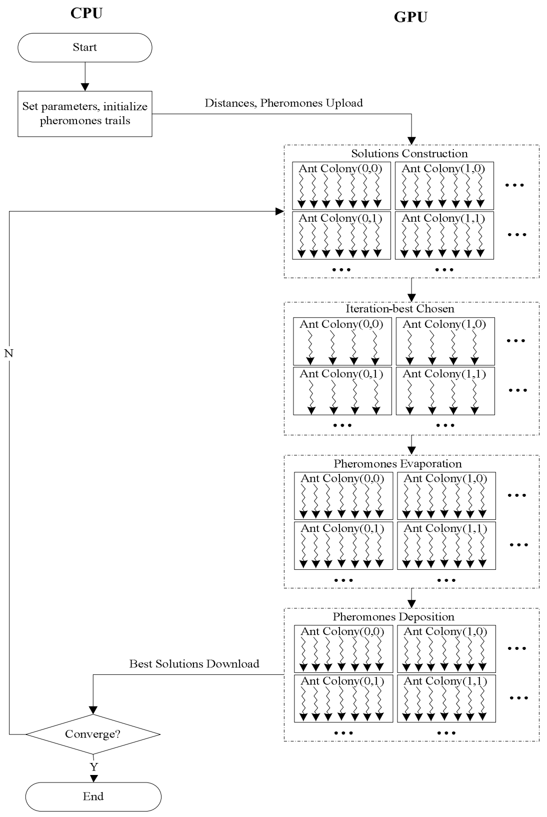

Fig.2 Framework of *GMMAS*

As shown in Fig.2, almost all steps of *GMMAS* run on GPU, while CPU only controls iteration process. We explain it for the TSP ($n$ is the size of problem, and $m$ is the number of ants in a sub ant colony):

Step 1: Parallel solutions construction. Each thread block has $m$ threads (each ants corresponds to one thread), and each thread tours on its own.

Step 2: Iteration-best solution chosen. Each thread block has $m$ threads, which find the iteration-best solution out using reduction technique from $m$ solutions within one ant colony.

Step 3: Pheromone volatilization. Each thread block has $n$ threads (each city corresponds to one thread), and each thread volatilizes pheromone on one edge.

Step 4: Pheromone deposition. Each thread block has $n$ threads, and each thread deposits pheromone on $w$ edges, where $w$ is the number of ants with deposition function.

## 3. Experiments

The TSP is an extensively studied problem in the literature and also plays an important role in ACO research: it is a standard test bed for new algorithmic ideas——a good performance on the TSP is often taken as a proof of their usefulness [8]. In this sub section, the TSP benchmarks from the library TSPLIB [14] are used in the experiment to compare *GMMAS* with serial *MMAS* on CPU. Serial *MMAS* has three versions, whose are *MMAS*1( $\beta = 2, \rho = 0.02$ ), *MMAS*2 ( $\beta = 2, \rho = 0.06$ ), *MMAS*3 ( $\beta = 5, \rho = 0.04$ ).While, $\rho$ of *GMMAS* is set to be (0.02, 004, 0.06 and 0.08), and $\beta$ is (2,3,4,5). Other parameters of *GMMAS* and *MMAS* are $m = n$ , $\alpha = 1$ , $\pi_0 = 1/\rho C^{nn}$ .Thus, *GMMAS* starts up 16 sub ant colonies, and each ant colony has $m$ ants.

All experiments are performed on a Dell Compatible PC with an AMD Athlon Dual Core Processor 3600+, 2GB main memory and Geforce 8800GTX graphic card, 1.35 GHz engine clock speed, 768 MB of device RAM, and 128 stream processors, organized into 16 multiprocessors.

### 3.1. Evolution of iteration-best solutions

We investigate the evolution process of iteration-best solutions of *GMMAS* and three *MMAS* versions on berlin52. The results are shown:

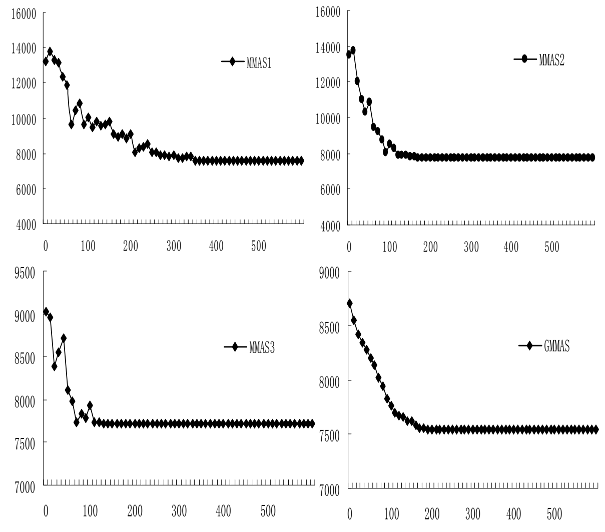

Fig.3 Evolution process of iteration-best solutions (berlin52)

Fig.3 proved that the parameter settings have great influence on the behavior of all algorithms. The algorithm will obtain good solutions if $\beta$ is large because of the influence of distance heuristic factor. On the other hand, it is easy to force the algorithm to converge much faster by a higher pheromone evaporation rate $\beta$ . Nevertheless, it may lead to the search stagnation state. *GMMAS* uses the combination of key parameter settings in a large range so that it not only accelerates the convergence process, but also draws a smooth curve of iteration-best solutions.

### 3.2. Solution quality

Table 1 shows the experimental results on six TSP problems and we perform 50 trials on each problem and adequate iteration numbers (listed below every problem) to ensure sufficient convergence on each trial. In order to roundly depict the behaviors of all algorithms, we give best, worse and average solutions in 50 trials.

Table 1 Solution quality

| Problem | Algori-thm | Best solution | Worse solution | Average solution |
|---|---|---|---|---|
| eil51 (1000) | *MMAS1* | 428.87 | 431.10 | 429.68 |
| | *MMAS2* | 428.87 | 441.73 | 435.11 |
| | *MMAS3* | 428.98 | 430.74 | 429.54 |
| | *GMMAS* | 428.87 | 428.98 | 428.89 |
| Bier 127 (1000) | *MMAS1* | 120416.78 | 121524.26 | 120990.85 |
| | *MMAS2* | 120433.04 | 121876.73 | 121119.88 |
| | *MMAS3* | 118715.50 | 119283.90 | 118906.70 |
| | *GMMAS* | 118293.40 | 118872.50 | 118696.60 |
| d198 (1500) | *MMAS1* | 16375.55 | 16839.97 | 16578.62 |
| | *MMAS2* | 16066.21 | 16291.51 | 16157.29 |
| | *MMAS3* | 15938.93 | 16021.41 | 15971.65 |
| | *GMMAS* | 15884.77 | 15973.27 | 15926.97 |
| pr264 (2000) | *MMAS1* | 50058.39 | 50502.82 | 50323.67 |
| | *MMAS2* | 49405.42 | 51252.59 | 49952.95 |
| | *MMAS3* | 49244.48 | 49303.75 | 49261.73 |
| | *GMMAS* | 49134.99 | 49144.69 | 49136.93 |
| lin318 (2500) | *MMAS1* | 42271.77 | 42701.99 | 42550.36 |
| | *MMAS2* | 43879.35 | 44063.617 | 43919.97 |
| | *MMAS3* | 42613.52 | 43437.33 | 42833.94 |
| | *GMMAS* | 42204.47 | 42469.59 | 42313.20 |
| rd400 (3000) | *MMAS1* | 15971.75 | 17167.51 | 16368.99 |
| | *MMAS2* | 16048.83 | 17167.51 | 16338.47 |
| | *MMAS3* | 15498.75 | 16063.51 | 15706.30 |
| | *GMMAS* | 15324.65 | 15423.56 | 15377.25 |

Both *MMAS* and *GMMAS* are whole according to the standard ACO algorithms framework, which don't undertake any special optimization for TSP. Even so, *GMMAS* has ability to gain high solution quality. For instance, the best value of pr2640 solved by *GMMAS* is 49134.99, which is equal to the best integer solution

(49135) gotten by all kinds of algorithms for TSP in TSPLIB.

### 3.3. Performance

*GMMAS* is a parallel algorithm and has 16 times workload than serial *MMAS*. We test the speedup under different size of problems:

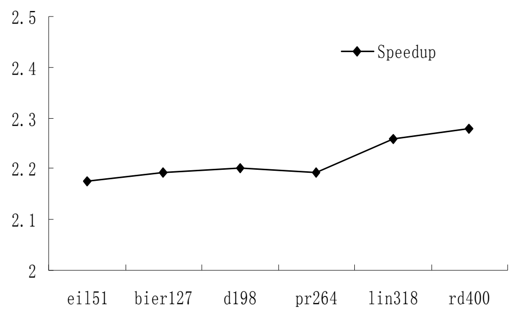

Fig.4 Iteration performance test

Speedup of *GMMAS* archives more than 2. Actually, GPU can get 32 upwards times faster than CPU under the same workload[1]. The stability of speedup shows that *GMMAS* has been fully parallelized and our performance optimization means to the GPU are effective.

## 4. Conclusion

In this research, we illustrate that ACO algorithms have several features that make the algorithm particularly suitable for GPU implementation. The CUDA technology used in our experiments is modern GPU architecture, which is adopted by many NVIDIA GPUs. As current trends indicated, future GPU designs, also based on general purpose multiprocessors, will offer even more computational power. Our primary purpose in this research is to prove that developing GPU-aware ACO software is possible and useful. More GPU-based approach should be further explored such as adaptive exchanging strategies suited to GPU. Another research direction is to employ GPU to other swarm intelligence algorithms.

**Acknowledgements:** This research is supported in part by the National Natural Science Foundation of China (60773097; 60873148) and Jilin Province Science and Technology Development Plan (20060532; 20080107).

[1] **The elapsed time for solving rd400 on CPU already achieves 23,609 seconds.**

## 5. References

[1] T. Stutzle, H-H. Hoos. MAX-MIN Ant System. Future Generation Computer System, 2000, 16(8), pp. 889-914.
[2] M. Dortgo. L-M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1997,1(l), pp. 53-66.
[3] T. Zhen, Q-W. Zhang, W-S. Zhang, et al. Hybrid ant colony Algorithm for the vehicle routing with time windows. in: Pro. International Colloquium on Computing, Communication, Control, and Management (CCCM 2008) ,2008, pp.8-12.
[4] L-H. Kuo, J-L. Ching. Ant colony optimization combined with taboo search for the job shop scheduling problem. Computers & Operations Research,2008, 35(4), pp. 1030-1046.
[5] M. Korurek, A. Nizam. A new arrhythmia clustering technique based on Ant Colony Optimization. Journal of Biomedical Informatics, 2008,41(6), pp. 874-881.
[6] M. Schluter, J-A. Egea, J-R. Banga. Extended ant colony optimization for non-convex mixed integer nonlinear programming. Computers & Operations Research,2009,36(7), pp.2217-2229.
[7] M.H. Aghdam, N. Ghasem-Aghaee, M-E. Basiri. Text feature selection using ant colony optimization. Expert Systems with Applications, 2009, 36(3), pp.6843-6853.
[8] M. Dorigo, T. Stuzle Ant Colony Optimization. MIT Press, 2007.
[9] Matthews, C. David. Improved lower limits for pheromone trails in ant colony optimization. In proc. 10th International Conference, Proceedings on Parallel Problem Solving from Nature, in LNCS, vol 5199, Springer, 2008, pp. 508-517.
[10]L. Chen, Ch-F Zhang. Adaptive Exchanging Strategies in Parallel Ant Colony Algorithm. Journal of Software, 2007, 18(3), pp. 617-624.
[11] B. Scheuermann, S. Janson, M. Middendorf. Hardware-oriented ant colony optimization.Journal of Systems Architecture, 2007, 53(7), pp. 386-402.
[12]J-D. Owens, D. Luebke, N. Govindaraju, et al. A survey of general-purpose computation on graphics hardware. Computer Graphics Forum, 2007,26(1),pp.80-113.
[13]Nvidia, NVIDIA CUDA Compute Unified Device Architecture-Programming Guide, 2008, http://developer.download.nvidia.com/compute/cuda.
[14] G. Reinelt. TSPLIB, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95, May 22, 2007.