Journal of Information & Computational Science 11:6 (2014) 1775–1784 April 10, 2014
Available at http://www.joics.com

# GACO: A GPU-based High Performance Parallel Multi-ant Colony Optimization Algorithm ⋆

Fan Li [a,b,*], Ming-lu Jin [a]

[a] *Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology Dalian 116024, China*

[b] *Network and Experimental Teaching Center, Xinjiang University of Finance and Economics Urumqi 830012, China*

**Abstract**

As a population-based algorithm, Ant Colony Optimization (ACO) is intrinsically massively parallel, and therefore it is expected to be well-suited for implementation on GPUs (Graphics Processing Units). In this paper, we present a novel ant colony optimization algorithm (called GACO), which based on Compute Unified Device Architecture (CUDA) enabled GPU. In GACO algorithm, we utilize some novel optimizations, such as hybrid pheromone matrix update, dynamic nearest neighbor path construction, and multiple ant colony distribution, which result in a higher speedup and a better quality solutions compared to other peer of algorithms. GACO is tested by the Traveling Salesman Problem (TSP) benchmark, and the experimental results show a total speedup up to 40.1× and 35.7× over implementation of Ant Colony System (ACS) and Max-min Ant System (MMAS), respectively.

*Keywords*: Ant Colony Optimization; Traveling Salesman Problem; High Performance Computation; Graphics Processing Units

## 1 Introduction

Ant Colony Optimization (ACO) [1, 2] is a constructive population-based meta-heuristic search method inspired by foraging behavior of real ants. It was first applied to Travelling Salesman Problem (TSP) [3] by Dorigo *et al.* in 1991 [4, 5]. To date, it has been utilized to solve a wide range of problems, in essence, many of which belong to graph-theoretic. The distinctive feature of ACO is a specific kind of probabilistic model, in which a structure called construction graph is coupled with a set of stochastic procedures called *artificial ants* [6]. For small and medium-scale optimization problems, ACO algorithms are known to have a significant ability of searching high-quality solutions in a reasonable time. However, for large-scale optimization problems, the

⋆This work was supported by the National Science Foundation (Grant No. 61163066) (Key Technologies Research on Information Hiding & Acquisition Based on Space-time-frequency Components)

*Corresponding author.
*Email addresses:* lifan2013666@163.com (Fan Li), mljin@dlut.edu.cn (Ming-lu Jin).

1548–7741 / Copyright © 2014 Binary Information Press
DOI: 10.12733/jics20103218

computational time of these algorithms has to seriously compromised when the current instances have a high dimension and/or are not easy to solve. That is to say, the efficiency of ACO algorithm is quite low in large real world problems. To lower computation time and improve the solution quality of ACO algorithms, some research effort about parallel computing has been done [7, 8].

Since the *artificial ants* can be independent to work, and are guided by an indirect communication through their surroundings (such as *pheromone* and other heuristic information). Therefore, ACO algorithms have a instinctive potential for parallel implementation using High Performance Computation (HPC) techniques. Over the past several years, HPC techniques have become popular and have been applied in various domains like data mining [9, 10], bioinformatics [11, 12, 13] and path optimization [14, 15], and so forth. More recently, parallel architectures such as multi-core CPU [12, 13] and many-core Graphics Processing Units (GPU) [16, 11] are becoming increasingly popular for hardware acceleration owing to their simplicity of programming and availability of cheap commodity hardware. It allows us to develop the new implementations of parallel ACO algorithms to improve the computational performance of ACO algorithms.

In this paper, we present a novel high-performance ACO parallel algorithm (named GACO), which is based on CUDA-enabled GPU. In GACO algorithm, each threads is regarded as one artificial ant, and hundreds of thousands *ant* work corporately on path construction, solution evaluation and the pheromone update. Max-min Ant System (MMAS) and Ant Colony System (ACS) share a same pheromone matrix. Also, GACO takes advantage of the strategy of dynamic nearest neighbor path construction, which significantly alleviates performance bottleneck of path search. As a result, GACO obtains a high search efficiency. Given the simulation data of TSP benchmark, GACO has a 33.7 $\times$ over its corresponding sequential implementation. Compared to latest advances of ACO parallel algorithm, GACO also achieves a speedup of speedup up to 40.1$\times$ and 35.7$\times$ over sequential implantation of ACS and MMAS, respectively.

## 2 Max-min Ant System for Traveling Salesman Problem

Researchers have presented a variety of methods to improve ACO since it was put forward. However, most of them are prone to fall into local optimum. To overcome the limitation, *Max-min* Ant System (*MMAS*) was introduced by Stäutzle and Hoos [17] to relieve the stagnation of search. MMAS is an improved algorithm over the original ant system by utilizing a strong exploitation of the search space while avoiding early search stagnation. To do this, MMAS takes advantage the following improvement: (1) Only the best ant is allowed to deposit (update) pheromone on the edges of the trail that it has constructed, and (2) Only maximum and minimum pheromone intensity levels on the edges are limited to $\tau_{max}$ and $\tau_{min}$, respectively. This helps to avoid a premature convergence of *MMAS* and reinforce exploration. $\tau_{max}$ and $\tau_{min}$ can be found in a problem dependent way.

The following will give a brief description how to utilize MMAS to solove Traveling Salesman Problem (TSP). TSP aims at finding the shortest (or "*cheapest*") round-trip (i.e., the minimum Hamiltonian cycle), which visits each of a number of "cities" exactly once. The symmetric TSP can be done through a fully connected weighted graph $G = (N, A)$, with $N$ being the set of nodes representing cities, and $A$ being the set of edges fully connecting the nodes. Since each arc $(i, j)$ is assigned a weight $e_{i,j}$, which may be distance, price, time or other factor of interest associated with edges $d_{i,j} \in A$. Using distances (associated with each arc) as cost values, the distance between cites $i$ and $j$ in the symmetric TSP, obviously, is is equivalent to the one between $j$ and

$i$, i.e., the inter-city distance $d_{i,j} = d_{j,i}$.

TSP is a well-known NP-hard optimization problem, and the method is often utilized as a standard benchmark for other heuristic algorithms [18]. The TSP was the first problem which can be solved by Ant Colony Optimization (ACO) [4]. MMAS can also be applied to solve TSP problem by the following steps:

(1) Pheromone intensity levels on all the edges are initialized to the maximum pheromone intensity level $\tau_{max}$.

(2) Ant $k$ records its own tabu table ($tabu_k$). Ant $k$ relies on the information which is offered by tabu table and a probability function to find the next *city*. By this way, ant $k$ can traversal all the *cities* exactly once, and finally come back to the starting point.

(3) Ant $k$ currently at city $i$ selects to move to city $j$ by applying the probabilistic transition function (1):

$$p_k(i,j) = \begin{cases} \dfrac{[\tau(i,j)]^{\alpha} \cdot [\eta(i,j)]^{\beta}}{\sum_{u \in J_k(i)} [\tau(i,u)]^{\alpha} \cdot [\eta(i,u)]^{\beta}} & \text{if } j \notin J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $j \notin J_k(i)$ (i.e., not in tabu table) is the set of unvisited cities by ant $k$, $\tau(i,j)$ is the pheromone intensity level on edge $(i,j)$, $\eta(i,j)$ is the visibility (also called *heuristic factor*) of $i$ from $j$ and usually is equal to $1/d(i,j)$. Herein $d(j,i)$ is the distance between cities $i$ and $j$, $\alpha$ determines the importance of pheromone intensity level, $0 \leq \alpha \leq 1$ and $\beta$ determines the importance of visibility $0 \leq \beta \leq 1$. The probabilistic function $p_k(i,j)$ is formulated to utilize the pheromone intensity $\alpha$ and the visibility $\beta$ on an edge $(i,j)$. $\alpha$ and $\beta$ are used to adjust the importance of the pheromone intensity level and the importance of visibility respectively, and they can change the behavior of ants in choosing the next city to which ants move.

(4) After completed their tours, all of the ants will update the pheromone levels on the edges by applying the following function:

$$\tau(i,j)_{new} \leftarrow (1-\sigma) \cdot \tau(i,j)_{old} + \triangle_{\tau(i,j)} \tag{2}$$

where

$$\triangle_{\tau(i,j)} = \sum_{k=1}^{m} \triangle_{\tau(i,j)} \ , \tag{3}$$

and

$$\triangle_{\tau(i,j)} = \begin{cases} \frac{Q}{L_k} & \text{if the } k-th \text{ ant uses edge } (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and $\rho$ $(0 \leq \rho \leq 1)$ is the pheromone intensity level decay parameter, $m$ is the number of ants, $Q$ is a constant, and $L_k$ is the tour length of the $k-th$ ant.

After all ants complete their trips, only the edges of the globally optimal trips, which are found from the beginning of the trial, have pheromone deposited upon them and then the pheromone from the all edges is evaporated. During the update of pheromone intensity level, the edges whose pheromone intensity level drop below $\tau_{min}$ will be set to $\tau_{min}$, and the edges whose pheromone intensity level exceed $\tau_{max}$ will be set to $\tau_{max}$.

# 3 Multi-ant Colony Optimization Algorithm Based on GPU

## 3.1 Hybrid Pheromone Matrix Update

In theory, sharing a common pheromone matrix is the simple way by which every ACO can use the matrix. In practice, however, it is not easy to obtain a good hybrid result since the best hybrid scheme must be based on the characteristics of a given ACO. In the case of *MMAS*, the pheromone $\tau_{i,j} \in P_1 = [1/(\rho\sigma C^{cb}), 1/(\rho C^{cb})]$ on a edge $(i, j)$, where $\sigma$ is a constant or only relates to the problem scale, and $C^{cb}$ is the length of the current optimal path. The lower and upper bound of $\tau_{m,n}$ is dynamically updated during of the execution of the *MMAS*. Different from *MMAS*, the pheromone in ACS is initialized as $\tau_0 = 1/(NL_{opt})$, where $L_{opt}$ is noted as the near optimal solution which is obtained by using nearest neighbors method, and $N$ is the number of nodes in TSP. The upper bound in theory $\tau_{i,j} = 1/L_{bs}$ can be computed according to the update rules of path, where $L_{bs}$ is the optimal length of path. That is, $\tau_{i,j} \in P_2 = [\tau_0, 1/L_{bs}]$.

Although both *MMAS* and ACS depend on the upper and lower bounds of pheromone, the *MMAS* utilizes the maximum value as initial value. As a result, the pheromone $\tau_{i,j}$ trend downward as a whole, only the edges on the optimal path are strengthen, which aims at achieving the differentiation of pheromone. On the contrary, $\tau_{i,j}$ in ACS is initialized with the minimum value. the pheromone of the edges which are not often visited may hover near of the minimum value. Therefore, the pheromone of the edges on the optimal path is keeping rise. The different strategies of *MMAS* and ACS lead to the difficulty how to hybrid use them properly.

We utilize the intersection between *MMAS* and ACS, and then present a uniform formula for global and local pheromone update. The details are follows. Since $L_{bs} \geq L_{opt}$, we can obtain the upper bound of $P_2 = 1/L_{opt}$. By contrast, the upper bound of $P_1 = 1/(\rho L_{opt})$, where $\rho$ is a small enough positive real number. Herein we have $\tau_{min} > \tau_0$ and $\tau_{max} > 1/L_{bs}$. Therefore, we use a new strategy in which $\tau_{i,j} \in P_1 \cap P_2 = [\tau_{min}, 1/L_{bs}]$. Herein the pheromone of all the edge are initialized to $\tau_{max}$, i.e., $\tau_0 = \tau_{max} = 1/\rho L_{bs}$, where $\rho$ is from the setting of *MMAS*.

Each of the evaporation factor $\rho$ in *MMAS* and ACS is only dependent upon its own setting of sequential implementation. In such way, a effective hybrid strategy of pheromone update can be obtained considering to the benefits of both MMAS and ACS.

## 3.2 Dynamic Nearest Neighbor Path Construction

The process of path searching in ACO easily becomes a performance bottleneck, especially when the scale of problem is large, the number of ants will increase correspondingly (usually the ant number $m$ is equal to the problem size $n$). We mainly have this step parallelized in GPU. To improve search efficiency, $m$ ants are expected to search the solutions in parallel in the CUDA cores.

As for the problem solving of TSP in complex and lager city map, the traversal from city $i$ connected to city $j$ through all of the shortest path of city, it is very low possible that city $j$ is the furthest one away from the city $i$. At the beginning of the iteration algorithm, however, due to the lack of adequate guidance, ants may have to wrongly choose the other city $j$ instead of the the nearest neighbor of $i$ city. Besides, if the ants release pheromone, then the information will

misguide the partial path construction of the subsequent ants. Thus, we propose a hybrid strategy for selecting city $j$. In this strategy, both the information of static nearest neighbor of city (noted as $DNN_i$) and the dynamic nearest neighbor $DL_j$ of the last iteration path information are used to determine the potential candidate city $Cand_j$, which be given by Eq. (5):

$$Cand_i^{t+1} = \{i | i \in NN_i \text{ and } DL_j \neq 1\} \tag{5}$$

where $DL_j$ can be defined by :

$$PC_j = \begin{cases} 1 & \text{if } \exists d_{jp}^{t,k} = d_{T^k[h]T^k[h+1]} \text{ and } d_{jp}^{t,k} \notin NN_j \ (j = T^k[h], g = T^k[h+1]) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where the solution trails of ant $k$ is the sequence $T^k[1]$, $T^k[2]$, $\cdots$, $T^k[n]$, $T^k[n+1] = T^k[1]$, and the ant will deposit the pheromone on the edge after the $t - th$ iteration of search. The construction of dynamic nearest neighbor path now is subject to some conditions. It helps the path construction be able to avoid from the weak edge candidates, which are obtained from last iteration at a relative high probability. By this method, it is expected to effectively accelerate the convergence of the algorithm, and thus enhance the diversity of solutions.

In this strategy, each ant is responsible for one work-item, which can construct a complete solution, and manage all data(such as probabilities calculations, list of visited cities, and so forth) needed for this phase. Algorithm 1 outlines the computional kernel which implements the ant system decision rule. Herein the 1-D domain range is set as $global_{size} \leftarrow num_{ants}$

The paths of all ants are stored in the matrix of candidate solutions (noted as $solution[\cdot][\cdot]$), with each row representing one whole solution of ant. The set of visited nodes (noted as $visited[\cdot]$) keeps track of the current visited nodes for each ant. The data structure of $visited[\cdot]$ will help to prevent duplicate selection which is forbidden by the TSP rule. In another word, all elements are initialized as *false*, and the $i - th$ element is set to *true* only when the $i - th$ node is chosen to be part of the solution of ant. At the current node $c$, the probability of each node $i$ being selected is stored in $select_{prob}[i]$ , whose value is based on the pheromone trails and heuristic information according to Equations (1) and (2). For highly memory access, all such kind data are cached in $choice_{info}[c][i]$.

In addition, to obtain a higher performance improvement from GPU, we conduct a series of optimization. For example, The function $Random(a, b)$ returns a uniform real-valued pseudo-number between $a$ and $b$ in Algorithm 1. The random number generator can be implemented on the GPU, named CUDA-RNG, which was proposed in our previous work [19]. In CUDA-RNG, CPU generates enough random seeds, then sends the seeds to GPU, and the rest random numbers are generated in GPU. In such way, we improve computation density in GPU, resulting in a low access latency of global memory. Furthermore, the build-in function *__powf* () in CUDA is also used to enhance the performance.

## 3.3 Multiple Ant Colony Distribution

In this paper, according to the characteristics of CUDA, we design a double parallel strategy in which multiple ant colony run in parallel and each ant in the single ant colony run in parallel at the same time. the optimum value for the final solution is based on multiple ant colony optimization.

**Algorithm 1:** Computational kernel for dynamic nearest neighbor path construction

**Input**: $PM$ - Pheromone matrix ; $DM$ -Distance matrix; $m$ - Number of ants; $n$ - Number of cites in TSP;

**Output**: $path$- Path of construction

```
/* Initialization */
```

$visited[\cdot] \leftarrow false$;

```
/* Selection of the initial cities */
```

$init_{city} \leftarrow$ **Random**$(0, n-1)$ ;

solution$[global_{id} \times n + 0] \leftarrow Init_{node}$;

visited$[global_{id} \times n + Init_{node}] \leftarrow true$;

**for** $s \leftarrow 1$ **to** $n-1$ **do**

  $sum_{prob} \leftarrow 0.0$ ;

  $current_{city} \leftarrow$ solution$[global_{id} \times n + (s-1)]$;

  `/* Calculate of the probabilities of cities */`

  **for** $k \leftarrow 0$ **to** $n-1$ **do**

    **if** $visited[global_{id} \times n + k]$ **then**

      selection$_{prob}[global_{id} \times n + k] \leftarrow 0.0$;

    **else**

      selection$_{prob}[global_{id} \times n + k] \leftarrow$ choice$_{info}[current_{city} \times n + i]$;

      $sum_{prob} \leftarrow sum_{prob} + select_{prob}[global_{id} \times n + k]$;

    **end**

  **end**

  `/* City selection */`

  $r \leftarrow$ **Random**$(0, sum_{prob})$;

  $k \leftarrow 0$;

  p $\leftarrow$ select$_{prob}[global_{id} \times num_{cities} + 0]$;

  **while** $p < r$ **do**

    $k \leftarrow k + 1$;

    $p \leftarrow p +$ select$_{prob}[global_{id} num_{cities} + i]$;

  **end**

  solution$[global_{id} \times num_{cities} + s] \leftarrow k$;

  visited$[global_{id} \times num_{cities} + k] \leftarrow true$;

**end**

The multiple ant colony distribution strategy for multiple ant colony using CUDA is shown in Fig. 1.

Let $b$ be the number of ant colony, each of which has more than one thread blocks (noted as $Block_i$, $i = 0, 1, 2, \cdots, b-1$), $PM$ be pheromone matrix and $DM$ be distance matrix of cities. All the ant colony share one pheromone matrix $P$. According to the characteristics of memory architecture of GPU, pheromone matrix $PM$ is stored in global memory since global memory has the biggest capacity. Since the data transfer back and forth between CPU and GPU takes long time, the search for solution is suggested to do as much as possible on the GPU.

As the distance matrix $DM$ remains unchanged during the search of optimal solution, it can be regarded as read-only data, which can be bound to texture memory. For read-only patterns, texture memory fetches is a better alternative to global memory reads because of texture memory cache, which can further improve the whole performance of GACO. As a result, we store the

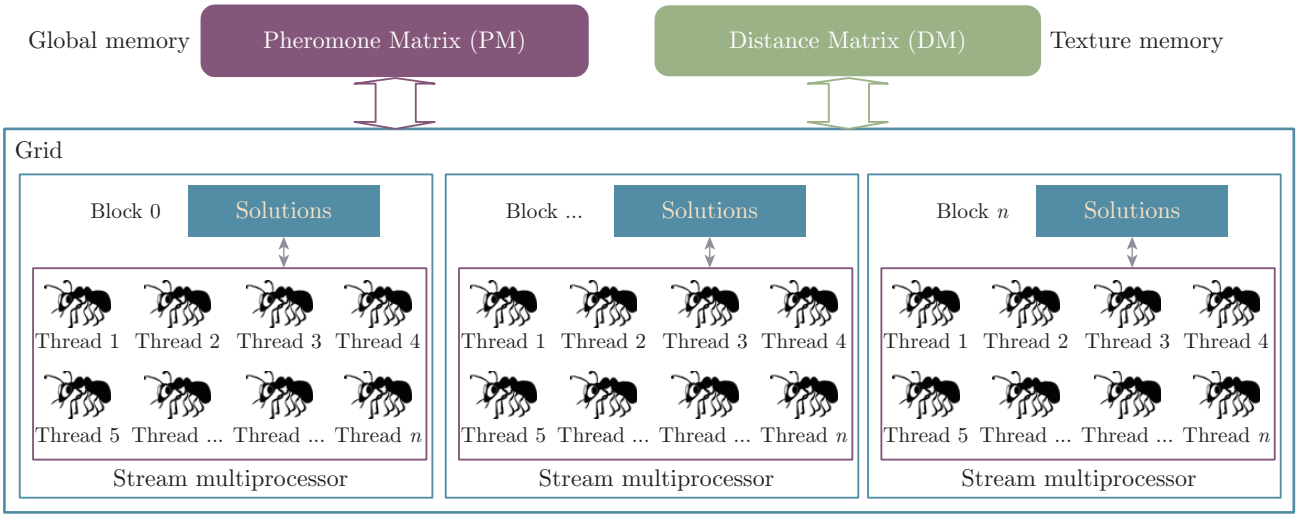


Fig. 1: The strategy of multiple ant colonies on GPU

distance matrix $DM$ into texture memory. Since global memory is much slower than shared memory, a profitable way of performing computation on the device (i.e., GPU) is to tile data to take advantage of shared memory. Thus, we store the current solutions into the shared memory, whose higher access speed helps to accelerate the solution reduction.

# 4 Experimental Results and Discussion

To demonstrate the performance of our proposed approach, we carry on some simulations on a 8-core CPU @2.1 GHz and 16 GB memory, a GPU NVIDIA Tesla and C1060 and C2050. The implementation is running a 64-bit Linux-based operating system. Our program has been compiled using gcc 4.5 and CUDA 5.

GACO is a hybrid algorithm of *MMAS* and ACS, and pheromone influence parameter $\beta$ =[2, 5] according to the features of the ACO parameter sensitivity. To focus on the parallelization the algorithm, the parameters of GACO like the number of ants $M, \alpha, \rho$, etc., are set based on the values recommended in [1]. For the purpose of this study, the key important parameters are $\alpha = 1$, $\rho = 0.5$ and the number of ants $M$.

A standard set of benchmark instances from the well-known TSPLIB library [20, 21] is used to test our proposed algorithm. All problems instances are defined on a complete graph and all distances are defined to be integer numbers. Since GACO is a heuristic algorithm, which may result in a different experimental results in each execution. Especially for TSP problems, the TSPLIB provides the best known solutions.

## 4.1 Performance Evaluation

The mean and/or standard deviation of solutions of all ACO algorithms have become the preferred indicator to evaluate their performance. In this paper, we compare GACO with sequential implementation MMAS, ACS, the parallel ant colony method based on the ring structure, A-COMAC, and its variants, ACOMAC +DNN [22]. The experimental results in terms of solution

quality are listed in Table 1, the number of runs $R = 50$, the number of iterations $I = 3000$.

Table 1 shows the mean of solutions of these compared algorithm. As for the solutions of the first four problems, all the experimental results are obtained from the real execution. Other experimental results are got directly from [22]. The results of the later two instances are also obtained from the actual execution, "—" means that there are no same test in [22]. From the Table 1, GACO achieves the best solution quality (i.e., the minimum mean of solutions).

Table 1: Comparisons of our GACO Solutions with MMAS, ACS and ACOMAC using 6 different TSP benchmark

| TSP Name | MMAS | ACS | ACOMAC | ACOMAC+DNN | GACO |
|---|---|---|---|---|---|
| eil61 | 432.9 | 435.6 | 430.05 | 430.670 | 416.52 |
| eil76 | 559.12 | 556.8 | 554.5 | 549.6 | 536.7 |
| kroa100 | 22, 570.6 | 22, 679.6 | 21, 475.8 | 21, 402.7 | 21, 226.9 |
| d198 | 16, 476.5 | 16, 725.6 | 16, 055.5 | 15, 755.6 | 15, 076.2 |
| lin318 | 43, 651.7 | 44, 158.9 | — | — | 41, 325.4 |
| rat575 | 7, 021.8 | 7, 169.6 | — | — | 6, 532.2 |

## 4.2 Overall Performance

Fig. 2 shows the speed-up over sequential implementation. Since tour construction needs to generates random number generation, and therefore the computation workload for the application can not easily balance when using the small size benchmarks. As for the TSP benchmark 'eil61', we only obtain speedup 8.9× and 4.7× over its responding sequential implementation using C2050 and C1060, respectively. The number of ants, which is equivalent to the number of threads running in parallel on the GPU, is relatively small for these instances. That is, we cannot harvest the GPU's computation power when problem size is small. As for the TSP benchmark 'lin318', whose problem size is much bigger than 'eil61', we get the best speedup 33.7× and ×26.3, respectively, over its responding sequential implementation. The TSP problem size of 'rat575'is larger than 'lin318', but we note that its speedups begin to deteriorate. The reason is that the GPU occupancy trend to be lower. A high GPU occupancy, which can hide global memory latency, is very important to improve the whole performance of GACO. This can be done by creating enough threads to keep the CUDA cores always occupied while many other threads are waiting for global memory accesses. As for some parts in GACO which have branches cannot get good performance, we try to utilize multi-core CPU to share the parts of job (noted as 'C2050+Multi-CPU' in Fig. 2). As a result, the best speedup of 35.6× can be reached when benchmark 'lin318' is tested. Since Tesla C1060 has less shared memory and CUDA cores than those of C2050, the speedup of Tesla C1060 is lower than C2050.

To accelerate the choice of the iteration-best solution, we utilize a parallel reduction technique that hangs up the execution of certain work-items. This technique needs to use barrier synchronization for ensure consistency of memory. In addtion, we use the optimizations such as multiple ant colony distribution, dynamic nearest neighbor path construction and hybrid pheromone matrix update, which further improve the performance of our proposed algorithm.

Fig. 3 shows the effect of implementing our proposed GACO, compared to the implementation of ACS, *MMAS*, ACOMAC and ACOMAC+DNN [22]. All the experiments are carried out in

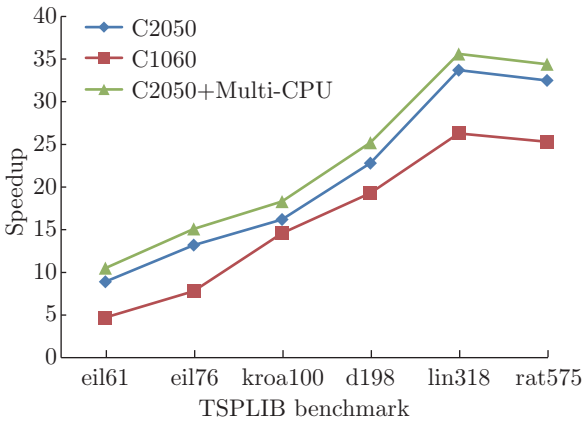


Fig. 2: Speedup over sequential implementation using C1060 and C2050, respectively

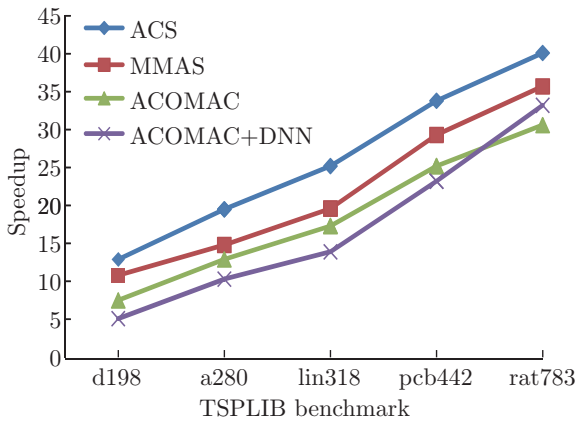


Fig. 3: Speedup over ACS, MMAS, ACOMAC and ACOMAC+DNN, respectively

GPU C2050. As the size of problem increase, we observe speedup 40.1×, 35.7×, 30.6×, and 33.2× factor over ACS, *MMAS*, ACOMAC and ACOMAC+DNN, respectively. Compared to *MMAS*, as shown in Fig. 3, we obtain a higher speedup over ACS, which means that ACS takes longer execution time than MMAS. This is because that ACS spends more time than MMAS on local pheromone update. ACOMA and ACOMAC+DNN employ more heuristic information, which results in a better performance compared to ACS and *MMAS*.

# 5 Conclusions

This paper has presented and discussed a novel high-performance ACO parallel algorithm (named GACO) on CUDA-enabled GPU. GACO utilizes some novel optimizations, such as hybrid pheromone matrix update, dynamic nearest neighbor path construction, and multiple ant colony distribution, resulting in a higher speedup compared other peers of GPU algorithm and a better quality solutions. For complex or large-scale problems in which abundant ants are needed, the ant-based parallel strategies similar to GACO are expected to fully explore the computational power of the GPUs.

# References

[1] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, Computational Intelligence Magazine, IEEE, Vol. 1, No. 4, 2006, 28-39

[2] A. Colorni, M. Dorigo, V. Maniezzo, P. di Milano, Distributed optimization by ant colonies, in Proc. of the First European Conference on Artificial Life, Paris, France, Vol. 142, 1991, 134-142

[3] Z. Michalewicz, D. B. Fogel, The Traveling Salesman Problem, Springer, 2004

[4] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol. 26, No. 1, 1996, 29-41

[5] M. Dorigo, V. Maniezzo, A. Colorni, V. Maniezzo, Positive feedback as a search strategy, Technical Report 91-016, 1991

[6] M. Zlochin, M. Birattari, N. Meuleau, M. Dorigo, Model-based search for combinatorial optimization: A critical survey, Annals of Operations Research, Vol. 131, No. 1-4, 2004, 373-395

[7] J. M. Cecilia, J. M. Garcia, A. Nisbet, M. Amos, M. Ujaldón, Enhancing data parallelism for ant colony optimization on GPUs, Journal of Parallel and Distributed Computing, Vol. 73, No. 1, 2013, 42-51

[8] K. Kobashi, A. Fujii, T. Tanaka, K. Miyoshi, Acceleration of ant colony optimization for the traveling salesman problem on a GPU, in Proc. of the IASTED International Conference Parallel and Distribited Computing and Systems, 2011, 108-115

[9] R. Wu, B. Zhang, M. Hsu, Clustering billions of data points using GPUs, in Proc. of the Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop, 2009, 1-6

[10] Y. Zhang, F. Chen, High performance sequence mining using pairwise statistical significance, in HPC Transitions towards Exascale Processing, IOS Press, 2013, 1-21

[11] Y. Zhang, S. Misra, A. Agrawal, M. M. A. Patwary, W. K. Liao, Z. Qin, A. Choudhary, Accelerating pairwise statistical significance estimation for local alignment by harvesting GPU's power, BMC Bioinformatics, Vol. 13, No. Suppl 3, S1, 2012

[12] Y. Zhang, F. Zhou, J. Gou, H. Xiao, Z. Qin, A. Agrawal, Accelerating pairwise statistical significance estimation using NUMA machine, Journal of Computational Information Systems, Vol. 8, No. 9, 2012, 3887-3894

[13] Y. Zhang, M. M. A. Patwary, S. Misra, A. Agrawal, W. K. Liao, Z. Qin, A. Choudhary, Par-PSSE: Software for pairwise statistical significance estimation in parallel for local sequence alignment, International Journal of Digital Content Technology and Its Applications, Vol. 6, No. 5, 2012, 200-208

[14] T. N. Bui, T. Nguyen, J. R. Rizzo Jr., Parallel shared memory strategies for ant-based optimization algorithms, in Proc. of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM, 2009, 1-8

[15] S. Tsutsui, ACO on multiple GPUs with CUDA for faster solution of QAPs, in Parallel Problem Solving from Nature-PPSN XII, Springer, 2012, 174-184

[16] Y. Zhang, S. Misra, D. Honbo, A. Agrawal, W. K. Liao, A. Choudhary, Efficient pairwise statistical significance estimation for local sequence alignment using GPU, in Proc. of IEEE 1st International Conference on Computational Advances in Bio and Medical Sciences, 2011, 226-231

[17] T. Stutzle, H. H. Hoos, MAX-MIN ant system, Future Generations Computer Systems, Vol. 16, No. 8, 2000, 889-914

[18] D. S. Johnson, L. A. McGeoch, The traveling salesman problem: A case study in local optimization, Local Search in Combinatorial Optimization, 1997, 215-310

[19] F. Li, M. L. Jin, A high performance random number generator using heterogeneous computing platform, Journal of Computational Information Systems, Vol. 9, No. 22, 2013, 9003-9011

[20] G. Reinelt, TSPLIB: A traveling salesman problem library, ORSA Journal on Computing, Vol. 3, No. 4, 376-384, 1991

[21] TSPLIB, Traveling salesman problem library, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, 2013

[22] C. F. Tsai, C. W. Tsai, C. C. Tseng, A new hybrid heuristic approach for solving large traveling salesman problem, Information Sciences, Vol. 166, No. 1, 2004, 67-81