Journal of Network and Innovative Computing
ISSN 2160-2174 Volume 4 (2016) pp. 200-208

Reducing travel time in VANETs using MACO with CUDA on GPU

Vinita Jindal [1], Punam Bedi [2]

[1,2]Department of Computer Science, University of Delhi, India
[1]vjindal@keshav.du.ac.in, [2]punambedi@ieee.org

***Abstract*- Over the past few years, with the advent of Compute Unified Device Architecture (CUDA), Graphics Processing Units (GPUs) have evolved into highly parallel programmable architecture and provide the solution to many real-world applications. In this paper, we deal with a GPU implementation of Modified Ant Colony Optimization (MACO) algorithm that reduces the overall travel time of the journey by avoiding congestion enroute. The GPU implementation provides the faster computation in order to further reduce the travel time for the vehicles on move. We are providing the parallel implementation for all the phases of MACO approach. The implementation is done using parallel architecture on the GPU at NVIDIA GeForce 710M using C++ language running with CUDA toolkit 7.5 on Microsoft Visual Studio 2013. The accuracy of the proposed parallel implementation of MACO algorithm is validated both graphically as well as statistically using hypothesis testing by comparing it with both, the parallel implementation of the standard Dijkstra algorithm and that of the existing MACO algorithm on a real world North-West Delhi map with an increased number of vehicles. The obtained results for the parallel implementation of MACO illustrate the significance in the reduction of travel time with 99% confidence.**

Keywords: Parallelization; MACO; Ant Colony Optimization; CUDA; Routing Problem; GPU; Hypothesis Testing.

I. Introduction

Graphic Processing Unit (GPU) was proposed by NVIDIA Inc. in 1999. GPU contain several independent computing units and many processing cores that enable faster computation and has been largely used in high performance computing (HPC). Generally, program based on GPU acceleration is several times faster than that of one which based on CPU [1], [2].

Vehicular Ad-hoc NETwork (VANET) is a special kind of decentralized ad-hoc network which consists of nodes representing the vehicles and edges signifying the distance between these nodes on underlying roads. In VANETs, reducing the travel time on roads through routing mechanism is one of the major issues. It always attracts researchers to design certain mechanism that can solve this problem efficiently in reasonably less time [3].

Finding a congestion free route can be one of the solution that may allow the commuters to reach the destination faster, provided that path may be a bit larger than the shortest and jammed path [4]. In general, it is a greedy human behavior that they are always seeking for shortest route to reach the destination. This can eventually cause congestion in heavy traffic conditions [5]. In real time, congestion increases if vehicles do not take optimized routes to reach their destination. The shortest route to the destination is not always the optimal one. However if all vehicles keep following their respective shortest routes, this will result in congestion on some set of routes during peak hours.

Ant Colony Optimization (ACO) algorithm was originally proposed by Dorigo in 1992, which solves the combinatorial optimization problems [6]. Authors in [7] modifies the original ACO and propose a Modified Ant Colony Optimization (MACO) algorithm that provides the optimized routes with the avoidance of the congestion enroute. MACO is the variation of the classical ACO in which repulsion effect is used instead of attraction towards pheromones for avoiding congested routes and dispersing the traffic towards paths with less pheromone value. Their approach was able to significantly reduce the overall travel time of the journey. All their work in the algorithm was carried by a single processor and hence computation time is more in case of large number of vehicles.

To overcome this drawback, parallel computing was introduced in which many computations are carried out simultaneously by multiple processors. Here, a large problem is divided into multiple small sub problems and each of them can be executed simultaneously on the different processors. This reduces the computation time as compared to the serial execution of the same problem. Parallel Computing mainly uses Flynn's Taxonomy in which it uses SIMD (Single Instruction Multiple Data). In this approach the instruction remains constant, but the data changes continuously. This

increases the data handling capacity of the processor and hence can reduce the computational time of the algorithm.

Due to the intensive computation of large scale optimization problems, parallel hardware is needed to finish computation timely. There are several GPU implementation of ACO on different domains like Travelling Salesmen Problem, Resource Constrained Project Scheduling Problem and Scientific workflow scheduling Problem to name a few [1], [8], [9], [10]. To provide and tackle the issues in complex problems, a large number of fitness function evaluations are needed to obtain an acceptable solution and it require large computation and time that can be optimized by using the GPUs to accelerate the speed of computation [11], [12].

In the paper we are proposing the parallel implementation of MACO to further reduce the overall travel time by providing the faster computations. The MACO approach comprises of route construction, pheromone deposition and pheromone evaporation. In the literature to the best of our knowledge, only route construction path has been implemented on GPU for ACO [13], [14]. But in our proposed work we are parallelize all the function used in the MACO algorithm to provide faster computations using GPU and hence reduce the overall travel time [15]. Experimental results show that the proposed work is successfully reducing computational time with increase in congestion. We have implemented our parallel algorithm using CUDA 7.5 toolkit on NVIDIA GEFORCE 710M GPU. We have hardcoded a network obtained from real world map of North-West Delhi. The obtained results were compared with both, the parallel implementation of standard Dijkstra's algorithm and the MACO algorithm proposed by [7]. Results show the significant improvement in reduction of overall travel time of the vehicles by the proposed technique under heavy traffic conditions.

A. Overview of GPU (Graphics Processing Unit)

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate various real time complex applications like analytics, enterprise, scientific, consumer and engineering. GPU consist of a massively parallel architecture with thousands of smaller, more efficient cores that are designed for handling multiple tasks simultaneously [16]. As shown in Figure 1 below, CPU contains multiple cores whereas GPU is having thousands of small cores. GPU can handle millions of lightweight threads simultaneously and significantly accelerate the data processing speed.

Due to rapid increase of performance of Graphics processing units (GPUs) as compared with traditional CPUs, GPUs have become a popular computing architecture in recent years. The simplified architecture of a NVIDIA GPU is shown in Figure 2 below. Here, GPU is divided into two major parts, Host and Device. Host refers to CPU and Device refers to GPU. Device is further divided into Grids and Blocks on which multiple threads can work in parallel through various memories. GPUs are consisting of four types of memories: Shared memory, Global memory, Constant memory and Texture memory to provide the storage and communication in between the threads. From 2007, NVIDIA has opened the possibility of programming GPUs using a specific language called CUDA defined in next section. We are using CUDA 7.5 toolkit for the implementation of MACO algorithm. In MACO algorithm, each thread will perform task as of independent ant. Independently route construction, pheromone deposition and pheromone evaporation can be done by each thread in parallel.

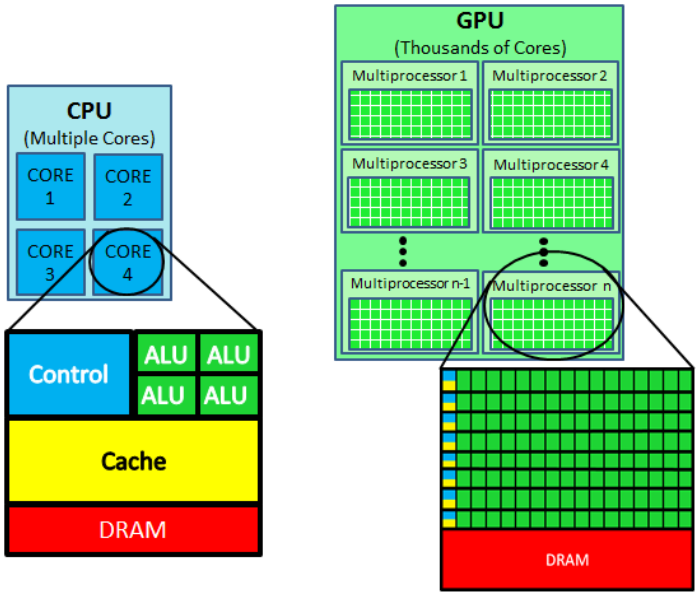

Figure 1: CPU vs. GPU architecture

B. Overview of CUDA (Compute Unified Device Architecture)

CUDA was first introduced by NVIDIA in 2007. It was developed for both Windows and Linux platform and later versions were also compatible with Mac OS. NVIDIA has developed CUDA especially for NVIDIA Graphic Cards environment and allows a great resource tool used to solve problems with a high degree of computational complexity [16], [17]. It is a unique programming language for the NVIDIA Graphic cards as it uses the all the cores simultaneously in Graphic Process Unit, GPU of the graphic card efficiently. To use the CUDA architecture, extended C language is used to program on NVIDIA cards. Kernel is executed in GPU which is written in C/C++ language. Kernel is executed as many times as selected by the programmer through number of threads. The building blocks of GPU are Grids. Grids are further divided into blocks and the blocks are divided into threads. There are various parallel computing platforms available as MPI (Message Passing Interface), OpenCL (Open Computing Language) and CUDA. This paper is implemented using CUDA toolkit version 7.5.

CUDA is software and hardware platform designed for general purpose computing on GPUs in order to take full

advantage of the maximum performance of GPUs in applications [18]. In CUDA computing model, threads are grouped into thread blocks, and threads within thread block can cooperate among themselves using synchronization primitives by sharing data via a global memory and shared memory. The advantage of the global memory is that it can be accessed by all threads directly, whereas the shared memory is only accessible to threads of one block. Compared to the global memory, the shared memory is considerably smaller and significantly faster. The data can be partitioned and fetched into the shared memory to provide higher throughput for more complex operations. While these architecture specifics of GPUs allow fine-grained parallelization for impressive increase of performance, it requires adaptation and re-formulation of algorithms resulting in more effective design on the GPU.

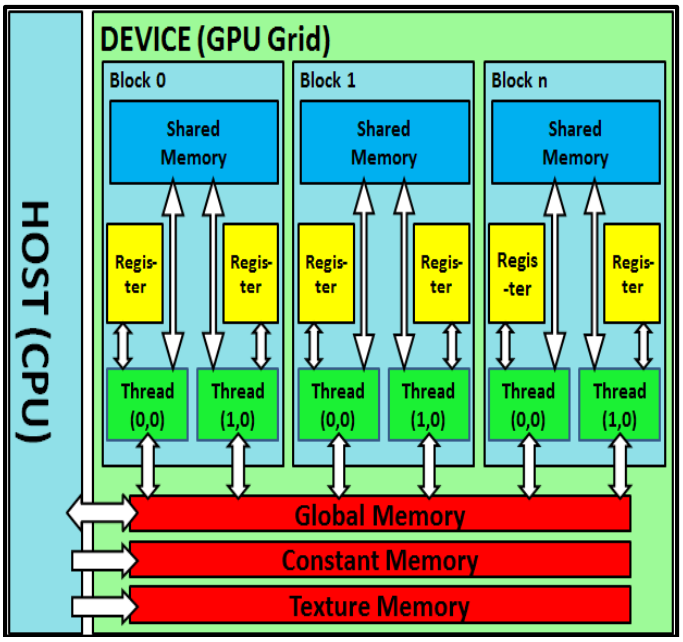

Figure 2: Simplified Architecture of a NVIDIA GPU

A typical CUDA-capable GPU is organized into an array of highly threaded streaming multiprocessors (SMs). These SMs form a building block; however, the number of SMs in a building block can vary from one generation of CUDA GPUs to another generation. Also, each SM has a number of streaming processors (SPs) that share control logic and instruction cache. Each GPU currently comes with up to 4 gigabytes of graphics double data rate (GDDR) DRAM, referred to as global memory. These GDDR DRAMs differ from the system DRAMs on the CPU motherboard in that they are essentially the frame buffer memory that is used for graphics. For graphics applications, they hold video images, and texture information for three-dimensional (3D) rendering, but for computing they function as very-high-bandwidth, off-chip memory, though with somewhat more latency than typical system memory. For massively parallel applications, the higher bandwidth makes up for the longer latency. The processing on CUDA is done in six steps. Here, first memory is being allocated on GPU device. Next, processing data is being copied into device through memory and then CPU host instructs the processing instructions to GPU device. This device executes the code in parallel. After completion of processing, the results are being copied back to memory and device memory is released. The CUDA processing flow is shown in Figure 3.

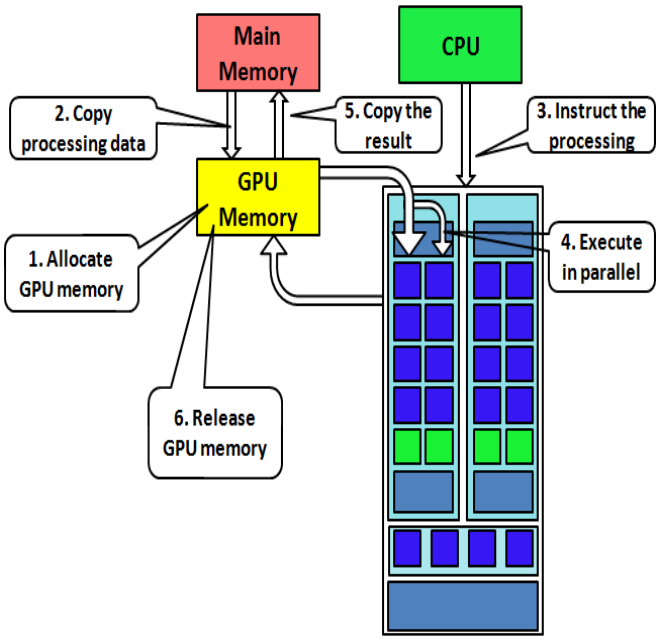

Figure 3: CUDA Processing flow

C. Overview of MACO (Modified ACO)

The MACO was developed to avoid the congestion in a city environment. It uses the existing Dijkstra's algorithm in low traffic conditions and modified ACO approach in heavy traffic conditions. Initially it selects the shortest path through Dijkstra's algorithm, and then uses MACO for optimization. MACO was used in a decentralized environment, where vehicles are treated as ants that leave pheromones on the trailed paths. When the accumulated pheromone trail reaches to a threshold value, the ant will be using the repulsion effect introduced in the pheromone behavior. On contrary from the classical ACO approaches, pheromone levels produce repulsion for other ants to avoid the congestion rather than an attraction [19], [20], [21]. In the absence of significant traffic, MACO behaves like the Dijkstra algorithm. In case of heavy traffic conditions, the deposition of the pheromone gradually increases due to large number of vehicles on roads. With time, the pheromone level is decreased using the evaporation process. Therefore, the working of the whole system depends on the deposition and evaporation rates of the pheromone as in each time step; every vehicle sense the updated pheromone

value based on which the decision for the route is taken. We have compared the MACO algorithm [7] with our proposed parallel implementation of MACO to show the significant reduction in overall travel time for the whole journey. The parallel implementation of MACO is also compared with the parallelized standard Dijkstra's algorithm to show its significant improvement in reducing the overall travel time for journey. The pseudocode for the MACO algorithm is given in Figure 4.

This paper is organized as follows: Section 2 presents the literature survey. The proposed parallel MACO algorithm is presented in Section 3. Section 4 describes the experimental setup and the obtained results. Finally, Section 5 concludes the paper.

II. Literature Survey

Swarm intelligence is a large field used in AI that includes the study of the behavioral patterns of living creatures like ants, bees, birds, termites and other social insects in order to model any processes. Swarms have the ability to solve complex tasks that are otherwise difficult to solve through existing computer algorithms. Ant colony optimization (ACO) is one of the most popular algorithms applied largely it networking domain to create self-organizing methods for routing related problems [6]. Ant colony optimization was proposed by Dorigo et al in early 1990s [21].

ACO is a meta-heuristic technique motivated through the communication strategy used by the real ants to solve an optimization problem and to discover minimum cost path in a network with given constraints. They make use of a chemical substance called pheromone for the exchange of information among them. In order to communicate with other ants, the ant leaves behind pheromone on its route. The other ants detect the presence of this pheromone and follow the path where concentration of pheromone is higher. Also if a particular route is not followed for some amount of time the pheromone starts evaporating thereby reducing the significance of that route. Selection of path based on pheromone trail by ants is a pseudo random process. It plays a major role in simulation of ACO algorithm [21]. Travelling salesman problem and quadratic assignments problem were amongst initial problems making use of ant colonies [22]. Many other domains also exist in literature that makes use of ACO technique for optimization [23], [20].

In the context of traffic simulation, the idea as adopted by real ants can be implemented to find out optimal path for the vehicles and also allowing communication between vehicles using pheromone. The pheromones in this traffic simulation can be taken as the characteristic of lanes or roads that are updated by each vehicle crossing that lane. Through this characteristic, vehicles get information about the traffic on roads thereby indirect communication takes place between vehicles. The pheromone value on every lane gives the indication of whether there is congestion ahead or not. This makes the vehicles capable to check for the congestion free route.

Several approaches using ACO have also been made in the traffic area [24], [19], [25]. The authors in [26] proposed a genetic approach for traffic light control and pedestrian crossing. Another researchers in their paper [27] use ACO with link travel time prediction in order to find routes to reduce travel times. A modification to the ACO algorithm is presented by [7] to reduce the overall travel time of the journey in the MACO algorithm. In this algorithm, all the steps are executed in sequential manner. As there are large numbers of vehicle in the system with dynamic nodes, computation process of the algorithm is time consuming. Efforts have been made in our work to improve the computation time by parallel implementation of the algorithm which will be helpful in reducing the overall travel time of the vehicles on move.

Despite the fact that ACO has been implemented using the parallelization on GPU by various researchers [14], [28], [13] in their work, it was found that none of them has implement all the parts of the algorithm in parallel as per best of our knowledge. The ACO consists of route planning, pheromone deposition and pheromone updation phase. Most of the earlier implementation parallelizes only the route planning phase. In our proposed work, we are implementing all the phases of the MACO algorithm in parallel to reduce the computation time of the algorithm and hence resulting in reduction of the travel time. MACO was proposed to reduce the overall travel time in VANETs [7]. The drawback of the algorithm was that it runs on a single CPU and in VANETs, where the number of vehicles is large; it will take large time for computations. In our proposed work, parallel processing using GPU is used to overcome this problem. The proposed parallel implementation of MACO is described in next section.

III. Proposed Parallel Implementation of MACO

MACO algorithm finds the optimal path for the vehicles by introducing the repulsion effect in pheromone to avoid the congestion in VANETs. The pheromone is a characteristic of road that is updated by each vehicle while crossing that road. Through this, vehicles get information about the vehicles on the roads thereby indirect communication takes place between vehicles. The pheromone value on every lane gives the indication of whether there is congestion ahead or not. This allows the vehicles to beware in advance and allows them to change their path in order to avoid the congested route. With this, the vehicles follow a path that may have a path length greater than or equal to the shortest path but it reduces their overall travel time from source to destination. In the algorithm, deviation of path takes place only in case of congestion.

MACO algorithm runs on a single CPU. There is large number of vehicles in VANETs with dynamic topology that require large amount of computation at faster speed. To increase the speed of computations, we are proposing the parallel version of MACO algorithm that makes use of GPU programming and

run the tasks in parallel, hence reducing the decision making time. It will allow the driver to take decision early and react accordingly and reduces the overall travel time as compared with its non-parallel counterpart. The parallel implementation of the algorithm is given in Figure 5 and Figure 6 as follows:

MACO Algorithm

The algorithm is using the following parameters:

t=0 //simulation steps.

Ph_i : value of pheromone on lane i.

$present_v$: current lane of vehicle v

$destin_v$: destination of vehicle v

N_t : number of vehicles in simulation at time t

```
While (True)
  t++
  for every vehicle, v in simulation
   Label: if present_v  = destin_v
                return
          else
              Route_v = set of all lanes leading to destin_v
           for every lane, ln on route of v
                update pheromone on ln: Ph_ln++
            for all subsequent lanes, k leading to destination
                select the lane with minimum Ph:
                 result= min[Ph_k]
             endfor
             k=lane corresponding to value of result
             if  N_t> threshold
                    deviate vehicle to minimum Ph lane
                     set present_v = k
              end if
             update pheromone on exit of v:  PH_ln --
         continue for loop
      go to Label
   continue for loop
continue while loop
```

Figure 4: Pseudocode for MACO Algorithm

Algorithm: CUDA_MACO (coordinates)

1. Allocate device memory using *cudaMalloc* function.
2. Copy inputs from host to device using *cudaMemcpy* function.
3. Invoke CUDA_MACO_*KERNEL* <<<grid size, block size>>> (coordinates, distances, route)
4. Copy results back from device to host using

Figure 5: CUDA_MACO Algorithm

In the parallel implementation of MACO algorithm, we hard coded the coordinates matrix obtained from the real time North-West Delhi map obtained using Google maps. In the CUDA_MACO function, first of all CUDA memory will be allocated to all the variables required by the device using cudaMalloc function. Then each of the variables will be copied in the device individually by using cudaMemcpy with HostToDevice option. Next, CUDA_MACO_KERNEL will be invoked on the grid with two parameters: grid size and block size. The entire tasks running on this kernel are parallelized. Next, results variables have to copied back from device to host by using cudaMemcpy with DeviceToHost option. At last, CUDA memory needs to be freed using the cudaFree function.

Algorithm: *CUDA_MACO_KERNEL* (coordinates, distances, route)

1. Invoke device function *Calculate_Distance* (coordinates, distances) to calculate distances among various coordinates.
2. Assign each thread for each node in the network
3. Initialize pheromone for each edge with a random value.
4. Find next node using the value of pheromone and threshold in order to avoid the congestion
5. Invoke device function *Evaluate_Route* (route, distances) to calculate the overall travel time.
6. Invoke device function *Increase_Pheromone* (route, distances).
7. Invoke device function *Evaporate_Pheromone* (route, distances)

Figure 6: CUDA_MACO_KERNEL Algorithm

In *CUDA_MACO_KERNEL* function, we define four device functions namely: *Calculate_Distance*, *Evaluate_Route*, *Increase_Pheromone* and *Evaporate_Pheromone* that will be called by the kernel to execute the tasks in parallel. Kernel will assign a random value to the pheromones and assign each thread corresponding to each node in the network. The selection of the next node is done based on the threshold and pheromone values. The *Calculate_Distance* function will

calculate the distance matrix from the coordinate matrix using the standard distance formula. *Evaluate_Route* function will calculate the overall travel time for all the vehicles in the system. *Increase_Pheromone* and *Evaporate_Pheromone* will update the value of the pheromone for each thread in parallel. After the parallelization, the speed of computation will get increased and we get results faster. This will give some time to driver for react and behave accordingly. This helps in reducing the overall travel time for the whole journey. Algorithm 1 presented in Figure 5 runs on the CPU while Algorithm 2 presented in Figure 6 runs on the GPU. In algorithm 1, at line 3, GPU kernel is being invoked and running of algorithm 2 starts parallel. Whole experimental setup with the results obtained is discussed in the next section.

IV. Experimental Study and Results

The parallel implementation of MACO algorithm was implemented for the vehicular environment on a PC with 2 GB RAM, Intel core i5-3230M 2.6 GHz processor running windows 8.1 with NVIDIA GeForce 710M. The graphics card has 2GB dedicated RAM. Application was written in CUDA 7.0 toolkit and C/C++ using Visual Studio 2010. NVIDIA graphic driver version 347.62 was used for CUDA compatibility. CPU applications were written in C language using its standard library. The algorithm was able to reduce the total travel time taken by vehicles to reach destination by introducing parallelization for all the functions for reducing the computation time of the MACO algorithm as discussed above. In the algorithm, the default path followed by vehicles is decided by Dijkstra's routing algorithm in the beginning. So, we are comparing the results obtained by the proposed parallel MACO with the parallel implementation of Dijkstra's algorithm along with that of the MACO algorithm. The simulations were performed for a hardcoded real world network of North-West Delhi map and obtained results were tested with different number of vehicles. In a real world map of North-West Delhi (NWD), obtained from OSM, we have 128 roads and 52 junctions. The snapshot for the network used in experiments is shown in Figure 7.

We have implemented both standard Dijkstra shortest path algorithm in parallel and MACO algorithm for the above specified network to test the performance of the proposed algorithm. Results obtained for overall travel time by all the three algorithms are shown in Table 1 with their graphical representation shown in Figure 8. It is clearly evident from the results that the proposed parallel MACO algorithm shows a significant decrease in the travel time taken by vehicles as compared with both Parallel Dijkstra's and the MACO algorithm.

Experiments are repeated 50 times to obtain the results. The results conclude that proposed parallel MACO is able to reduce overall travel time by 60% to 71% when compared with the parallel implementation of the standard Dijkstra's algorithm and 50% to 54% when compared with MACO algorithm. Thus, it will help commuters to reach to their destination faster by reducing the overall travel time and hence providing a solution to an important research problem in transportation.

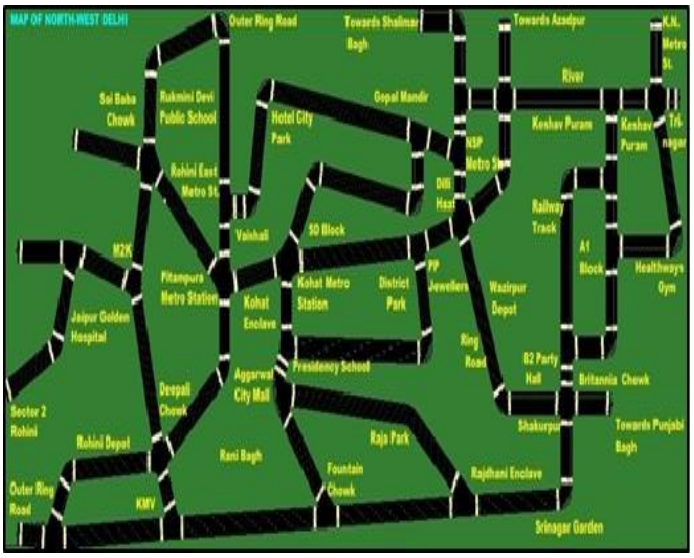

Figure 7: Screen shot for real time North-West Delhi network

Table 1. Results obtained in real world North-West Delhi network

No. of Vehicles	Overall Travel Time (in sec)		
	Parallel Dijkstra	MACO	Parallel MACO
100	478	379	189
200	549	428	213
300	734	513	252
400	862	575	281
500	912	602	294
600	1126	742	362
700	1364	894	435
800	1448	948	458
900	1570	1025	487
1000	1798	1174	549
1100	1895	1215	576
1200	2042	1303	617
1300	2189	1392	657
1400	2337	1481	698
1500	2475	1569	739
1600	2632	1657	779
1700	2779	1746	820
1800	2926	1835	861
1900	3074	1923	902
2000	3221	2012	942

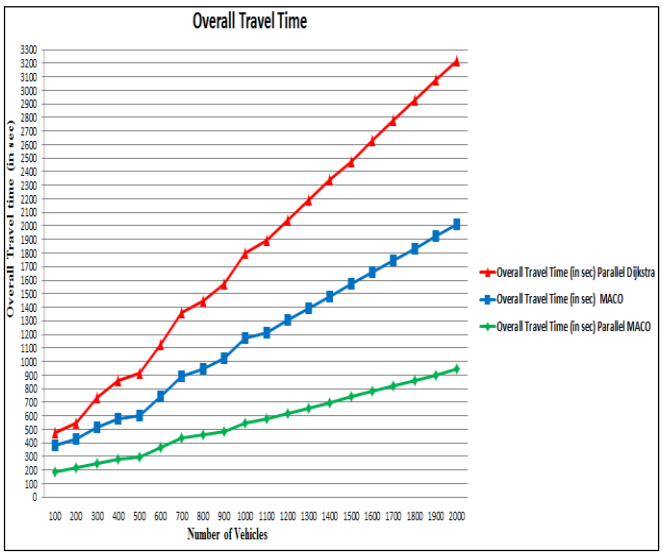

Figure 8: Graphical representation of computation time in real world North-West Delhi network

We use the statistical hypothesis testing using t-test for accepting the performance of the proposed parallel implementation of the MACO approach over standard Dijkstra and MACO algorithm proposed by the authors in [7]. The t-test is used to perform a hypothesis test of the null hypothesis that the overall travel time in parallel implementation of MACO is less than that of parallel implementation of Dijkstra as well as MACO algorithms. We are using t-test: paired two samples for means assuming unequal variance. In our experiment Null hypothesis is defined as follows:

$$H_0: \mu_1 \geq \mu_2$$

Further, alternate hypothesis is defined as

$$H_1: \mu_1 < \mu_2$$

Where μ_1 is the mean of first algorithm and μ_2 is the mean of second algorithm under t-test. In the paper, μ_1 refers to the mean of parallel implementation of MACO algorithm. μ_2 refers to the mean of parallel implementation of Dijkstra or MACO algorithms respectively as per the case in consideration. In significance testing, T, a test statistic is used to estimate whether observed data belongs to the null hypothesis or not. If the observed test statistic differs significantly from the hypothesized values, then null hypothesis is rejected and we claim the better performance of proposed algorithm under examination. For the experimentation, we have selected two different significance levels for α equals to 0.01 and 0.05 respectively.

The statistical hypothesis testing is performed for all the three algorithms used during experimentation in consideration (i.e. parallel implementation of MACO, parallel implementation of Dijkstra and MACO algorithms) for a network generated from real time North Delhi map.

In the statistical hypothesis, probability, p is considered. If p is less than 0.01, then null hypothesis is rejected and alternate hypothesis is accepted with 99% confidence. Otherwise, p is check again with another value of α, null hypothesis is rejected in case of less value than that of 0.05 and alternate hypothesis is accepted with 95% confidence.

First, we consider statistical hypothesis testing using t-test for parallel implementation of Dijkstra and parallel implementation of MACO algorithms. The mean and variance for parallel implementation of Dijkstra was found to be 1820.55 and 760732.89 respectively. For parallel implementation of MACO, mean and variance was 555.55 and 58149.84 respectively. T-stat was recorded as 6.25 for parallel implementation of Dijkstra and parallel implementation of MACO algorithms. p obtained for one tail was 0.000001, which is less than both α's i.e. 0.01 and 0.05. Hence null hypothesis is rejected and alternative hypothesis is accepted with 99% confidence.

Next, statistical hypothesis testing using t-test for MACO and parallel implementation of MACO algorithms are to be considered. The mean and variance value for MACO was 1170.65 and 275067.50 respectively. For parallel implementation of MACO, a mean and variance value was 555.55 and 58149.84 respectively. T-stat was 4.77 for MACO and parallel implementation of MACO algorithms. p obtained for one tail was 0.000029, which is less than both $\alpha = 0.01$ and $\alpha = 0.05$. Thus null hypothesis is rejected and alternative hypothesis is accepted with 99% confidence.

Hence, it is inferred that the proposed parallel implementation of MACO approach is outperforming both parallel implementation of Dijkstra and MACO approaches with 99% confidence.

V. Conclusion

In VANETs, large number of vehicles and limited road capacity leads to congestion, which results in increase in overall travel time. Hence, there is a need of some mechanism by which one can take decision faster to avoid the congestion and reach the destination faster by reducing overall travel time. MACO algorithm is presented in literature to reduce the congestion, but due to its serial implementation, it is comparatively slow in execution. Thus, in this paper a parallel implementation of MACO algorithm using CUDA toolkit 7.5 on NVIDIA GeForce 710M architecture that provides us the advantage of using both CPU and GPU's processors for giving results faster in order to react quicker and hence reduce the overall travel time is presented. To validate the approach, a road network model was obtained from a real world map of North-West Delhi and hardcoded into the system. The algorithm has been simulated using CUDA toolkit 7.5 in C++ language on NVIDIA GPU with Visual Studio 2013. The results obtained by parallel implementation of the MACO algorithm were compared graphically and statistically with the results obtained through both, the parallel implementation of

standard Dijkstra's algorithm and the existing MACO algorithm.

In the experiments, it was found that the use of parallel MACO algorithm reduced the overall travel time approximately by 60% - 71% with the increase in number of vehicles as compared with parallel implementation of Dijkstra's algorithm. Further, comparing it with the existing MACO algorithm it was found that travel time has been reduced approximately by 50% - 54% by the parallel MACO algorithm. Hence, both experimental and statistical results show that, the proposed algorithm reduces the overall travel time in comparison to the time involved in prevailing approaches with 99% confidence.

Acknowledgment

The authors duly acknowledge the University of Delhi for the support in work on this paper under the research grant number RC/2015/9677.

References

1. Wang, P., Li, H., Zhang, B.: A GPU-based Parallel Ant Colony Algorithm for Scientific Workflow Scheduling., 37-46 (2015)
2. Bedi, P., Jindal, V., Dhankani, H., Garg, R.: ATSOT: Adaptive Traffic Signal Using mOTes. In : 10th International Workshop on Databases in Networked Information Systems, DNIS 2015, Japan, vol. LNCS 8999, pp.152-171 (2015)
3. Habtie, A., Abraham, A., Midekso, D.: In-Vehicle Mobile Phone-Based Road Traffic flow Estimation: A Review., 331-358 (2013)
4. Mitra, S., Ghosh, T.: Congestion Control and Revocation of Misbehaving Vehicles in VANET., 43- 54 (2013)
5. Bell, M., Bonsall, P., Leaky , G., May, A., Nash, C., O'Flaherty, C.: Transport Planning and Traffic Engineering. John Wiley & Sons, NY, United State of America (1997)
6. Dorigo, M., Caro, G., Gambardella, L.: Ant Algorithms for Discrete Optimization. Artificial Life 5(2), 137-172 (1999)
7. Jindal, V., Dhankani, H., Garg, R., Bedi, P.: MACO: Modified ACO for reducing travel time in VANETs. In : Third International Symposium on Women in Computing and Informatics (WCI-2015), ACM, Kochi, India, pp.97-102 (2015)
8. Cecilia, J., García, J., Nisbet, A., Amos, M., Ujaldón, M.: Enhancing data parallelism for Ant Colony Optimization on GPUs., 42-51 (2013)
9. Bukata, L., Šůcha, P., Hanzálek, Z.: Solving the Resource Constrained Project Scheduling Problem using the parallel Tabu Search designed for the CUDA platform., 58-68 (2015)
10. Alejandro, F.-P., S., G.-R.: Meta-Heuristic Algorithm based on Ant Colony Optimization Algorithm and Project Scheduling Problem (PSP) for the Traveling Salesman Problem., 173 - 182 (2013)
11. Tan, Y., Ding, K.: A Survey on GPU-Based Implementation of Swarm Intelligence Algorithms., 1-14 (2015)
12. Laua, M., Srinivasan, R.: A hybrid CPU-Graphics Processing Unit (GPU) approach for computationally efficient simulation-optimization., 49-62 (2016)
13. Cecilia, J., Garcia, J., Ujaldon, M., Nisbet, A., Amos, M.: Parallelization strategies for ant colony optimisation on GPUs. In IEEE, ed. : IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp.339-346 (2011)
14. Fu, J., Lei, L., Zhou, G.: A parallel ant colony optimization algorithm with GPU-acceleration based on all-in-roulette selection. In IEEE, ed. : Third International Workshop on Advanced Computational Intelligence (IWACI), pp.260-264 (2010)
15. Jindal, V., Bedi, P.: Reducing Travel Time in VANETs with Parallel Implementation of MACO (Modified ACO). In Publishing, S., ed. : Innovations in Bio-Inspired Computing and Applications, Kochi, India, pp.383-392 (2016)
16. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General Purpose GPU Programming. Addison-Wesley, United States (2010)
17. Kirk, D., Hwu, W.-m.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann Publishers, Elsevier, USA (2010)
18. Kirk, D., Hwu, W.-m.: Programming Massively Parallel Processors-A Hands-on Approach. Morgan Kaufmann, Elsevier (2010)
19. Bedi, P., Mediratta, N., Dhand, S., Sharma, R., Singhal, A.: Avoiding Traffic Jam Using Ant Colony Optimization - A Novel Approach. In : International Conference on Computational Intelligence and Multimedia Applications, Sivakasi, Tamil Nadu, India, vol. 1, pp.61-67 (2007)
20. Bell, J., McMullen, P.: Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 18, 41–48 (2004)
21. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, USA (2004)
22. Elloumia, W., Abeda, H., Abra, A., Alimi, A.: A comparative study of the improvement of performance using a PSOmodified by ACO applied to TSP. Applied Soft Computing 25, 234–241 (2014)
23. Deneubourg, J., Aron, S., Goss, S., Pasteel, J.: The self-organizing exploratory pattern of the Argentine ant. Journal of Insect Behaviour 3(2), 159-168 (1990)

24. Nanda, B., Das, G.: Ant colony optimization: a computational intelligence technique. International Journal of Computer Commmunication Technology 2(6), 105–110 (2011)
25. Rizzoli, A., Montemanni, R., Lucibello, E., Gambardella, L.: Ant colony optimization for real-world vehicle routing problems : From theory to applications. Swarm Intelligence 1, 135-151 (2007)
26. Turky, A., Ahmad, M., Yusoff, M.: The use of genetic algorithm for traffic light and pedestrian crossing control. International Journal of Computer Science and Network Security 9(2), 88–96 (2009)
27. Claes, R., Holvoet, T.: Ant colony optimization applied to route planning using link travel time prediction. In : International Symposium on Parallel Distributed Processing, pp.358-365 (2011)
28. Dawson, L., Stewart, I.: Improving Ant Colony Optimization performance on the GPU using CUDA. In IEEE, ed. : IEEE Congress on Evolutionary Computation (CEC), pp.1901-1908 (2013)

Author Biographies

Vinita Jindal is currently pursuing Ph. D. under the supervision of Dr. Punam Bedi from the Dept. of Computer Science, University of Delhi. She is working in the area of Vehicular Ad-hoc Network. Earlier, she completed her M.C.A. in 2000 and M.Phil. in 2007. Since then she has been working as Assistant Professor in Keshav College, University of Delhi.

Punam Bedi received her Ph.D. in Computer Science from the Department of Computer Science, University of Delhi, India in 1999 and her M. Tech. in Computer Science from IIT Delhi, India in 1986. She is an Associate Professor in the Department of Computer Science, University of Delhi. She has about 30 years of teaching and research experience and has published about 200 papers in National/International Journals/Conferences. Dr. Bedi is a member of AAAI, ACM, senior member of IEEE, and life member of Computer Society of India. Her research interests include Vehicular ad-hoc networks, Mobile ad-hoc network, Web Intelligence, Soft Computing, Semantic Web, Multi-agent Systems, Intelligent Information Systems, Intelligent Software Engineering, Intelligent User Interfaces, Requirement Engineering, Human Computer Interaction (HCI), Trust, Information Retrieval and Personalization and Recommender Systems.