International Journal of Grid Distribution Computing
Vol. 8, No.4, (2015), pp. 37-46
http://dx.doi.org/10.14257/ijgdc.2015.8.4.04

A GPU-based Parallel Ant Colony Algorithm for Scientific Workflow Scheduling

Pengfei Wang, Huifang Li and Baihai Zhang

School of Automation, Beijing Institute of Technology , 5 South Zhongguancun Street, Haidian District, Beijing, 100081, China
E-mail: huifang@bit.edu.cn, wpf999@yeah.net

Abstract

Scientific workflow scheduling problem is a combinatorial optimization problem. In the real application, the scientific workflow generally has thousands of task nodes. Scheduling large-scale workflow has huge computational overhead. In this paper, a parallel algorithm for scientific workflow scheduling is proposed so that the computing speed can be improved greatly. Our method used ant colony optimization approaches on the GPU. Thousands of GPU threads can parallel construct solutions. The parallel ant colony algorithm for workflow scheduling was implemented with CUDA C language. Scheduling problem instances with different scales were tested both in our parallel algorithm and CPU sequential algorithm. The experimental results on NVIDIA Tesla M2070 GPU show that our implementation for 1000 task nodes runs in 5 seconds, while a conventional sequential algorithm implementation runs in 104 seconds on Intel Xeon X5650 CPU. Thus, our GPU-based parallel algorithm implementation attains a speed-up factor of 20.7.

Keywords: *workflow scheduling, ant colony optimization, parallel computing, GPU computing, CUDA*

1. Introduction

Graphic Processing Unit (GPU) was proposed by NVIDIA Inc. in 1999. GPU contains several independent computing units and many processing cores that can make 2D and 3D rendering. GPU, essentially a parallel processor, due to its great float computing ability, is designed to do graphics rendering and general-purpose computing in its architecture. GPU has become a new computing paradigm and been largely used in high performance computing (HPC). GPU has been widely used in areas such as astronomy, molecular dynamics, hydromechanics, image processing, signal processing, pattern recognition, data processing in oil prospecting and cryptography. Generally, program based on GPU acceleration is faster than that of CPU based one with several times.

Ant colony optimization (ACO) algorithm [1-2] was initially proposed by Marco Dorigo in 1992, and it works well when solving combination optimization problems [3]. Parallel ant colony algorithm is a kind of ant colony algorithm designed to run over parallel hardware. Due to the intensive computation of large scale optimization problems, parallel hardware is needed to finish computation timely. Currently, parallel ant colony optimization algorithm includes: cluster and MPI based parallel algorithm [4], multi-core CPU and OpenMP based parallel algorithm [5], GPU and CUDA based parallel algorithm [6-9].

Scientific workflow scheduling problem is a combinatorial optimization problem. Benedict et al. solved the scheduling of scientific workflows using Niched Pareto Genetic Algorithm (NPGA) [10]. Jia Yu et al. presented a genetic algorithm approach to address scheduling optimization problems in workflow applications, based on two QoS constraints, deadline and budget [11-12]. Wieczorek et al. solved bi-criteria scheduling of

ISSN: 2005-4262 IJGDC
Copyright ⓒ 2015 SERSC

scientific workflows for the grid [13]. Plankensteiner et al. proposed a dynamic execution and scheduling heuristic capable of scheduling workflow applications with a high degree of fault tolerance [14]. These methods and algorithms are sequential computing methods. When the scale of workflow becomes larger, scheduling computational overhead becomes larger. So it is necessary to introduce parallel optimization algorithms to solve large-scale scientific workflow problems.

In this paper, we propose a workflow scheduling algorithm based on GPU accelerated parallel ant colony optimization algorithm. Due to the speedup of GPU based computation is generally several times faster than that of CPU, we choose GPU as our computing device. Further, we choose ant colony optimization algorithm and its parallel form to solve scientific workflow problems. We have implemented our parallel algorithm using CUDA [15-16] and tested our code in NVIDIA Tesla M2070. The GPU implementation attains a speed-up factor of 5.8~20.7 over the conventional CPU implementation for problems with different scale.

This paper is organized as follows: in section II, we describe the scientific workflow scheduling problem and its mathematic model; in section III, we present GPU-based parallel ant colony optimization algorithm and its implementation details; in section IV, we present experimental analysis of the proposed algorithm; finally, we conclude our work in section V.

2. Problem Description

All printed scientific workflow model is described with no loop directed graph $DAG = < V, E >$, where node in the graph denotes task nodes of the workflow, edge denotes relationship between tasks. The i-th task is denoted as T_i ($1 \le i \le N$). If task T_j depends on task T_i, then there is an edge between T_i and T_j, denoted as $< i, j > \in E$. T_j can only be executed after T_i finishes execution. T_i's predecessor tasks are denoted as $pred(T_i)$ and its successive tasks are denoted as $succ(T_i)$. Suppose the workflow begins from task T_1, thus $pred(T_1) = \varnothing$ (because there is no task before T_1). When the workflow stops at task T_N, $succ(T_N) = \varnothing$. For a task T_i, suppose there exist M_i candidate computing resource service to be allocated to run task T_i. Denote the candidate computing resource service as S_{ik} ($1 \le k \le M_i$), and suppose one task can only choose one computing resource service. Let the makespan of task T_i executed over S_{ik} be t_{ik} and the cost of task T_i executed over S_{ik} be c_{ik}. Suppose decision variables of the workflow scheduling problem are x_{ik}, if task T_i is allocated with computing resource S_{ik}, then $x_{ik} = 1$, otherwise $x_{ik} = 0$.

The overall makespan of the workflow is determined by its each task completion time and the topology of the DAG. Let t_i be the time to complete task T_i, so t_N is the makespan of the whole workflow. The scheduling problem is to map every T_i onto some S_{ik} to achieve a minimum execution cost and also complete the workflow execution within the deadline D. The mathematical description of scientific workflow scheduling problem is as follows:

 Copyright ⓒ 2015 SERSC

$$\min \sum_{i=1}^{N} \sum_{k=1}^{M_i} c_{ik} x_{ik} \tag{1}$$

$$\text{s.t.} \quad t_i \le t_j - \sum_{k=1}^{Mj} t_{jk} x_{jk} \quad \langle i, \ j \rangle \in E \tag{2}$$

$$t_i \ge 0 \qquad i=1, 2,\ldots, N \tag{3}$$

$$t_N \le D \tag{4}$$

$$\sum_{k=1}^{M_i} x_{ik} = 1 \qquad i=1, 2,\ldots, N \tag{5}$$

$$x_{ik} \in \{0,1\} \qquad i=1, 2,\ldots, N; \ j=1,2,\ldots, M_i \tag{6}$$

Formula (1) is the objective function of the problem, i.e., the total cost of scientific workflow; the total cost is the sum of each task's cost.

Formula (2) represents the timing constraint relationship between tasks, task T_j can only begin after the completion of T_i, T_i belongs to the predecessor task set of T_j, i.e., $T_i \in pred(T_j)$. The start time of T_j is $t_j - \sum_{k=1}^{Mj} t_{jk} x_{jk}$.

Formula (3) represents that the completion time of each task node is non-negative, and formula (4) represents that the total time of the workflow (makespan) meets the deadline constraints.

Formula (5) represents that each task node can only select one candidate computing resource.

Formula (6) describes that the problem decision variables x_{ik} are boolean variables.

3. The Design and GPU Implementation of Parallel Ant Colony Algorithm

In this section, we will introduce ant colony optimization algorithm for scientific workflow scheduling problem and its GPU-based parallel implementation in detail.

3.1. Algorithm for Scientific Workflow Scheduling

The algorithm consists of three parts: data initialization, solution construction and pheromone update. It is a continuously iterative process which includes building the solution and updating the pheromone.

Data initialization includes: pheromone matrix initialization, heuristic information matrix initialization, random number generator initialization, and other related auxiliary variable initialization. Each element of the pheromone matrix is initialized to τ_0, regarded as the algorithm parameter, and its value is specified by user. Each element of the Heuristic information matrix is initialized to $1/(c_{ik} t_{ik})$. It is used to guide the ants to choose computing resources with the low cost and short execution time in the solution construction. The random number generator generally initializes the random number seed, because the ants are ultimately driven by the random number when building the solution. If the random number seed is not set,

the runtime system will use the default seed, this will cause the same results of algorithm in each run, thus there is no statistical significance in the multiple runs.

The construction of the solution is the core part of the algorithm, in each round iteration, there are several ants to build the solution of the problem parallel. Here, the solution of the problem is an allocation scheme of a workflow task and computing resources, expressed as the vector X = (x_1 , x_2 , ..., x_N), $x_i = k$ means that computing resources S_{ik} are allocated for the task T_i . Building solution is the process that ants select the computing resources S_{ik} for the task T_i . For a workflow network with N tasks, it needs N times selection in total. In each selection, the ant selection are uncertain, but according to the specific distribution law or heuristics based on selection probability, this is the node selection rules of ants, node selection rules are described in detail in section 3.2.

An important aspect of the algorithm is the pheromone updating, including local pheromone updating and global pheromone updating. Local ant pheromone updating is in the process of constructing the solution, and the global pheromone updating is iteratively performed in each round. Pheromone update aims to realize the ant positive feedback in solution searching space, i.e. after an excellent solution to be found, the subsequent possibility that ants choose this solution will increase. Pheromone update rule is described in detail in section 3.3.

The input of the algorithm includes: the directed graph matrix of a scientific workflow, computing resource cost matrix, computing resource time-consuming matrix and workflow deadline. The output of the algorithm includes: the best allocation scheme found by the algorithm and its objective function value. Algorithm pseudo-code is as follows:

Pseudo code of the algorithm:

```
Algorithm ACO_for_ workflow_scheduling
input DAG[N][N]  // matrix of workflow DAG
input C[N][M]    // matrix of cost
input T[N][M]    // matrix of time
input D          // deadline of workflow
begin
Initialize pheromone matrix
Initialize heuristic information matrix
Initialize random generator
best_value := +∞
best_solution:= ∅

for ( i:=1;  i<=nc;  i++)
for ( r:=1; r <= R; r++)
      solution[r] = ant_create_solution( )
      c := cost of solution[r]
      t := total time of solution[r]
      if ( t > D )
          c:=c+1000000
      end-if
      if ( c < best_value )
          best_value:=c
          best_solution:= solution[r]
      end-if
    end-for
    Update pheromone
```

 Copyright ⓒ 2015 SERSC

end-for
output best_value , best_solution
end.

Pseudo code of ant colony generated solution:
procedure ant_create_solution( )
begin
solution:= ∅
for (i:=1; i<=N; i++)
Choose node *k* according to the node selection rule
solution:= solution ∪ {<i,k>}
Update local pheromone
end-for
return solution
end.

The time complexity of the algorithm is $O(nc \times R \times N \times M)$, where nc is the iteration times, R is the number of ants, N is the number of tasks and $M = max\{M_i\}$.

3.2. Node Selection Rules

In this section, we will introduce the node selection rules.

Suppose q is a random number with uniform distribution over [0, 1], q_0 ($0 \le q_0 \le 1$) is a parameter. When $q \le q_0$, ant selects ordered pair $< i, k >$ according to empirical knowledge:

$$k = \arg\{ \max_{u \in allowed(t)} [\tau_{iu}(t)\eta_{iu}^{\beta}]\} \tag{7}$$

When $q > q0$, ant select ordered pair $< i, k >$ according to the following probability equation:

$$P(\mathrm{i,k,t}) = \begin{cases} \dfrac{\tau_{ik}(t)\eta_{ik}^{\beta}}{\sum_{u \in allowed(t)} \tau_{ik}(t)\eta_{ik}^{\beta}} & \mathrm{k} \in allowed(t) \\ 0 & otherwise \end{cases} \tag{8}$$

where *i* is the current task number, *k* is the computing resource number allocated to task T_i, $\tau_{ik}(\mathrm{t})$ is the pheromone corresponding to ordered pair $< i, k >$, $\tau_{ik}(\mathrm{t}) = \tau_0$ (τ_0 is the initial value of the pheromone), η_{ik} is the heuristic information of $< i, k >$. In our algorithm, $\eta_{ik} = 1/(c_{ik}t_{ik})$, β is the weight of heuristic information, and allowed(*t*) is the set of available computing resources.

3.3. Pheromone Update Rule

Pheromone update rules include two kinds–the local and global. For local pheromone update rules, when the ant has selected an ordered pair $< i, k >$, it updates the pheromone as follows:

$$\tau_{ik}(t+1) = (1-\rho)\tau_{ik}(t) + \rho\tau_0 \tag{9}$$

Copyright ⓒ 2015 SERSC

Where ρ ($0 < \rho < 1$) is the evaporation factor of the local update rule.

Global pheromone update is done after one iteration as:

$$\tau_{ik}(t+m) = (1-\gamma)\tau_{ik}(t) + \gamma\Delta\tau \tag{10}$$

$$\Delta\tau = \begin{cases} 1/best\_value & <i,k> \in best\_solution \\ 0 & otherwise \end{cases} \tag{11}$$

Where γ ($0 < \gamma < 1$) is the evaporation factor of the global update rule, *best_solution* is the global optimal solution until the current iteration, and *best_value* is the corresponding objective function value.

3.4. GPU-based Algorithm Implementation

The algorithm is implemented in three parts: initialization, constructing problem solution and pheromone matrix update. The initialization part is executed by CPU and the other two parts are executed by GPU. The construction of problem solution consists of four kernel functions: *ant_create_solution*, *evalue_total_time*, *evalue_cost* and *select_node*. Pheromone matrix update has two kernel functions: *loacl_update* and *global_update*. Here we introduce 6 kernel functions as follows.

Kernel function *ant_create_solution* simulates an ant to get solution of the problem, which is stated with keyword **__global__** and called from the host code. When solving the problem, *ant_create_solution* function triggers several threads to run over GPU to search the solution space in parallel. Kernel function *ant_create_solution* is a loop with N steps (N is the task number in the workflow). In the loop, one of the candidate services of the *i*-th task is selected and added into the solution space. When the solution space is constructed, the execution time of the workflow is evaluated. If the time requirement is satisfied, the solution vector is returned; otherwise the solution vector is abandoned. *ant_create_solution* function will call *select_nod*e and *evalue_total_time* functions.

Kernel function *select_node*, *evalue_total_time*, *evalue_cost*, *local_update* and *global_update* are stated with keywords **__device__** and called by the kernel function *ant_create_solution*.

Kernel function *select_node* selects a computing resource service from the candidate resources, and this kernel is the implementation of the node selection rule with roulette wheel method. This method is essentially to generate discrete random variables satisfying equation (8).

Kernel function *evalue_total_time* computes the overall makespan of the workflow. This function searches DAG of the workflow based on breadth-first search method, and computes t_N of the terminal node.

Kernel function *evalue_cost* computes the total cost of the workflow. When the makespan of the workflow is greater than deadline D, it means that the solution is unfeasible, so the function will add a very big number into the cost.

Kernel function *local_update* updates local pheromone according formula (9), and ant generating a solution needs to call this function and the pheromone of the corresponding solution is updated accordingly.

Kernel function *global_update* updates the global pheromone according to formula (10) and (11). This function is executed only once during one iteration and updates pheromone with the optimal solution.

Besides the above functions, there are some auxiliary functions in the algorithm, and we omit their description here.

 Copyright ⓒ 2015 SERSC

4. Performance Evaluation

4.1. Experimental Settings

To verify the performance and speedup of the proposed GPU-based parallel ant colony optimization algorithm, we use CUDA C to implement the algorithm and compare it with a CPU-based sequential algorithm.

The hardware used in the experiment is: NVIDIA Tesla M2070 GPU with 14 SM units, each SM unit with 32 CUDA cores (so there are 448 cores totally), 6GB device memory. Theoretically, the floating performance with single precision of the GPU is 1.8TFlops. We used Intel Xeon X5650 CPU running in 2.66GHz to run the CPU-based sequential ant colony optimization algorithm for scientific workflow scheduling.

Software used in our experiment is: Red Hat Enterprise Linux 5.4 X86_64, CUDA ToolKit 4.2 and GNU g++ compiler.

The parameters of the algorithm are listed in Table 1.

Table 1. Algorithm's Parameters

Parameter	Value
β	1
ρ	0.2
γ	0.2
$\tau 0$	0.01
q0	0.1

To test the performance of the algorithm, we experimented over 3 problems with different scales. For the three instances, the number of workflow task nodes is N=12, N=100 and N=1000 respectively, and randomly generated workflow networks are utilized in the instances with N=100 and N=1000. There are 10 candidate services for each workflow task node. Workflow deadlines of the three problems are D=30, D=150 and D=400 respectively. The problem instances are listed in Table 2.

Table 2. Problem Instances

No.	N	M	D
#1	12	10	30
#2	100	10	150
#3	1000	10	400

4.2. Experimental Results

In our performance evaluation process, parallel and sequential algorithms are all executed 100 times, and the results are the average value of the 100 times execution of the two algorithms.

Copyright ⓒ 2015 SERSC

The performance comparison of our parallel algorithm with the sequential one is listed in Table 3. The objective function value is the smaller the better, and the makespan of workflow should meet the deadline constraints.

Table 3. Algorithm Performance Comparison

Problem instances	Sequential algorithm		Parallel algorithm	
	Objective	Makespan	Objective	Makespan
#1	53.0	30.0	56.2	29.5
#2	408.2	143.6	283.6	148.6
#3	4759.8	388.6	4082.8	394.4

When there are 1000 tasks in scientific workflow, i.e., N=1000, the best cost of workflow found by sequential algorithm and parallel algorithm are 4880.14 and 4214.28 respectively. When N=100, their best cost are 421.69 and 371.41 respectively. But in small scale problem instances, i.e., N=12, the cost found by parallel algorithm is bigger than that of the sequential one. We will improve our algorithm performance for small scale problems in future work.

Time consumption comparison of these two algorithms are listed in Table 4. The speed-up factor is defined as the ratio of sequential to parallel algorithm running time.

Table 4. Time Consumption Comparison

Problem instances	Running Time (Sequential algorithm)	Running Time (Parallel algorithm)	Speed-up
#1	20.50ms	3.51ms	5.8
#2	306.87ms	51.83ms	5.9
#3	104.082 sec	5.018 sec	20.7

From the experimental results, we can see that for large scale workflow scheduling problems, GPU-based parallel ant colony optimization performs better than CPU-based sequential algorithm and has great speedup (the solution precision is also guaranteed).

5. Conclusions

In the real application, the scientific workflow generally has thousands of task nodes. As the problem scale is huge, there are many feasible solutions in the solution space making computation intensive. The commonly used sequential optimization algorithm needs long time to obtain the optimal solution making it infeasible for practical scientific workflow scheduling. To speed up the problem solving, we propose GPU based parallel ant colony optimization algorithm. Scientific workflow scheduling is a kind of combination optimization problem, and ant colony optimization algorithm works well for combination optimization problems, performance of GPU based parallel algorithm is overwhelming compared with CPU-based algorithms, so GPU-based parallel ant colony optimization algorithm has great advantage to solve scientific workflow scheduling problems. Our proposed algorithm is implemented with CUDA C and compared with

 Copyright ⓒ 2015 SERSC

sequential algorithm. Experimental results show that our parallel algorithm obtain the same solution compared with sequential algorithm, but the speedup is great, about 5.8 times to 20.7 times speedups.

Acknowledgements

The authors are grateful to the anonymous referees for their valuable comments and suggestions to improve the presentation of this paper. This work was supported in part by a grant from the National Natural Science Foundation of China (No. 61211130359).

References

[1] M. Dorigo, "Optimization, learning and natural algorithms, Ph.D. dissertation", Dipartimento di Elettronica, Politecnico di Milano, **(1992)**.
[2] M. Dorigo, V. Maniezzo and A. Colorni, "The ant system, Optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man, and Cybernetics–Part B. vol. 26, no. 1, **(1996)**.
[3] M. Dorigo, T. Stützle, "Ant Colony Optimization", MIT Press, **(2004)**.
[4] Y. Liu, G. Wu, "Research on MPI-based parallel max-min ant system", Applied Mechanics and Materials, **(2012)**, pp. 198-199.
[5] U. Boryczka, J. Kozak and R. Skinderowicz, "Parallel ant-miner, parallel implementation of ACO techniques to discover classification rules with OpenMP" Proceedings of 15th International Conference on Soft Computing, **(2009)** June 24-26, Brno, Czech.
[6] A. Delévacq, P. Delisle, M. Gravel and M. Krahecki, "Parallel ant colony optimization on graphics processing units", Journal of Parallel and Distributed Computing, vol. 73, no. 1, **(2013)**.
[7] K. Kobashi, A. Fujii, T. Tanaka and K. Miyoshi, "Acceleration of ant colony optimization for the traveling salesman problem on a gpu", Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems, **(2011)** December 14-16, Dallas, TX, United States.
[8] J. M. Cecilia, J. M. García, A. Nisbet, M. Amos and M. Ujaldón, "Enhancing data parallelism for ant colony optimization on gpus", Journal of Parallel and Distributed Computing, vol. 73, no. 1, **(2013)**.
[9] A. Uchida, Y. Ito and K. Nakano, "An efficient GPU implementation of ant colony optimization for the traveling salesman problem", Proceedings of 2012 3rd International Conference on Networking and Computing, **(2012)** December 5-7, Naha, Japan.
[10] S. Benedict and V. Vasudevan, "Scheduling of scientific workflows using Niched Pareto GA for grids", Proceedings of 2006 IEEE International Conference on Service Operations and Logistics, and Informatics, **(2006)** June 21-23, Shanghai, China.
[11] K. Plankensteiner, R. Prodan and T. Fahringer, "Scheduling scientific workflows to meet soft deadlines in the absence of failure models. Proceedings of 16th International Euro-Par Conference on Parallel Processing, **(2010)** August 31- September 3, Ischia, Italy.
[12] M. Wieczorek, S. Podlipnig, R. Prodan and T. Fahringer, "Bi-criteria scheduling of scientific workflows for the grid. Proceedings 8th IEEE International Symposium on Cluster Computing and the Grid, **(2008)** May 19-22, Lyon, France.
[13] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Scientific Programming, 14, **(2006)**.
[14] J. Yu, R. Buyya and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids", Proceedings of 1st International Conference on e-Science and Grid Computing, e-Science 2005, **(2005)** December 5-8, Melbourne, Australia.
[15] NVIDIA Corp. CUDA Programming Guide Version 5.5, **(2013)**.
[16] NVIDIA Corp. CUDA C Best Practice Guide Version 5.5, **(2013)**.

Copyright ⓒ 2015 SERSC

Authors

Pengfei Wang, received M.A. degree in computer science from Southwest Jiaotong University, China. He is studying for the Ph.D. degree in systems engineering at Beijing Institute of Technology, China. His research interests are workflow management technology and intelligence algorithm.

Huifang Li, is a professor of systems engineering at Beijing Institute of Technology, China. She received the Ph.D. degree in systems engineering from Xi'an Jiaotong University, China. Her research interests are workflow management technology and cloud computing.

Baihai Zhang, is a professor of control science and engineering at Beijing Institute of Technology, China. He received the Ph.D. degree in control theory and control applications from Harbin Institute of Technology, China. His research interests are wireless sensor network and intelligence algorithm.

 Copyright ⓒ 2015 SERSC