1st.International Conference of Information Technology to enhance E-learning and other Application,
(IT-ELA 2020), Baghdad College of Economic Sciences University, Baghdad, Iraq

A Parallel Implementation Method for Solving the Shortest Path Problem for Vehicular Networks

Salim A. Mohammed Ali
Department of Networks Engineering
Al-Nahrain University
Baghdad, Iraq
salimm@ieee.org

Emad H. Al-Hemiary
Department of Networks Engineering
Al-Nahrain University
Baghdad, Iraq
emad@coie-nahrain.edu.iq

***Abstract*—Vehicular networks are becoming an important area in research in many aspects. One aspect is the shortest path problem, since vehicles are mobile and condensed in behaviour, finding the shortest path for all vehicles to their destinations simultaneously and repeatedly is a cumbersome task. In this paper, a known algorithm is used for finding multi-source multi-destination shortest paths among all nodes, which is Hedetniemi's Algorithm. A recent parallel computation method of OpenCL (Open Compute Language) is used to reduce the complexity of the algorithm and achieve faster calculation for the paths simultaneously and repeatedly. Also, the paper investigates the parallelism efficiency and monitor the system performance when using high dimensional input data. A comparison is made through implementing the same algorithm using sequential computation versus parallel computation, and acceleration of 40X speedup was obtained for a significant number of nodes.**

Keywords—Hedetniemi, OpenCL, Shortest path, Vehicular networks.

I. INTRODUCTION

SP (Shortest path) problems are widespread in computer networks and transportation. In graph theory, it is defined as figuring out a path between two nodes, such that the total weights of all-surpassing nodes on the path is minimum [1]. There are three main approaches to implementing shortest path problems: single source, single destination, and all-pairs shortest path problems. Each one can be implemented depending on the purpose of the problem. The third type, all-pair shortest path problems, is implemented in many applications such as bioinformatics, network traffic, data management of social networking, and routing protocol problems [2]. This paper uses the third type because it suits vehicular networks such that finding the best routes for vehicles by using a dedicated server to manage all vehicles computations instead of each vehicle computing its route. Since IoT (Internet of Things) [3], [4] applications and services are becoming available on many devices even vehicles, vehicle GPS (Global Positioning System) information can be easily sent to the server via reliable means such as mobile networks [5]. When SP problems have a tremendous size of graphs, several vehicles in this scenario, the parallel computation can be ushered to solve such problems, especially when these sorts of problems have real-time settings. The selected algorithm is Hedetniemi matrix sum, which finds the all-pairs shortest path problem. Implementing Hedetniemi on sequential basis using contemporary computing devices will have tremendous computations, and it will produce high latency, which cannot be tolerated in vehicular networks or other similar applications. However, the structure of the algorithm mentioned above has inherent parallelism one can utilize.

Therefore, exploiting the embedded parallelism within the Hedetniemi algorithm will minimize the computation time and can be preferable upon other available algorithms such as Dijkstra, Bellman-Ford, and Floyd-Warshall [6]. The other available algorithm cannot be implemented efficiently using parallel computation due to the steps of the algorithm that are not parallel friendly. Several parallel implementations are implemented in the literature such as in [7], only 17% improvement was obtained using a parallel implementation of Dijkstra, and in [8], they obtained 6x speedup of parallel implementation of K-shortest path algorithm, which is still lower than what this research has achieved.

The goal of this paper is to implement the algorithm in OpenCL [9], which is a parallel programming framework that enables a computer system to use different devices for computation such as CPUs, GPUs, and other accelerator devices like FPGAs, and exploits parallel computation.

The structure of the paper is presented as follows; the second section enlightens the role of the shortest path in vehicular networks and reviews the cloud-based technologies used in this area, with a simple introduction of OpenCL computing framework. The following section explains Hedetniemi algorithm in details with a computational example. The third section shows the parallel implementation of the algorithm using OpenCL, then followed by results and discussion. The final section concludes the paper with the main outcomes.

II. LITERATURE REVIEW

A. Shortest Path Problems

Suppose a network of nodes, or weighted graph, which can be defined as, a group N of nodes, a group E of edges, a weighted-function of real values (F: E → ℝ), an element n of

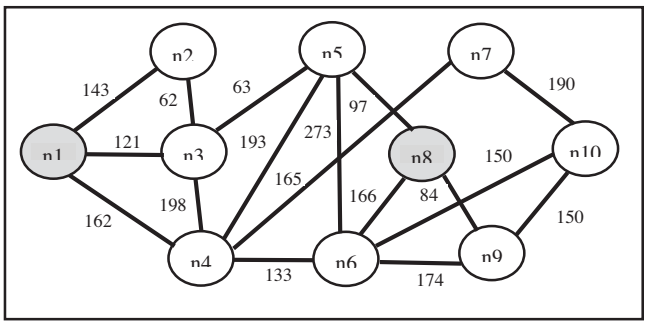


Fig. 1. Nodes with weighted paths

Authorized licensed use limited to: University of Roehampton. Downloaded on July 29,2024 at 15:49:42 UTC from IEEE Xplore. Restrictions apply.

N, and a route r from n to each n' of N, so that ∑f(r) is the minimum among all routes connecting n to n' [10]. As for the all-pairs SP problems, one has to find out the routes of each pair of nodes (n, n') in the network. To illustrate the case, Fig. 1 shows a set of nodes with connected paths as an example.

To find the shortest path between nodes 1 and 8, one can choose the minimum route from the set of all paths between the two nodes. The main point of this paper is to find the set of all minimum paths among all nodes; supposedly, these nodes are either vehicles' instant locations or their desired destinations. Plenty algorithms are available in the literature to solve shortest path problems, such as Dijkstra, Bellman-Ford, and Floyd-Warshall [6]. In this paper, Hedetniemi algorithm is chosen and implemented.

B. Cloud-based Vehicular Networks

A plethora of researches is conducted in vehicular networks, especially in VANETs (Vehicular Ad hoc Network) to address shortest path problems [11]. However, with the introduction of fast and reliable mobile networks and the proliferation of V2X (Vehicle to Everything) services [12], vehicles information and data processing is migrating to could services [13-14]. In this case, instead of using Ad hoc networks for solving shortest problems, it is possible to perform it on the cloud. Cloud computing has advantages and disadvantages. One essential benefit of using cloud computing in determining the best routes for vehicles is the use of machine learning to anticipate and analyze the best routes before routes become congested and blocked [15]. The main disadvantage of using cloud computing is the requirement of heavy computational servers that should achieve a tremendous number of calculations in low latency limits. For this reason, accelerating even little percentage of computing will save a great amount of cost and time.

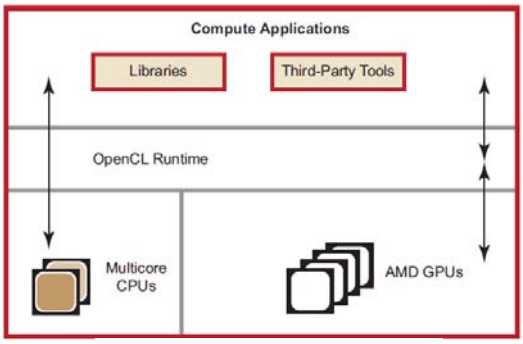
Most technologies used in today's applications, such as Google Maps, which assists vehicle drivers to choose the best route rather it is the shortest distance, least time, or the most economical route uses Dijkstra's algorithm [16]. The latter solves one to one shortest path problems efficiently and less costly than other SP algorithms. However, based on [7] implementing Dijkstra's algorithm in a parallel computational environment results of 17% improvement, which is why it is better to use other parallel-friendly algorithms when using parallel computation environments. For example, in the K-shortest path algorithm, 6X speedup is obtained when exploiting parallelism because the algorithm has parallel-friendly structure [8].

C. OpenCL Computational Framework

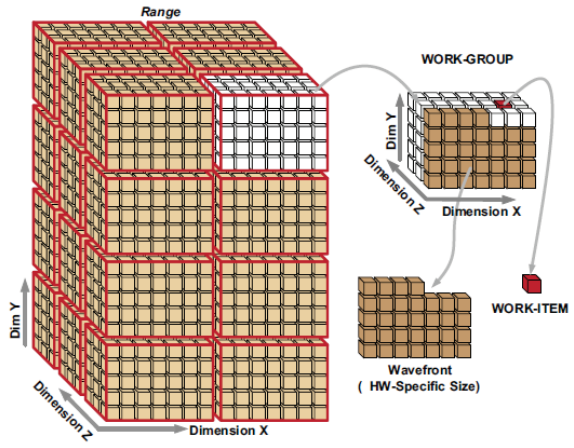
OpenCL, as defined in Fixstars Corporation OpenCL programming book [9], is "a framework suited for parallel programming of heterogeneous systems". The framework comprises the OpenCL directives, in addition to the compiler with the runtime environment that is required to run the code which can be written in either C or Fortran programming languages. OpenCL is supported by many hardware platforms including CPUs, GPUs and FPGAs.

Figure 2a shows the ecosystem of various layers in using OpenCL environment for AMD (Advanced Micro Devices) compute devices and Fig. 2b shows how work-items (processing units) are distributed to perform the same tasks for different inputs and run the mathematical computations simultaneously [17].

OpenCL maps work-items is the processing units that are in charge of executing kernel (the operations needed to be computed simultaneously), to be initiated onto an N-dimensional grid (ND-Range). The programmer is free in arranging these groupings. In AMD GPUs, it runs on wavefronts (collection of work-items that run simultaneously); each workgroup comprises of multi-wavefronts as shown in Fig. 2b, there is a middle step for preparation the work-items to execute in parallel by assigning the number of wavefronts in a single workgroup. This process entails to preferable configuration for obtaining highest parallelization [17].



(a)



(b)

Fig. 2. (a) Parallel processing software ecosystem, (b) Work-Item groupings in parallel hardware platforms.

III. HEDETNIEMI'S ALGORITHM

The third approach aims at finding all the shortest paths among all nodes or vertices simultaneously in a weighted-graph, this is useful in many aspects, but it requires many calculation steps. As shown in Fig. 1. Suppose there is a road network with 10 vehicles having all distances or weights defined instantly. The goal is to find the SP between any two vehicles at any given time. Hedetniemi's algorithm is one solution to this pragmatic problem, since the structure of the algorithm suites the problem and can be scaled to any dimension, when having a large number of vehicles.

The mathematical construction of the algorithm is based on computing power matrices, and the operation is called

Authorized licensed use limited to: University of Roehampton. Downloaded on July 29,2024 at 15:49:42 UTC from IEEE Xplore. Restrictions apply.

Hedetniemi matrix sum, which is represented by the symbol ǂ [10]. To explain the algorithm easily, suppose the weighted graph shown in Fig. 1, with nodes n_1, …., n_n, the graph is mapped to an n×n "adjacency matrix" A = [a_{ij}], represented by [10]:

$$aij = \begin{cases} 0, & if\ i = j \\ x, & if\ i \neq j\ with\ an\ edge\ of\ weight\ x\ between\ i\ and\ j \\ \infty, & otherwise. \end{cases} \quad (1)$$

where i and j represent matrix dimension, x represents the weight between nodes i and j, ∞ represents the unknown path or route total weights between nodes i and j and will be computed in the next matrix power. In a computer program, large numbers are used instead of infinity to represent the temporarily unknown weight. The graph of Fig. 1 has the following adjacency matrix A as shown in table I.

TABLE I. THE FIRST POWER ADJACENCY MATRIX OF THE GRAPH IN FIG. 1

A^1	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10
n1	0	143	121	162	∞	∞	∞	∞	∞	∞
n2	143	0	62	∞	∞	∞	∞	∞	∞	∞
n3	121	62	0	198	63	∞	∞	∞	∞	∞
n4	162	∞	198	0	139	133	165	∞	∞	∞
n5	∞	∞	63	139	0	273	∞	97	∞	∞
n6	∞	∞	∞	133	273	0	155	166	174	150
n7	∞	∞	∞	165	∞	155	0	∞	∞	190
n8	∞	∞	∞	∞	97	166	∞	0	84	∞
n9	∞	∞	∞	∞	∞	174	∞	84	0	150
n10	∞	∞	∞	∞	∞	150	190	∞	150	0

Hedetniemi matrix sum, denoted by ǂ, can be explained through the following mathematical definition and two theorems [18]:

Definition 1 "Let A be an m×n matrix and B an n×p matrix. Then the Hedetniemi matrix sum is the m× p matrix C = A ǂ B, whose (i, j)th entry is":

$$Cij = \min(a_{i1} + b_{1j},\ a_{i2} + b_{2j}, \ldots\ldots, a_{1n} + b_{nj}) \quad (2)$$

Theorem 1 "In a connected, weighted graph with n vertices, the (i, j)th entry of the Hedetniemi matrix A^{n-1} is the length of the shortest path between v_i and v_j".

Theorem 2 "For a connected, weighted graph with n vertices, if the Hedetniemi matrix $A^k \neq A^{k-1}$, but $A^k = A^{k+1}$, then A^k represents the set of lengths of shortest paths, and no shortest path contains more than k edges".

Therefore, if the algorithm is applied to the above adjacency matrix, it will end up with the following final adjacency matrix, A^4 is shown in table II. So that, the SP between each pair of nodes, or vehicles, will be directly identified through intersecting the source node column with destination node row.

However, this algorithm can be stopped just after computing a few matrix powers, depending on the connectivity among the nodes. In this case, matrix A^5 is identical to matrix A^4, and the computation is stopped.

TABLE II. THE FOURTH POWER ADJACENCY MATRIX

A4	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10
n1	0	143	121	162	184	295	327	281	469	445
n2	143	0	62	260	125	295	425	222	306	445
n3	121	62	0	198	63	331	336	170	254	481
n4	162	260	198	0	139	133	165	236	307	283
n5	184	125	63	139	0	273	304	97	181	423
n6	295	295	331	133	273	0	155	166	174	150
n7	327	425	336	165	304	155	0	321	340	190
n8	281	222	170	236	97	166	321	0	84	239
n9	469	306	254	307	181	174	340	84	0	150
n10	445	445	481	283	423	150	190	239	150	0

To remove the computational overhead of the last matrix, one can check for no more ∞ item is existed in the last matrix, which means no more unknown shortest paths are existed among all nodes. This process was actually implemented in the code.

Figure 3 shows the actual pseudo-code for Hedetniemi adjacency matrix power computation [10]. The halting condition is valid when the values of the successive matrix powers are matching.

```
procedure Hedetniemi(G: weighted simple graph)
{G has vertices v1, ...., vn and weights w(vi, vj) with
w(vi,vi) = 0 and w(vi, vj) = ∞ if (vi, vj) is not an edge}
for i := 1 to n
  for j := 1 to n
     A(1,i,j) := w(vi, vj)
t := 1
repeat
  flag := true
  t := t + 1
  for i:= 1 to n
     for j := 1 to n
          A(t,i,j) := A(t-1,i,j)
          for k:=1 to n
             A(t,i,j) := min{A(t,i,j), A(t-1,i,j)+A(1,k,j)}
           if A(t,i,j) ≠ A(t-1,i,j) then flag := false
until t = n-1 or flag = ture
{A(t,i,j) is the length of the shortest path between vi and
vj }
```

Fig. 3. Hedetniemi shortest path algorithm [10].

IV. OPENCL IMPLEMENTATION

As can be seen above in Fig. 3, the algorithm pseudo-code, the computation will take a non-negligible amount of time for determining each adjacency matrix power, since it has four loops inside each other. In order to overcome this incompetence, the most inner loop is implemented as a kernel for the OpenCL program. Furthermore, the row-column Hedetniemi matrix sum will be executed concurrently, so this will give us ample acceleration, especially when the size of the matrix is large. In other words, the size of the matrix would not effect on the execution time unless it exceeds the computing device boundary.

OpenCL kernel design, associated with the algorithm, is the main component of parallelism that can be accomplished

Authorized licensed use limited to: University of Roehampton. Downloaded on July 29,2024 at 15:49:42 UTC from IEEE Xplore. Restrictions apply.

by the accelerating devices, for example, Multi-Core CPUs or GPUs. Therefore, it is sufficient to present the kernel for 16X16 grid of computing units, and higher levels of computation follow the same rule. Fig. 4 shows the lines of code that represents the 16X16 kernel for the OpenCL framework.

The kernel is specifically written to comply with the algorithm to reduce its complexity. More specifically, the most inner two loops shown in the algorithm is replaced by the (globalidx & globalidy), which are the wavefront dimensions illustrated in Fig. 2b. The job of OpenCL Framework is to map each computation to a specific work item, such that all work items process concurrently and the results of the computations are forwarded to the higher-level host computing device, usually the main CPU. However, as shown in Fig.4, the block size has been assigned equals to 16, which means the dimension of the wavefront is 16X16. This number can be changed according to the specifications of the parallel computing device.

```
#define BLOCKSIZE 16

__kernel void hedetneimiMatrix(__global flaot *mO,
        __global float *mA,
                uint widthA)
{
        int globalidx = get_global_id(0);
        int globalidy = get_global_id(1);

        float sum = 0;
        float temp = 999;  //this represent infinity
        int xz = 0;

        for (int i=0; i<widthA; i++)
                {
                float tempA = mA[globalidy *
                widthA + i];

                float tempB = mA[globalidy * widthA
                + i];
                sum = (tempA + tempB);
                if (temp>sum) temp=sum;
                }
        mA[global * widthA + globalidx]=temp;
        mO[globalidy * widthA + globalidx] temp;
}
```

Fig. 4. OpenCL kernel for Hedetniemi algorithm.

The width and the height of the adjacency matrix are the same since it is symmetrical, only two buffers are sent to the kernel: the matrix as an array, and its dimension width. The rest of the OpenCL code includes the environment variables definition and the preparation of the buffers, which will then distribute the computation effort among the work-items. Fig. 5 shows a sample of 16 nodes implementation showing both the first adjacency matrix and the final one using a commodity laptop with basic CPU and GPU.

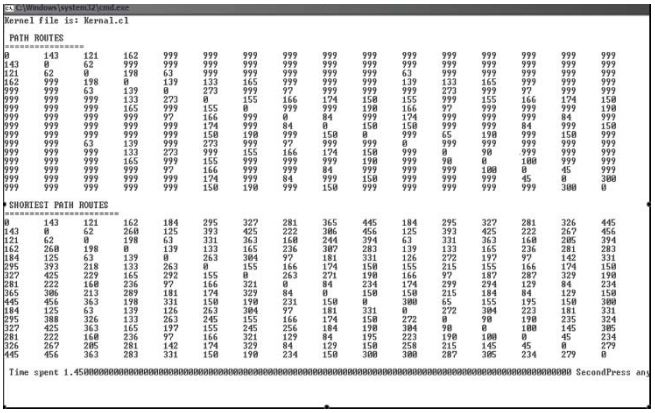


Fig. 5. A sample implementation of 16 nodes matrix.

Many adjacency matrices were generated using *Matlab* [19], ranging from (16 to 4096) nodes. The algorithm was implemented on each matrix using both CPU and GPU individually. Results of tests are discussed in the next section.

V. RESULT AND DISCUSSION

The following table shows the computation time, including memory read [16], of each matrix for the CPU (original implementation) and GPU (using the new method) respectively ranging from 16 to 4096 nodes:

TABLE III. COMPUTATION TIME RESULTS OF ALGORITHM IMPLEMENTATION ON CPU AND GPU.

# of Vertices	CPU (sec)	GPU (sec)
16	1.263	1.029
32	1.279	1.107
64	1.264	1.045
128	1.263	1.092
256	1.295	1.092
512	1.575	1.123
1024	13.136	1.264
2048	97.797	3.807
4096	1000.601	23.619

Authorized licensed use limited to: University of Roehampton. Downloaded on July 29,2024 at 15:49:42 UTC from IEEE Xplore. Restrictions apply.

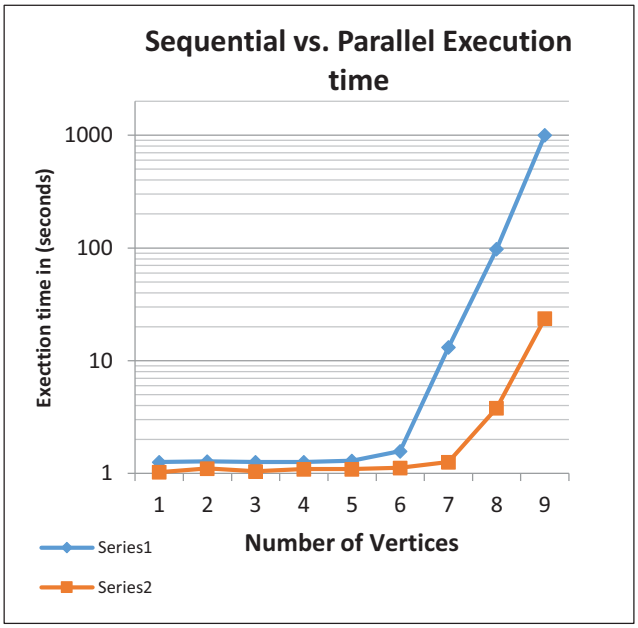


Fig. 6. Execution time difference between the sequential and parallel implementation

Figure 6 shows a logarithmic graph with two curves based on the data in Table I.

As can be seen from the results the parallel implementation of the algorithm has greatly reduced the computational time especially with the increment of several nodes or vertices, which in this case, the number of vehicles in the vehicular networks. The implementation is not only limited to vehicular networks but also it can be used on many other applications and routing algorithms such as Mobile Ad hoc networks [20].

On the other hand, there are some limitations regarding OpenCL; one is the number of workgroups and items, which it had to be multiple of the block size used in the framework. This constraint would restrict the problem variables to certain ranges. The other limitation that can be observed from OpenCL is the buffer or memory transfer overhead. The time needed to transfer large data from the host to the device (GPU) is not negligible that sometimes exceeds the time needed for the host device to complete a task sequentially. In terms of the algorithm computation process, there was redundant computing regarding the adjacency matrix. The upper and lower triangular matrix has the same elements, with the main diagonal elements of zeros. So, practically if this algorithm is implemented sequentially or normal mode, there will be more than 50% less computation.

VI. Conclusions

As a conclusion, the implementation of the algorithm is successful, and the obtained results from the sequential implementation and the proposed method are identical. The performance of the new method with a large number of nodes is so effective. Although the redundancy computation seems wasteful, OpenCL implementation improves the performance by n-times, depending on the computational distribution and the computing device capability. Therefore, comparing the factor of n-times with the wastefulness of 2-times seems more practical and efficient. Therefore, it can be concluded that to get good results, and the input data should be large enough that takes the original implementation ample time to compute and be specific to certain design limits for accomplishing maximum efficiency. Comparing to the parallel implementations of other SP algorithms such as Dijkstra (17% speedup) and K-shortest path (6X speedup), which are used in most routing applications, Hedetniemi's algorithm parallel method of computation has achieved 40x speedup by using commodity hardware computing device.

References

[1] A.J. Bondy and U.S.R. Murty. *Graph Theory with Applications*. Ontario, Canada: Wiley, 1991, pp. 1-2

[2] G.J. Katz and J.T. Kider. "All-pairs shortest-paths for large graphs on the GPU," in *Proc. EGGH-EGGH*, 2008, 47-55.

[3] I. M. Al-Joboury and E. H. Al-Hemiary, "Internet of Things (IoT): Readme," in *Qalaai Zanist Scientific Journal*, vol. 2, no. 2, pp. 343–358, 2017.

[4] I.M. Al-Joboury and E. H. Al-Hemiary, "Internet of things Architecture Based Cloud for Healthcare," in *Iraqi Journal of Information & Communications Technology*, vol. 1, no.1, pp.18-26, 2018.

[5] A. Mustafa, M. I. A al-Nouman and O. A. Awad, "A Smart real-time tracking system using GSM/GPRS technologies," *2019 First International Conference of Computer and Applied Sciences (CAS)*, Baghdad, Iraq, 2019, pp. 169-174.

[6] A.A. Tamimi. "Comparison Studies for Different Shortest path Algorithms." *International Journal Of Computers & Technology*, vol. 14 no. 8, p.p 5979-5986, 2015.

[7] H. Ortega-Arranz, Y. Torres, D. R. Llanos and A. Gonzalez-Escribano, "A new GPU-based approach to the Shortest Path Problem," *2013 International Conference on High-Performance Computing & Simulation (HPCS)*, Helsinki, 2013, pp. 505-511.

[8] A.P. Singh and D.P. Singh, "Implementation of K-shortest Path Algorithm in GPU Using CUDA," *2015 International Conference on Computer, Communication and Convergence (ICCC 2015), 2015, pp. 5-13.*

[9] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara, J. Son and S. Miki. *The OpenCL Programming Book. Fixstars, 2012, pp. 17-32.*

[10] S. Arlinghaus, W. Arlinghaus, and J. Nystuen, "The Hedetniemi Matrix Sum: An Algorithm for Shortest Path and Shortest Distance," in *Geographical Analysis*, vol. 22, No. 4, Ohio State University Press, 1990, pp. 351–360.

[11] C. P. Dubey, V. Kumar, B. Sharma and G. Kaur, "Shortest Path Algorithm for Distributed VANET using Grid Computing," 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2018, pp. 118-121.

[12] S.A. Mohammed Ali and E. H. Al-Hemairy, " Minimizing E2E Delay in V2X over Cellular Networks: Review and Challenges," in *Iraqi Journal of Information & Communications Technology*, vol. 2, no.4, pp. 31-42, 2019.

[13] A. Mustafa, M. I. A al-Nouman and O. A. Awad, " Cloud-Based Vehicle Tracking System," in *Iraqi Journal of Information & Communications Technology*, vol. 2, no.4, pp. 21-30, 2019.

[14] D. A. Bahr and O. A. Awad, " LTE Based Vehicle Tracking and Anti-Theft System Using Raspberry Pi Microcontroller," in *Iraqi Journal of Information & Communications Technology*, vol. 2, no.1, pp. 10-25, 2019.

[15] L. Liang, H. Ye and G. Y. Li, "Toward Intelligent Vehicular Networks: A Machine Learning Framework," in *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 124-135, Feb. 2019.

[16] D. R. Lanning, G. K. Harrell, and J. Wang, "Dijkstra's Algorithm and Google Maps,", in *Proceedings of the 2014 ACM Southeast Regional Conference (ACM SE'14),* New York, NY, United States, March 2014, No. 30, pp. 1-3.

[17] S. A. M. Ali, A. M. Al-Kadhimi and S. Hasan, "Investigating the Parallel Components of TLD Algorithm Using OpenCL Computation Framework," *2018 10th Computer Science and Electronic Engineering (CEEC)*, Colchester, United Kingdom, 2018, pp. 36-40.

Authorized licensed use limited to: University of Roehampton. Downloaded on July 29,2024 at 15:49:42 UTC from IEEE Xplore. Restrictions apply.

[18] S. Arlinghaus, W. Arlinghaus, and J. Nystuen. "The Hedetniemi matrix sum: A real-world application." *An Electronic Journal of Geography and Mathematics*, vol. I, no. 2, 1990.

[19] MATLAB and Statistics Toolbox Release 2012b, The MathWorks, Inc., Natick, Massachusetts, United States.

[20] M. Abdulkadhim, S. A. M. Ali and S. Hasan, "A Reliability Improved Routing as a Qos Measure for Mobile Ad-Hoc Networks." in *International Journal of Engineering & Technology*, vol. 7, pp.201-204, Aug. 2018.

Authorized licensed use limited to: University of Roehampton. Downloaded on July 29,2024 at 15:49:42 UTC from IEEE Xplore. Restrictions apply.