Implementing a Pickup and Delivery Problem with Time Windows Algorithm on a GPU Cluster

Geeratiya Srimool, Putchong Uthayopas
Department of Computer Engineering
Faculty of Engineering, Kasetsart University
Bangkok, Thailand
gigs@hpcnc.com, pu@ku.ac.th

Juta Pichitlamkhen
Department of Industrial Engineering
Faculty of Engineering, Kasetsart University
Bangkok, Thailand
Pichitlamken.j@gmail.com

***Abstract*- This work presents an implementation of a high speed Pickup and Delivery Problem with Time Window (PDPTW) problem using GPU cluster. This problem represents a class of a major logistic problem. The software implemented is tested on 8 nodes GPU cluster equipped with two of Tesla M2050 (448 cores) card on each node. The result shows a speedup of nearly 7 times for a small problem and 43 times using 4 nodes for a large problem. In the presentation, some factors that affect the performance will be discussed.**

Keywords-component; PDPTW; CUDA; GPU Cluster; Parallel Computing;

I. INTRODUCTION

For a modern industry one of the most important task is the planning and management of a logistics system. A good planning can lead to a substantial cost reduction for the operation. Thus, digital computer has been used extensively to analyze and plan the transportation routes for vehicles. However, for a complex logistics problem, which includes many drop points, complex road network, time windows requirement can be a challenging problem. This kind of problem usually consumes a very high computing power due to the complexity of the problem and the algorithm that can solve this problem. One of the problems in this class is the PDPTW (Pickup and Delivery Problem with Time Windows) [1] problem. The objective of PDPTW is to minimize the number of vehicles that can serve all transportation requests and still observe the time windows constraint, the goods quantity upload to vehicles, and the vehicle capacity. The PDPTW problem is very compute intensive for a large problem size.

In this paper, the application of parallel computing on Graphics Processing Unit (GPU) [2] to PDPTW problem has been proposed. The parallelization technique for this problem has been proposed. In addition, the method proposed has been extended to work on a GPU cluster system. The experiments show that a speed up of nearly 43 times can be achieved using 4 GPU cluster nodes for a large PDPTW problem.

This paper is organized as follows. First, Section II gives the explanation of the theory and related work. Then, Section III proposes the implementation on GPU Cluster. Section VI gives the experimental results. Finally, Section V concludes the work and proposes the future works.

II. THEORY AND RELATED WORK

The PDPTW model is a large logistics problem that solves the problem when a fleet of vehicles must service all transport requests that specifies the load size, service time, and time window. The vehicle will transport goods from the pickup node to the delivery node respecting the capacity and time constraints. The PDPTW goal is to minimize the number of vehicle needed which can substantially reduce the logistics cost [3].

The parameters used to solve PDPTW problem are;

$N = \{0,1,...,n\}$: Set of nodes, where 0 is the depot and $\{1,...,n\}$ are the customers.

$\frac{n}{2}$: Size of pickup and delivery nodes.

$q[i], i \in N$: Size of load of node i request.
(if $q[i] \rangle 0$, the node i is a pickup)
(if $q[i] \langle 0$, the node i is a delivery)
(if $q[i] = 0$, the node i is a depot)

$t[i][j], \quad i, j \in N$: Traveling time between node i and j .

$e[i], i \in N$: Lower limit time window of node i .

$l[i], i \in N$: Upper limit time window of node i .

$p[i], i \in N$: Pickup at node i .

$d[i], i \in N$: Delivery at node i .

$s[i], i \in N$: Service time at node i .

$M = \{1,2,...,m\}$: Set of available vehicle.

C : Vehicle Capacity.

Assumptions

1. *All vehicles have the same capacity* C .
2. *Each vehicle will depart from and return to the depot* 0 .
3. *Each vehicle leaves and arrives at the depot exactly once.*
4. *When vehicle depart of the depot, must travel to pickup node first.*
5. *Each customer should be served exactly once.*
6. *If the vehicle arrives to customer before lower time window* $e[i]$ *, the vehicle has to wait.*

7. *The service time at each customer cannot overtake the upper time*

This problem is very compute intensive in its nature. Many efforts have been known on solving this using many platforms. Recently, the use of GPU has been emerged as a cost effective solution for solving last scale problem. Moreover, many large cluster such as TSUBAME in Japan has been equipped with GPU and show that a very powerful computing system wit cost effective and energy efficient can be built.

The GPGPU Technology [1] is an emerging technology that based on employing a fine grain parallelism on commodity graphic processors. The GPU's core consists of many stream processors and performs a computation in SIMD (Single Instruction Multiple Data) style. The architecture of GPU shown in Fig 1. The processing on GPU is organized into a Grid. A Grid consists of an array of blocks. Each block consists of many thread processors.

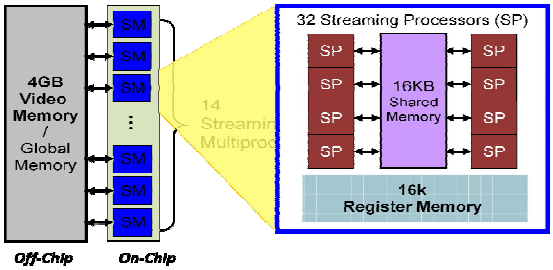

Fig. 1. GPU Architecture [4] (Courtesy of NVIDIA Inc.)

The GPU memory hierarchy [4] consists of registers, local memory, shared memory, global memory, constant memory and texture memory as shown in Fig 2. Each class of memory has a different data access time. Thus, a careful consideration must be taken in utilizing this properly. The shared memory access time is lower than the global memory access time but limited for the thread processors in the same block.

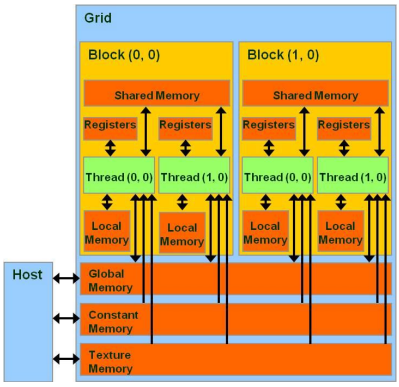

Fig. 2. Memory Organization on NVIDIA Graphics Processing Units [4] (Courtesy of NVIDIA Inc.).

CUDA (Compute Unified Device Architecture) [5] is a general purpose parallel computing architecture on GPU from NVIDIA. The CUDA compiler is called NVCC [6], which generates both instructions for CPU and GPU.

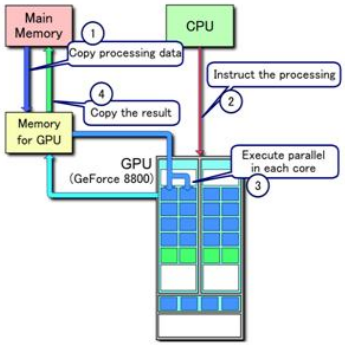

Fig. 3. Processing flow on CUDA [6].

Fig. 3 illustrates the step in using GPU. First, the input data from main memory is copied to global memory on GPU through PCI Express bus. Then, the computing on GPU can start in parallel on each thread. The parallelism can be specified using modified C language primitive. Finally, the output data are copied back to main memory.

III. Implementation Details

In this paper, the *Insertion Heuristic algorithm* [8][9][10] for solving PDPTW is addressed. The steps algorithm performed to solve PDPTW is shown below:

Algorithm 1: solving PDPTW using Insertion Heuristic algorithm

1. *Let L be the list of customers.*
2. *Let A be the list of pickup nodes.*
3. *Sorting A in an array by lower limit time window from least to greatest.*
4. *Let all vehicles have empty routes.*
5. *Let $R[k][\ ]$ be the list of a pickup and delivery route for vehicle k; $1 \le k \le m$.*
6. *Select member in A. Given $r1$ is $\{A[a] \mid 1 \le a \le \frac{n}{2}\}$, if $r1$ and delivery node of $r1\left(d[r1]\right)$ are member of L.*
7. *Take $r1$ and $d[r1]$ into $R[k][\ ]$. Remove $r1$ and $d[r1]$ from L.*
8. *Select member in A. Given $r2$ is $\{A[q] \mid a \le q \le \frac{n}{2}\}$, if $r2$ and $d[r2]$ are member of L. Inserting $r2$ and $d[r2]$ into route of vehicle k at a feasible position where the route is minimized of time spent.*
9. *Check route. If the capacity of vehicle k cannot be overload, $r2$ and $d[r2]$ are removed from L and go to (8.) again.*
10. *If L is not empty, go to (5.) for searching new route for vehicle $k+1$.*
11. *Improving the initial solution base on Tabu Search.*

The implementation of this algorithm on a GPU system is as follow. First, the problem can be distributed to multiple nodes using MPI. Since there is very minimal communication among tasks, this will allows us to utilize more GPU on cluster nodes and achieve a much higher scalability for a large scale problem. In this work, the execution of parallel code on GPU cluster is started as follows. First, master task read the configuration of problem and distributes it to each node. The, each node computes the sub-tasks assign to it using GPU. Finally, the best solution on each node is returned to master task, which, in turn, select the best solution obtained, and report to user. This concept is illustrated in Fig. 4.

In order to determine the workload on each node, number of pickup nodes must be consider. First, time windows must sort the pickup node list. Then, the number of feasible solution or workload for each node can be calculated as shown in Equation (1).

$$W = \left\lceil \frac{\frac{n}{2} \times (\frac{n}{2} - 1)}{2} \right\rceil / S \qquad (1)$$

where,

W is the workload of each node

n is a number of customers.

S is number of compute node in cluster.

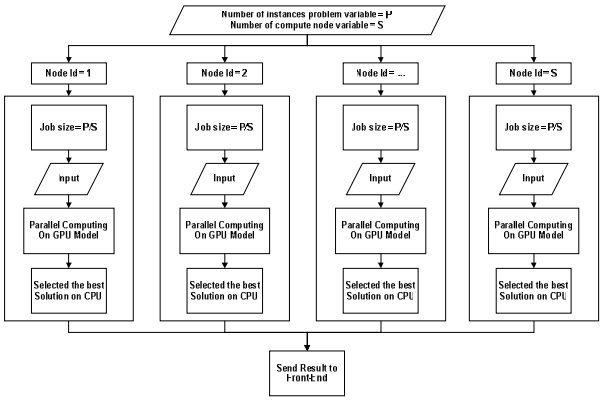

Fig. 4. Work distribution on a GPU Cluster.

In order to utilize the GPU on each node, the problem must be break down onto block and thread. The technique is to assign each thread on GPU to computes one instance of problem. This can be done by allowing user to define number of thread per block. Then, the number of block needed for the grid can be computed by dividing number of task W with number of thread per block T. Hence, the kernel function of this problem will be invoked using W and T as number of block and number of thread. This will enable users to adjust number of thread based on the GPU architecture being utilized at that time. After every thread finishes the computing, the result is copied to main memory for further processing.

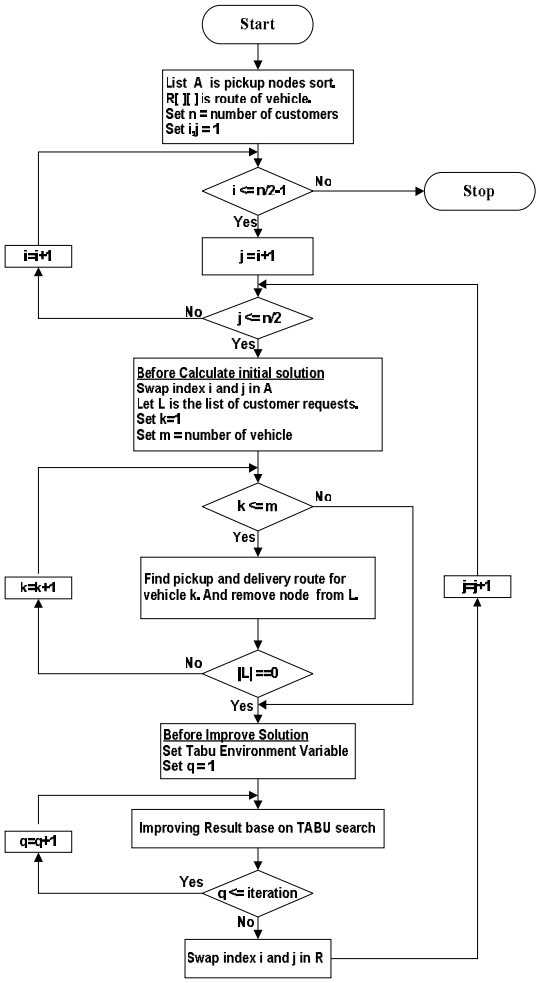

Fig. 5. Algorithm used in solving PDPTW problem.

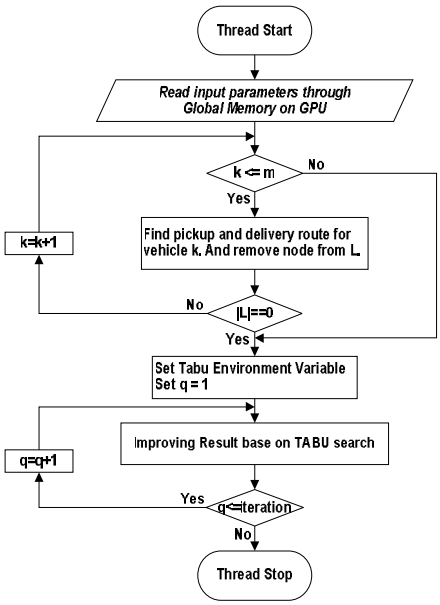

Fig. 6. Thread processing flow on GPU.

IV. EXPERIMENTS AND RESULTS

In this work, the experiments have been conducted on an 8 nodes GPU cluster in Kasetsart University. Each node is equipped with Intel Xeon Quad core 2.4GHz and Nvidia Tesla M2050 (448 cores), 1.15GHz and 2Gbytes of GPU memory. In this work, only 4 nodes is used.

In this work, we consider 3 factors that affect the performance of GPU cluster processing namely, the number of the compute node, number of thread processors within block on GPU, and memory access strategy. Table I shows the execution time of a sequential program on the CPU while Table II to V show the execution results on the GPU cluster.

TABLE I
EXECUTION TIME (IN SECONDS) ON SEQUENTAIL COMPUTE

Problem Sizes (locations)	Execution time(Seconds)			Total Time (Sec)
	Read Input	*Compute*	*Write Output*	
106	0.0011	762.00	0.0011	762.0030
212	0.0014	16645.40	0.0014	16645.4029
422	0.0017	263030.00	0.0017	263030.0000

TABLE II
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 1 COMPUTE NODE

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	142.04	150.17	135.21
212	1380.60	1517.01	1731.14
422	21314.93	38775.73	37867.82

TABLE III
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 2 COMPUTE NODE

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	135.95	139.14	135.67
212	838.64	853.08	862.96
422	12061.03	18637.36	18629.25

TABLE IV
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 3 COMPUTE NODE

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	128.71	126.90	135.99
212	792.53	819.77	854.73
422	8417.74	13810.23	13932.70

TABLE V
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 4 COMPUTE NODE

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	124.66	127.48	136.08
212	767.84	768.10	854.67
422	6062.87	8207.91	8557.12

Table VI-IX show the execution time on a GPU cluster where each thread processors can access both global memory and shared memory on GPU.

TABLE VI
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 1 COMPUTE NODE WHICH EACH THREAD ACCESSING SHRAED MEMORY ON GPU

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	171.75	115.26	121.48
212	2352.15	1459.71	2003.22
422	34302.41	19991.13	32893.87

TABLE VII
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 2 COMPUTE NODE WHICH EACH THREAD ACCESSING SHRAED MEMORY ON GPU

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	110.94	110.98	113.90
212	1453.43	987.64	649.64
422	18252.56	11041.95	16972.06

TABLE VIII
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 3 COMPUTE NODE WHICH EACH THREAD ACCESSING SHRAED MEMORY ON GPU

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	110.09	110.53	113.66
212	1031.31	625.46	643.47
422	12300.72	6850.99	12094.65

TABLE IX
EXECUTION TIME (IN SECONDS) USING 32, 64 AND 128 THREAD ON 4 COMPUTE NODE WHICH EACH THREAD ACCESSING SHRAED MEMORY ON GPU

Problem Sizes (locations)	Number of threads within block on GPU		
	32	*64*	*128*
106	110.47	110.75	113.88
212	967.74	598.62	636.97
422	10084.05	6491.41	9886.76

The speedup for this experiment is shown both in Fig. 7, Fig. 8 and Fig. 9.

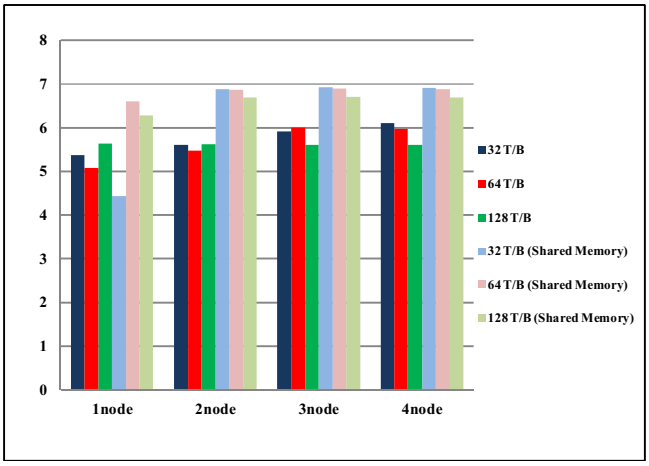

Fig. 7. Speedup for the problem that includes 106 sites.

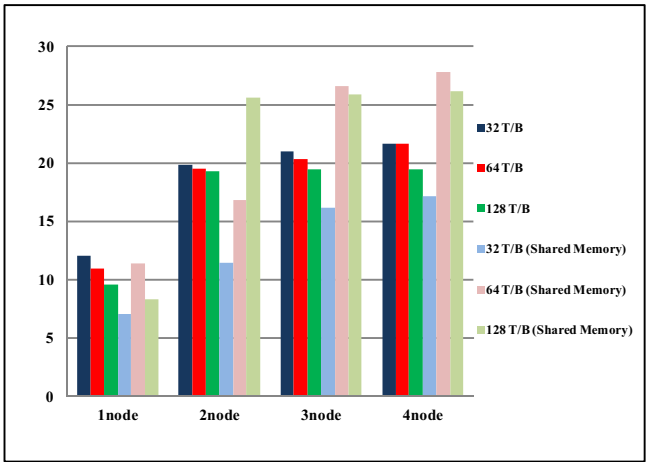

Fig. 8. Speedup for the problem that includes 212 sites.

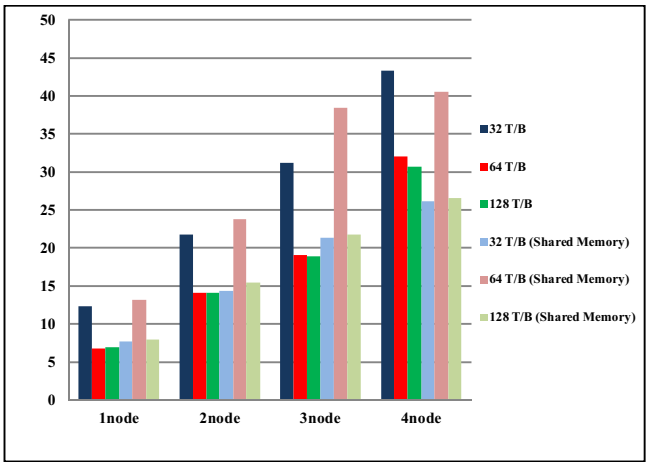

Fig. 9. Speedup fpr the problem that includes 422 sites.

A few facts can be observed from the experiments. First, a large problem, speed up increases as the number of compute nodes is increased. This shown that the problem can scale on a cluster. This is due to the low communication overhead and the highly parallel nature of this algorithm. Then, we can clearly observe that the change in number of threads per block and the use of shared memory will have some impact to the performance of the execution. It can be seen that the change in number of threads can substantially increases the performance. Anyway, an appropriated number of threads must be used to gain the maximum performance. Thus, a careful consideration must be done on adjusting the number of threads to match the GPU configuration used. In addition, the use of shared memory do help further increase the speedup but the impact will depend on the configuration decision made earlier on selection the number of threads.

V. Conclution and Future Work

In this paper, we propose the use of parallel computing on GPU cluster to accelerate the speed of solving a pickup and delivery problem with time windows (PDPTW). The experimental results demonstrate that a significant speed up can be attained. However, choice of the number of compute node, the partition of threads on GPU, and the use of shared memory on GPU will have an effect on the performance. It is important to conduct small experiments to tune up these values. This work can be extended by implementing the program on a GPU cluster with multiple GPU boards. In addition the perform prediction of speed up can be done to adapt the algorithm to better optimize the solution for PDPTW problem.

References

[1] M.W.P. Savelsbergh and M. Sol, “The General Pickup and Delivery Problem”, Transporation Science, 29(1), pp. 17-29, 1995.

[2] M. A. Gray, "Getting Started with GPU Programming", Computing in Science & Engineering, vol. 11, pp. 61-64, 2009.

[3] R. Kammarti, S. Hammadi, P. Borne and M.Ksouri, “Improved Tabu Search In An Hybrid Evolutionary Approach For The Pickup and Delivery Problem With Time Windows”, in Proceedings of the international IEEE Conferrence on Intelligent Transporation Systems, Austria, September, 2005.

[4] Johan Seland , “CUDA Programming”, Geilo Winter School, 2007.

[5] W. Enhua and L. Youquan, "Emerging technology about GPU," in Circuits and Systems, IEEE Asia Pacific Conference, pp. 618-622, 2008.

[6] [Online]. Available: http://en.wikipedia.org/wiki/CUDA.

[7] M. Pharr and R.Fernando, “ GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation”, Addison Wesley Professional, 2005.

[8] Quan Lu and Maged M. Dessouky, “A New Insertion-based Construction Heuristic for Solving the Pickup and Delivery Problems with Time Windows”, European Journal of Operational Research, vol. 175, pages 672-687, 2006

[9] Hoong Chuin LAU and Zhe LIANG, “Pickup and Delivery with Time Windows : Algorithms and Test Case Generation”, in Proceedings of the 13th International Conferrence on Tools with Artificial Intelligence, pp. 333-340, 2001.

[10] ANTOINE LANDRIEU, YAZID MATI and ZDENEK BINDER, “A tabu search heuristic for the single vehicle pickup and delivery problem with time windows”, Journal of Intelligent Manufacturing, vol. 12, pp. 497-508, 2001.