IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

Novel Method to Improve ACO Performance on the GPU Using CUDA for Nurse Roster Scheduling Problem

Mr. A. P. Pande[1], Mr. B. S. Patil[2] Mr. A.U. Patil[3]

Department of Information Technology, P.V.P.I.T, Budhgaon Sangli, MH, India[1, 2, 3]

Abstract: This paper shows the accomplishment of parallel Ant Colony Optimization algorithm on the Graphics Processing Unit (GPU) to solve nurse roster scheduling problem (NRSP).We put on the Schedule formation and pheromone update phases of Ant colony Optimization using a data parallel method. We applied roulette wheel selection method for schedule formation and pheromone update. The parallel accomplishment of roulette wheel selection method considerably cuts the execution time of Schedule formation. Our new parallel accomplishment executes up to 8-12x faster than sequential execution at the same time as preserving the quality of the Schedules formation.

Keyword: CUDA, GPU, NRSP, NVDIA, ACO

I. INTRODUCTION

Ant Colony Optimization (ACO) [1] is a well-known population-based algorithm for modelling and solve discrete optimization problems. Ant algorithms model the comportment of real ants to elucidate diversity of optimization and disseminated control problem. We applied Ant Colony Optimization algorithm to solve Nurse Roster Scheduling Problem (NRSP) where the main objective is to attain the optimum schedule solution about a set of schedules. The easiest accomplishment of Ant Colony Optimization consists of two core phases one is Schedule construction and second is pheromone update. To improve quality of schedules, the additional local search stage also be applied after schedules have been constructed before accomplishment of the pheromone update phase. The method of Schedule formation and pheromone update is operated iteratively until a cessation requirement is meet up. The indirect communication of is can be obtained using a pheromone matrix. Each ant has formed a new schedule and update pheromone matrix. It will impact consecutive repetitions of the algorithm and the additional computation time required for schedule formation as the number of Nurses and number of day's increases, hence requires significant CPU time. So to improve computational time we implemented parallel ACO with roulette wheel selection method, the Schedule formation and pheromone update phases are performed individually for each ant which builds Ant Colony Optimization (ACO) remarkably appropriate to GPU parallelization.

The first key procedure for implementing ACO in parallel manner where each ant assigns to an individual processing element and organises a colony of ants. In second key procedure where intact colony of ants to a dispensation element usually improved with a method of inter connecting between the colonies. The Manifold colonies are accomplished in parallel, potentially diminishing the number of repetitions a for determination. For parallel programming, NVIDIA CUDA is a programming architecture for emerging general purpose applications for execution [2]. Compute Unified Device Architecture exposes the GPU's enormously parallel building so that parallel code can be written to accomplish significantly faster than its optimized sequential equivalent.

The parallel ACO implementations on the GPU using Compute Unified Device Architecture focus on both the implementation of the algorithm and the superiority of the solutions. Data parallel approach is applied to execute both phases of Ant Colony Optimization in parallel on the Graphics Processing Unit. For the Schedule formation phase, our method uses a new parallel implementation of the roulette wheel selection algorithm which is called DS-Roulette. DS-Roulette conducts the modern hardware architecture, extends parallelism, and reduces the execution time. For the pheromone update phase, we incorporate the methodology of MAX and MIN Ant colony System, and linked with our accomplishment.

II. RELATED WORK

ACO algorithms can be categorized as coarse grained or fine grained considering parallel implementation. The ants are individually represented to processing elements with communication between processing elements being ant to ant is the fine grained method, and entire colonies are represented to processing elements with communication between colonies to colony is course grained method. This section studies the existing parallel Ant colony Optimization techniques that focus the GPU. Catala et al. [1] explains the first GPU accomplishments of ACO directed at exhibiting the Delinquent. Their accomplishments depend upon a direct Graphics processing unit using graphics models to resolve general-purpose problems.

Jiening et al. [2] realised the MMAS algorithm to resolve the TSP. This work published prior to the edict of CUDA and their accomplishment was composite than its CPU equivalence. A parallel Schedule construction phase resulted in a slightly better performance

Fu et al. [3] applied a parallel MMAS for GPU to solve the Travelling Salesman Problem. Their technique focused on more MATLAB implementation and less on the GPU. They described a speedup, however, their relative CPU implementation was MATLAB-based which is fundamentally dawdling due to an interpreted language.
Zhu and Curry [4] designed a Search algorithm using ant colony optimization to resolve non-linear function problems using CUDA. They re-counted performance nearby2.5x over the sequential execution.

Bai et al. [5] explained a multiple colony version of MMAS using coarse-grained CUDA to resolve the Travelling Salesman Problem. Each ant colony is represented to thread block and inside block each thread is represented to an ant. This method produces Schedules with a feature analogous to the CPU accomplishment but the speedup re-counted up to 2x.

Weiss [6] developed a parallel form of Ant Miner GPU, an addition of the MMAS algorithm. The each ant inside the colony is signified to a separate CUDA thread. He claims that, method join with the Ant Miner GPU algorithm allows for considering larger population size. All phases of the work are moved to the GPU to avoid costly moves to and from the Graphics Processing Unit.

A. Data-parallelism

The ACO algorithm for solving the TSP on the GPU using CUDA is implemented by Cecilia et al. [3]. The existing task-based method of representing one ant per thread is fundamentally not matched to the GPU. With a task-based method, each thread must store each ant's memory. This method applicable for small Schedules but problematic with larger Schedules because of limited shared memory available. Other technique to use fewer threads per block, which reduces GPU usage or global memory used to store each ant's memory. Itintensely decreases the performance of kernels. The task-based parallelism is warp-branching. The ants construct a Schedule, and the execution paths of ants are generally differ due to conditional statements intrinsic for using roulette wheel selection on the output of the random proportional rule. All threads within the branch are serialized and execute sequentially until the branching section is complete for warp branches, thus significantly delaying the branching code performance. The warp divergence and memory issues are avoided byData parallelism. Data parallelism representing each ant to a thread block and all threads inside the thread block work in cooperation to perform a collective task such as Schedule construction. Here thread is responsible for a singular day and the likelihood of visiting a day can be calculated using a proportionate selection method known as I-Roulette [3] without branching the warp. For implementation of pheromone update phase on the GPU.The 5x speedup factor is reported when both the Schedule construction and pheromone update phases are executed on the GPU. The majority of the execution time spent on the Schedule construction phase.

III. IMPLEMENTATIONS

Our parallel implementation of the ACO algorithm for execution on the GPU is presented in this section. Here data parallel approach is implemented for representing each ant in the colony to a thread block. To maximize performance we executed each phase of the algorithm on the GPU.
The first phase of the algorithm constructs the nurse data and assigns memory and the relevant data structures. For any given nurses size n and days size d, the constraints are loaded into a matrix, for every pair of distinct nurse for each day. To store each ant's current Schedule and Schedule length ant memory is allocated. A pheromone matrix is initialized on the GPU to store pheromone levels and a secondary structure called choice info is used to store the product of the denominator. After completing initialization phase, using greedy search, the pheromone matrix is artificially scattered with a Schedule generated.

A. Schedule construction

This phase is applied repeatedly until a new Schedule is created. Algorithm shown in Figure 1 gives details about Schedule construction.

```
Procedure Construct Solution
Schedule[1]  = assign the ant on a random day
for j = 2 to n - 1 do
for l = 1 to n do
Prob[l] =CalProb(Schedule [1 :  j - 1],l)
end-for
Schedule[j]=RouletteWhlSelection(prob)
end-for
Schedule[n] =   remaining day
Schedule cost =CalScheduleCost(Schedule)
End
```

Figure 1: Pseudo code for construction of solution

After the initialization, the first inner for loop repeats n - 2 times to build an complete Schedule note that there are only n-2 choices to make as once n-2 days have been chosen. Within the inner for-loop, the probability of moving from the last visited day to all other possible days is calculated. Calculating the probability consists of two stages: retrieving the value of choice info[j][l] and checking if day l has already been visited in the current iteration in which case the probability is set to 0. The next day to visit is selected using roulette wheel selection.

TABLE I. ROULETTE WHEEL SELECTION

INPUT	REDUCED	NORMALIZED RANGE	RANGE
0:1	0:1	0:1	> 0:0 &<0:1
0:3	0:4	0:25	> 0:1 &< 0:25

0:2	0:6	0:375	> 0:25 &< 0:375
0:8	1:4	0:875	> 0:375 &< 0:875
0:2	1:6	1:00	> 0:875 &< 1:0

Roulette wheel selection is illustrated in Table. I where 1 item has to be chosen from 5in proportion to the value in the first column labelled 'input'. To obtain cumulative totals reduce the set of input values so reduced values are normalized so that the sum of all input values normalizes to 1 and the portion of the roulette wheel corresponding to some item is calculated. Generation of random number is final step that is between 0.0 to 1.0 is the last step however, the linear nature of the algorithm is the divergence in control flow, parallel random number generation for thread synchronization. The entire Schedule is stored in shared memory. But for large instance, shared memory is often exhausted. To address these problem we present Double-Spin Roulette(DS-Roulette) which is a highly parallel roulette selection algorithm that deeds warp-level parallelism, reduces shared memory dependencies, and decreases the overall instruction count which is performed by the GPU. In the sequential implementation of roulette wheel selection, each ant constructs a Schedule one day at a time and each ant is processed consecutively. For parallel implementation of Schedule construction phase using a data-parallel approach, each thread is assigned to each block so that m blocks occupied by mants.

B. Pheromone update

Pheromone update is the last stage of the ACO algorithm which consists of two phases, one is pheromone evaporation and second is pheromone deposit. The pheromone evaporation phase is small to parallelize as all edges are evaporated. A single thread block is propelledwhichassigns each thread to an edge and reduces the value using constant factor. An overlaying strategy is used to cover all edges. The second phase is pheromone deposit, which deposits a quantity of pheromone for each edge belonging to a constructed Schedule for each ant. To ensure correctness of the pheromone matrix the atomic operations must be used because of each ant perform this step is parallel.

Atomic operations are expensive as computational so alternative approach using scatter to gather transformations is used. In this approach it removes the dependency on atomic operations. To reduce the usage of atomic operations and increase convergence speed, we implement the pheromone update where each ant makes a single atomic in job on a memory value storing the Schedule length. This single operation per block allows the lowest Schedule value to be saved without extra kernels. For the Schedule construction phase we begin m thread blocks representing m ants where Schedule cost is equivalent to the deepest overall cost.

IV. EXPRIMENTAL RESULTS

In this section, we summarize the results obtained using above technique on various instances of the NRSP and results are compared to other parallel and sequential implementations. We use standard ACO parameters and reduce rate of evaporation from 0.5 to 0.1 on the pheromone matrix for both GPU and CPU implementations. The reduced evaporation rate confirms that the pheromone matrix still has a dequatepherom one to impact the Schedule construction. For analysis our implementation we used an NVIDIA GT 610 GPU and an Intel i3 CPU. Our implementation was written C language and compiled using the latest CUDA toolkit and executed on operating system windows 7 using Microsoft Visual Studio.

A. Solution Quality

To calculate the superiority of the Schedules formed, we compared the results of our GPU execution against an existing CPU execution for the set number of repetitions. Our new method was able to match and decrease the size of the Schedules constructed when using identical parameters and number of repetitions. Table 2 and Fig.2 illustrations an evaluation of the average superiority of Schedules acquired through the existing CPU and new GPU execution.

Table 2: Average solution quality

Nurse Roster Instance	CPU Average Solution Quality	CPU Average Solution Quality
Instance 20	0.85	0.92
Instance 21	0.75	0.8
Instance 22	0.8	0.9
Instance 23	0.7	0.8
Instance 24	0.75	0.8

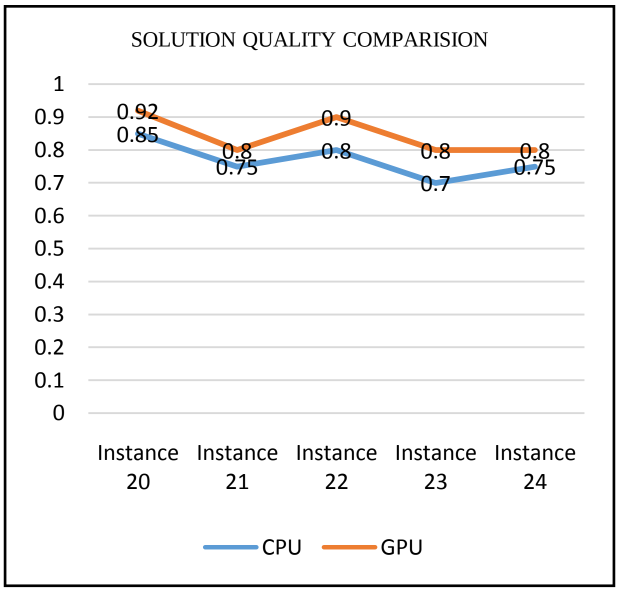
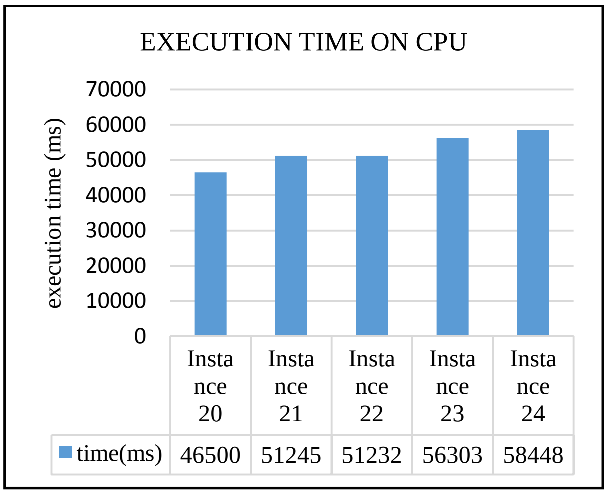
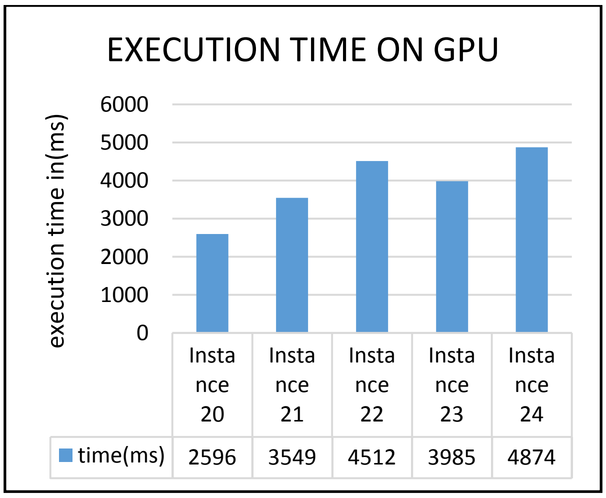


Figure 3: Compare average solution quality on cpu and gpu



	Instance 20	Instance 21	Instance 22	Instance 23	Instance 24
■ time(ms)	46500	51245	51232	56303	58448

Figure 3: Execution time on CPU



	Instance 20	Instance 21	Instance 22	Instance 23	Instance 24
■ time(ms)	2596	3549	4512	3985	4874

Figure 4: Execution time on GPU

The results shown in figure 3 & 4 proves that with GPU implementation with this novel method is 8-12x faster than the sequential execution. The Schedule construction Phase uses the more of the total execution time. The Schedule construction phase uses a new efficient execution of roulette wheel selection and it is able to fetch similar speedups to other algorithms limited by the execution time of proportionate range. The pheromone update execution is between 1-9x time faster than the existing CPU execution.

V. CONCLUSIONS

In this paper, we implemented a data-parallel GPU execution of the ACO algorithm to solve nurse roster scheduling problem. We implements both the construction of Schedule and pheromone update phases on the GPU. The obtained result shows an execution speed up 8-12x faster than the existing CPU execution. For large data sets, our algorithm handles share memory efficiently as compared to existing system. Overall an efficient parallel implementation of roulette wheel selection gives better performance than existing parallel and sequential implementation and we ensure that this parallel implementation of algorithm is more appropriate with other heuristic problem solving areas.

REFERENCES

[1] A. Catala, J. Jaen, and J. Modili, “Strategies for accelerating ant colony optimization algorithms on graphical processing units,” in IEEE Congres on Evolutionary Computation (CEC), Sept. 2007, pp. 492– 500.
[2] W. Jiening, D. Jiankang, and Z. Chunfeng, “Implementation of ant colonies algorithm based on GPU,” in Sixth Int. Conf. on Compter Graphics, Imaging and Visualization (CGIV), Aug. 2009, pp. 50–53.
[3] J. Fu, L. Lei, and G. Zhou, “A parallel ant colony optimization algorithmwith GPU-acceleration based on all-in-roulette selection,” in ThirdInt. Workshop on Advanced Computational Intelligence (IWACI), Aug.2010, pp. 260–264.
[4] J. M. Cecilia, J. M. Garc´ıa, A. Nisbet, M. Amos, and M. Ujaldon, “Enhancing data parallelism for ant colony optimization on GPUs,” J.Parallel Distrib. Comput., vol. 73, no. 1, pp. 42–51, 2013.
[5] W. Zhu and J. Curry, “Parallel ant colony for nonlinear function optimization with graphics hardware acceleration,” in Proc. Int. Conf. on Systems, Man and Cybernetics, Oct. 2009, pp. 1803–1808.
[6] H. Bai, D. Ouyang, X. Li, L. He, and H. Yu, “MAX-MIN ant systemon GPU with CUDA,” in Fourth Int. Conf. on Innovative Computing, Information and Control (ICICIC), Dec. 2009, pp. 801–804.
[7] A. Del`evacq, P. Delisle, M. Gravel, and M. Krajecki, “Parallel ant colony optimization on graphics processing units,” J. Parallel Distrib. Comput., vol. 73, no. 1, pp. 52–61, 2013.
[8] M. Dorigo, “Optimization, learning and natural algorithms,” Ph.D. dissertation, Dipartiento di Elettronic, Politecnico di Milano, Milan, Italy, 1992.
[9] M. Manfrin, M. Birattari, T. St¨utzle, and M. Dorigo, “Parallel ant colony optimization for the traveling salesman problem,” in Fifth Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS),ser. Lecture Notes in Computer Science, M. Dorigo, L. M. Gambardella,
[10] M. Birattari, A. Martinoli, R. Poli, and T. St¨utzle, Eds., vol. 4150. Springer Verlag, 2006, pp. 224–234.
[11] T. St¨utzle, “Parallelization strategies for ant colony optimization,” inFifth Int. Conf. on Parallel Problem Solving from Nature (PPSN-V). Springer-Verlag, 1998, pp. 722–731.
[12] T. St¨utzle and H. H. Hoos, “MAX-MIN ant system,” Future Gener. Comput. Syst., vol. 16, no. 9, pp. 889–914, Jun. 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=348599.348603
[13] D. Kirk and W.-M. W. Hwu, Programming Massively Parallel Processors:A Hands-on Approach. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.
[14] NVIDIA, “Inside Kepler,” http://developer.download. nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0642-GTC2012-Inside-Kepler.pdf
[15] Y. You, “Parallel ant system for traveling salesman problem on GPUs,” GPUs for Genetic and Evolutionary Computation, GECCO, http://www.gpgpgpu.com/gecko 2009 (last accessed 29/01/2013).
[16] W. W. Hwu, GPU Computing Gems Emerald Edition. Morgan Kaufmann, 2011.
[17] W.-M. W. Hwu, GPU Computing Gems Jade Edition. Morgan Kaufmann, 2011.
[18] M. Dorigo, “Ant Colony Optimization - Public Software,” http://iridia.ulb.ac.be/˜mdorigo/ACO/aco-code/ public-software.html (last accessed: 29/01/2013.
[19] NVIDIA, “CUDA C Programming Guide,” http://docs.nvidia.com//cuda/cuda-c-programing-guide/index.html (last accessed 29/01/2013).