A New Parallel Matrix Multiplication Algorithm on Tree-Hypercube Network using IMAN1 Supercomputer

Orieb AbuAlghanam, Mohammad Qatawneh
Computer Science Department
University of Jordan
Amman-Jordan

Hussein A. al Ofeishat
Computer Science Department
Al-Balqa applied university Jordan
Amman-Jordan

Omar Adwan, Ammar Huneiti
Computer Information Systems
University of Jordan
Amman-Jordan

***Abstract*—The tree-hypercube (TH) interconnection network is relatively a new interconnection network, which is constructed from tree and hypercube topologies. TH is developed to support parallel algorithms for solving computation and communication intensive problems. In this paper, we propose a new parallel multiplication algorithm on TH network to present broadcast communication operation for TH using store-and-forward technique, namely, one-to-all broadcast operation which allows a message to be transmitted through the shortest path from the source node to all other nodes. The proposed algorithm is implemented and evaluated in terms of running time, efficiency and speedup with different data size using IMAN1. The experimental results show that the runtime, efficiency and the speedup of the proposed algorithm decrease as a number of processors increases for all cases of matrices size of 1000×1000, 2000×2000, and 4000×4000.**

Keywords—MPI; supercomputer; tree-hypercube; matrix multiplication

I. INTRODUCTION

Parallel matrix multiplication is considered as a backbone for several scientific applications. Many studies developed matrix multiplication from different perspective in [3], [6], [7], [10], [14] the authors studied the limitation bandwidth for memory-processor through communication and how to reduce the gap between memory and processor speed, they present a communication efficient mapping of a large-scale matrix multiplication algorithm. In [1] parallel matrix multiplication algorithm on hypercube multiprocessors, while in [2], [5], [6] the authors applied a matrix multiplication with a hypercube algorithm on multi-core processor cluster. In [3] the author proposed a Distribution-Independent Matrix Multiplication Algorithm called DIMMA which is new, fast, and scalable matrix multiplication algorithm, for block cyclic data distribution on distributed-memory concurrent computers. In this paper we apply matrix multiplication on tree-hypercube which was used before in adaptive fault tolerate in routing algorithm [8], [11], [12], [17].

Many proposals for matrix multiplication algorithms were done on different networks type whether it was homogeneous or heterogeneous in order to reduce the time drastically to improve the system performance. Matrix multiplication needs high computation time especially when the size of the matrix becomes huge, therefore some problems need years to be solved using a personal computer. The interconnection networks are the core of a parallel processing system which the system's processors are linked. Due to the big role played by the networks topology to improve the parallel system's performance, several interconnection network topologies have been proposed for that purpose; such as the tree, hypercube, mesh, ring, and Hex-Cell (HC) [12], [13], [16].

Among the wide variety of interconnection networks structures proposed for parallel computing systems is Tree-Hypercube network which received much attention due to the attractive properties inherited in their topology [4], [8], [9], [15].

This paper aimed to design and analyze efficient matrix multiplication algorithm on tree-hypercube network. Experimentation of the proposed algorithm was conducted using IMAN1 supercomputer which is Jordan's first supercomputer. The IMAN1 is available for use by academia and industry in Jordan and the region.

The rest of the paper is organized as follows. Section 2 summarizes the definition of Tree-Hypercube network. In Section 3 the proposed algorithm is explained, Section 4 is an overview of sequential and parallel matrix multiplication is discussed. In Section 5, the evaluation results are illustrated. Finally, section 6 is the conclusion for the paper.

II. DEFINITION OF TREE-HYPERCUBE NETWORK

Tree-hypercube can be defined as follows: each node is labelled with the level number (L) and the node number (X) with the form (L,X). It is mainly a tree which represents a hypercube in each level because each level i in tree has s(i) nodes that represents the number of processing element as a hypercube [4], [9]. For very level L between 0 and d, each node (L,X) in level L, where X= X(d logs)-1……X0 is adjacent to the following s children nodes at level L+1 (L+1, X.a) for a =0,…..,s-1 as shown in Fig. 1(a) and (b) [8].

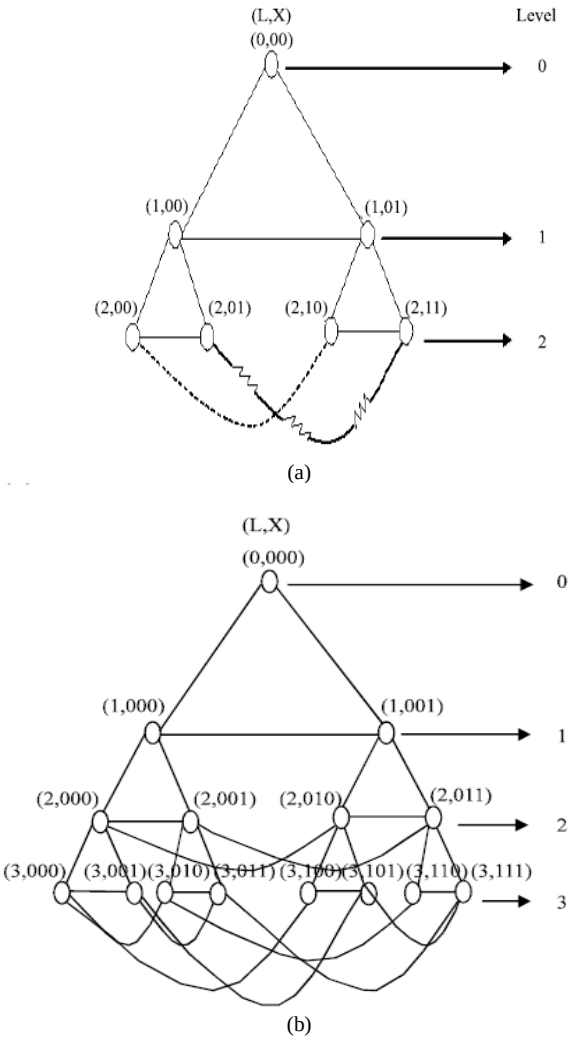

Fig. 1. Tree-hypercube (a) TH (2,2) and (b) TH (2,3).

III. Proposed Algorithm

Matrix multiplication is a simple and widely used algorithm in many scientific applications. Matrix multiplication is well structured in the sense that elements of the matrices can be evenly distributed to the nodes and communication among nodes which have regular pattern. Therefore, exploiting data parallelism based on message passing is suitable for solving the matrix multiplication problem. Fig. 2 shows a sequential algorithm of an n x n matrix multiplication consisting of three nested loops. As can be seen, the complicity of the algorithm is n3.

```
Matrix-Multiplication(A,B)
n =A.rows
Let C be a new n×n matrix
For i=1 to n
For j=1 to n
Cij=0
For k=1 to n
Cij=Cij + aik bkj
```

Fig. 2. A Sequential Multiplication Algorithm for two matrices [18].

In this section, we propose a new Parallel Matrix Multiplication Algorithm on Tree-Hypercube Network Using IMAN1 Supercomputer. The data distribution for matrix multiplication can be divided either striped partitioning or checkerboard partitioning. We use a block striped partitioning distribution to study the performance of message passing execution model. The column wise block striping of an (n x n) matrix on p processors (P0, P1, …, Pp-1) processor Pi contains columns: (n/p) (i), ((n/p) i)+1, …, ((n/p) (i+1)) – 1 [5].The proposed parallel matrix multiplication algorithm on Tree-Hypercube network consists of three stages as shown in Fig. 3. The first stage shows the distribution of data among nodes (processors), second stage describe the multiplication process at each processor, and finally the third stage represent the collecting the data.

Input: Matrix A and B
Output: Matrix C on Tree-Hypercube using parallel Matrix Multiplication

Stage 1: Broadcast matrix A and B

1. CN (Coordinator Node) generates a set of blocks of Matrix A.
2. CN generates a set of blocks according to the Matrix B.

3. Wait for acknowledgment message from the coordinator who received the data.

4. Send a message for the CN informing the process completion.

5. CN stops the process of distribution and announces the beginning of the next Stage

Stage 2: Do the multiplication in parallel

6. Multiply the stripes of matrix A with stripes of Matrix B (for each block) of data.

7. Using sequential matrix Multiplication. Where all processors perform Cij= $\sum_{k=0}^{k-1} \boldsymbol{A(ik)B(kj)}$

Stage 3: Collecting the data

A- Global Data Combining

8. In each level in the Tree-Hypercube interconnection, send in parallel the multiplication matrix to the TH root nodes.

B- Combining Data in the Tree-Hypercube root nodes

9. CN combines the collected matrices correctly from roots nodes in matrix C.

10. Exit

Fig. 3. Pseudocode Tree-Hypercube Algorithm for Matrix Multiplication.

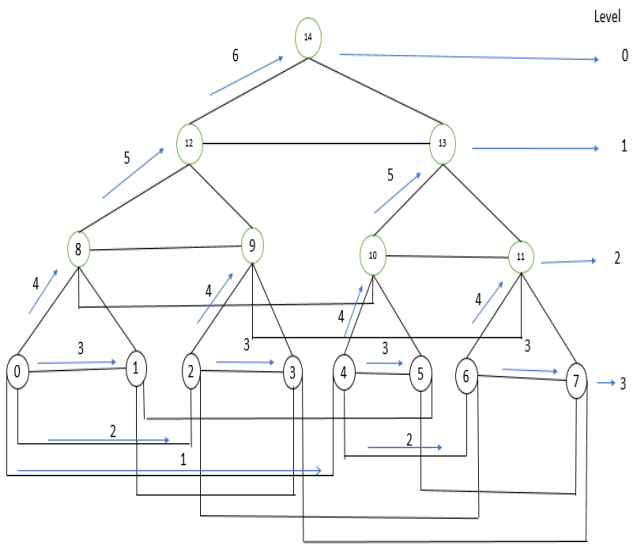

Fig. 4. Parallel matrix multiplication tree-hypercube view.

Fig. 4 shows one-to-all broadcast on a Tree-Hypercube (2,3). Broadcast communication operations are used in many parallel algorithms; such as vector-matrix multiplication, vector-vector multiplication, matrix multiplication, Gaussian elimination, shortest paths, LU (lower–upper) factorization, maximal vectors and prefix sums. Our one-to-all broadcast algorithms on TH network are based on the E-Cube routing algorithm, where this routing algorithm can be found in [1]. Each node can send a message while receiving another message on the same or different link; that is bidirectional links are used. Also, it can send or receive a message on only one of its links at a time. Moreover, a store-and-forward routing technique is used, where each intermediate node must fully receive and store the entire message before sending it to the next node. Fig. 4 presents one-to-all broadcast algorithm which can be defined as broadcasting a message from a source node to all other nodes in the network. Respectively, in matrix multiplication which is applied in tree hypercube, the coordinator creates the partitions depending on the number of processors and the matrix size, once the data is received by any processor it takes the part that it is responsible for and then resends it again to the nodes that has a direct link with it. In TH sending data acts as hypercube in each level, after the data distribution is done. Each processor does it is own computations and sends back the result to the parent from which it got the data until all data is accumulated and sent back to the coordinator.

A. Partition Procedure Analysis

In the first partitioning of the original array will take n/p, assuming the partitioning is always balanced i.e. the number of columns or the numbers of rows are equal to the number of processors.

In our tree-hypercube network only one processor has data, which is called the coordinator. In the first step part of data will be sent from the coordinator to the directly connected processor, secondly, both processors will start sending other parts of data to the processors that are connected directly to them. TH (2,2), each processor will receive a row form matrix A and a column from matrix B, assuming the matrix size is 7x7. Canon algorithm is a distributed algorithm for matrix multiplication for two-dimensional meshes. It is especially suitable for computers laid out in an N × N mesh and it works well in homogeneous 2D grids, so the main advantage of the algorithm is that its storage requirements remain constant and are independent on the number of processors. As shown in Fig. 5, the matrix multiplication using Canon algorithm with 49 processor, it aligns the blocks of A and B in such a way that each process multiplies its local sub-matrices. This is done by shifting all sub-matrices Ai,j to the left (with wraparound) by i steps and all sub-matrices Bi,j up (with wraparound) by j steps [5].

B. Parallel Analysis

Parallel matrix multiplication is analyzed according to the communication time, computation time and complexity. Communication steps involve the distribution of data splitter between the processors, and gathering results. First of all, the number of processors will affect both the data scatter and the communication time, to calculate the initialization time for the tree-hypercube TH. The initialization time will equal the number of steps to distribute data between the processor units. We need 4 steps to scatter data among 7 processors and 6 steps for 15 processors, so we need (2L) for initialization time and the same time to gather the results from all processors. So, the total communication time equals (2L).

Complexity is the time required to perform matrix multiplication in each processor for data size (n/p) which is required for all processors, this means each processor needs (n/p) x (n^3) time, which makes the total complexity ((n^4)/p).

Execution time depends on the number of processor and the size of data as discussed previously the complexity is the time needed for multiplication and the communication time which is (2L) + ((n^2) / (p^2)).

A00	A06	A05	A04	A03	A02	A01
A16	A15	A14	A13	A12	A11	A10
A25	A24	A23	A22	A21	A20	A26
A34	A33	A32	A31	A30	A36	A35
A43	A42	A41	A40	A46	A45	A44
A52	A51	A50	A56	A55	A54	A53
A61	A60	A66	A65	A64	A63	A62

×

B00	B61	B52	B43	B34	B25	B16
B60	B51	B42	B33	B24	B15	B06
B50	B41	B32	B23	B14	B05	B66
B40	B31	B22	B13	B04	B65	B56
B30	B21	B12	B03	B64	B55	B46
B20	B11	B02	B63	B54	B45	B36
B10	B01	B62	B53	B44	B35	B26

C00	C01	C02	C03	C04	C05	C06
C10	C11	C12	C13	C14	C15	C16
C20	C21	C22	C23	C24	C25	C26
C30	C31	C32	C33	C34	C35	C36
C40	C41	C42	C43	C44	C45	C46
C50	C51	C52	C53	C54	C55	C56
C60	C61	C65	C65	C65	C65	C66

Fig. 5. Initial data distribution of matrix A 7×7, B 7×7 and C 7×7 on 49 processors (2,2).

IV. Evolution Result

Results are evaluated in terms of speedup, running time and the parallel efficiency of matrix multiplication. The proposed parallel matrix multiplication on the Tree-Hypercube network is implemented by the library Message Passing Interface MPI, in which MPI processes are assigned to the cores. It will be parallel computation if the MPI process is assigned to a core; but if more than one MPI process is assigned to the same core, then it will be concurrent computation. These results were produced in IMAN1 Zaina cluster. Also, open access MPI library was used in our implementation. The experiment was applied on tree-hypercube topology with different size of matrices and different number of processors in reference to the number of nodes in a tree-hypercube, for example in the below table the number of processors represent the number of nodes inside tree-hypercube definition. After running this experiment multiple times, the average was taken as result. The hardware and software specifications are illustrated in Table I.

TABLE I. Hardware and Software Specification

Hardware specification	Dual Quad Core Intel Xeon CPU with SMP 16 GB RAM
Software specification	Scientific Linux 6.4 with open MPI1.5.4 and C++ compiler
Matrix size	1000x1000, 2000x2000,4000x4000
Number of processors	1,3,7,15,31

A. Run Time Evaluation

Fig. 6 depicts run time for sequential matrix multiplication in different size 1000×1000, 2000×2000 and 4000×4000 we can see as the matrix size increase the runtime increases proportionally.

Fig. 7 shows the run time for Parallel matrix multiplication in different sizes 1000×1000, 2000×2000 and 4000×4000 as illustrated in Fig. 7 when the number of processes increases the time needed for multiplication decreases. On the other hand, as shown below the run time at a certain number of processes will stop decreasing because the problem becomes smaller than the number of processors and the communication time will increase because this will cause an overhead, the general behaviour can be summarized with the following points:

Scenario 1: When the processors number increases the run time decreases due to parallelism and distribution of tasks on more than one processor that are working together at the same time. (This case is applied when we move from 3 processors to 31 processors.)

Scenario 2: When the processors number increases the run time increases due to communication overhead which is caused from increasing the number of processors more than a specific data size which in its turn decreases the benefits of the parallelism. (This case is applied when we move from 15 processors to 31 processors in 1000×1000 matrices and also may occur in number of processors larger than 31 in 2000×2000 and 4000×4000 matrices.)

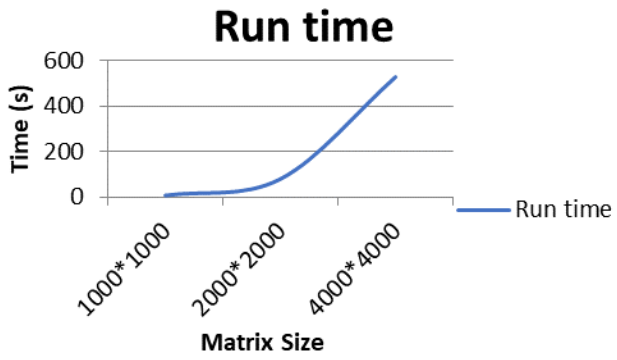

Fig. 6. Runtime for sequential matrix multiplication with different matrix size.

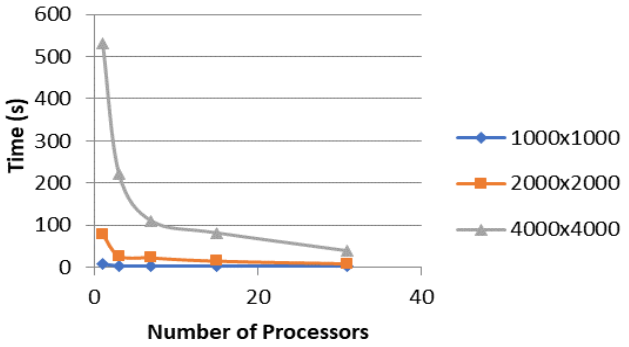

Fig. 7. Runtime of matrix multiplication according to the number of processors and with different matrix size.

B. Speed up Evaluation

The speedup is the ratio of serial to parallel execution time as depicted in Fig. 8. Using (1) the number of processors values tested were 3,7,15 and 31. The speedup increases when the number of processors increases but this is not always applicable because we have limitations for parallelism that appears in matrices with size 1000×1000.

Speedup = sequential processing time/ parallel processing time (1)

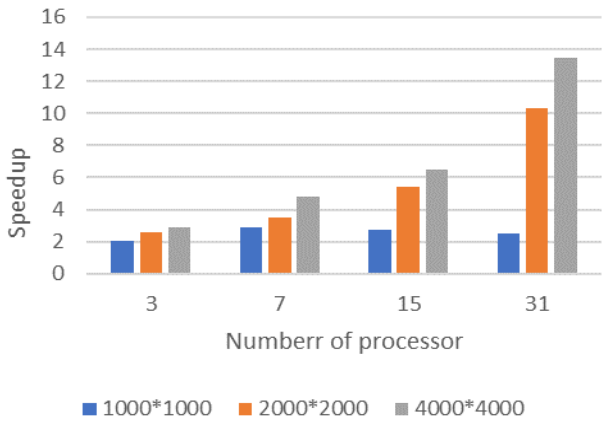

Fig. 8. The Speedup of the three different sizes of matrices on different number of processors.

C. Parallel Efficiency Evaluation

Efficiency is the ratio of speedup to the number of processors. In an ideal parallel system S=P and E=1 but in practice S<P and E is between 0 and 1. Equation 2 is used to calculate the efficiency. Fig. 9 displays the parallel efficiency for matrix multiplication according to different number of processors and different matrix size. In small numbers of processors between (3 and 7) efficiency was 97% for all matrix multiplication of matrix size 4000×4000. On the other hand, when the number of processors increased between (15 and 31) efficiency started to decrease i.e. reached 8% in 1000×1000 with 31 processors because the communication time will increase between the processes. Total communication time equals (2L), so the communication time will increase when the number of processors increases.

The reason why the efficiency goes down, is that the communication time equals 2L where L is number of levels for tree hypercube, as long as the communications processes are increasing the communication time will increase for example when the number of processers is 3 the communication time will be 2 on the other hand when we have 31 processors the communication time will be 8 seconds for the same data size. Also, as seen in equation 2 the number of processes is in inverse relationship with efficiency.

$$\text{Efficiency} = \text{speedup} / \text{number of processors} \quad (2)$$

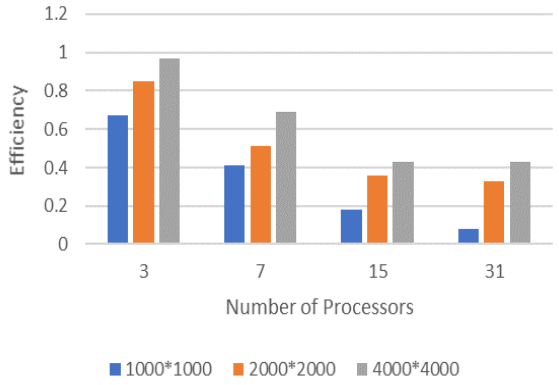

Fig. 9. Efficiency of the three different sizes of matrices on different number of processors.

V. Conclusions and Future Works

In this paper, we present Matrix multiplication using tree-hypercube in terms of running time, speedup and efficiency. The algorithm was implemented using MPI library, and the results were produced by IMAN1 super computer. The evaluation was based on matrix size and number of processors. Results showed that runtime was in general decreasing when the processer's number is increased except for matrix size 1000×1000 when the number of processors exceeded 15 the runtime increased because of the communication overhead. The speedup in all different matrices size increased when the number of processors increased except for the cases in which the communication overhead increases. We achieved up to 97% efficiency in large matrices such as 4000×4000 sized matrix.

In our future work we are aiming to proceed with a comparative study for matrix multiplication using different interconnection networks and it will be applied also on different algorithms such as sorting, then see the performance for each and compare the results with different studies.

References

[1] P. Lee, "Parallel Matrix Multiplication Algorithms on Hypercube Multiprocessors", International Journal of High Speed Computing, (1995).

[2] J. Zavala-Díaz, J. Pérez-Ortega, E. Salazar-Reséndiz, and L. Guadarrama-Rogel matrix,"multiplication with a hypercube algorithm on multi-core processor cluster", DYNA,Vol (82), no(191). 2015

[3] J. Choi,"New Parallel Matrix Multiplication Algorithm on Distributed-Memory" ,Concurrent Computers. High Performance Computing on the Information Superhighway, 1997. HPC Asia '97.Pages: 224 - 229, DOI: 10.1109/HPC.1997.592151

[4] M. Al-Omari, and H. Abu-Salem, "Tree-hypercubes: A multiprocessor interconnection topology", Abhath Al-Yarmouk, 6: 9-24. (1997).

[5] A. Grama, G. Karypis, and V. Kumar, "Introduction to Parallel Computing," Second Edition. Addison Wesley (2003).

[6] S. Panigrahiand, S. Chakraborty ,"Statistical Definition of an Algorithm in PRAM Model & Analysis of 2 × 2 Matrix Multiplication in 2n Processors Using Different Networks", IEEE International Advance Computing Conference (IACC),Pages: 717 - 724, DOI: 10.1109/IAdCC.2014.6779412. (2014)

[7] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels, "Parallel Matrix Multiplication on Memristor-Based Computation-in-Memory Architecture". International Conference on High Performance Computing & Simulation (HPCS)Pages: 759-766, DOI: 10.1109/HPCSim.2016.7568411

[8] M. Qatawneh, "Adaptive Fault Tolerant Routing Algorithm for Tree Hypercube Multicomputer".vol. 2, no. 2, pp. 124-126. (2006).

[9] M. Qatawneh. "Embedding Linear Array Network into the tree-hypercube Network", European Journal of Scientific Research, 10(2). 2005, pp. 72-76.

[10] Md. Nazrul Islam, Md. Shohidul Islam, M.A. Kashem, M.R. Islam, M.S. Islam "An Empirical Distributed Matrix Multiplication Algorithm to Reduce Time Complexity", Proceedings of the International MultiConference of Engineers and Computer Scientists Vol IIIMECS 2009, March 18 - 20, 2009, Hong Kong).

[11] M. Qatawneh. "Multilayer Hex-Cells: A New Class of Hex-Cell Interconnection Networks for Massively Parallel Systems.", International journal of Communications, Network and System Sciences, 4(11). 2011.

[12] M. Qatawneh. "Embedding Binary Tree and Bus into Hex-Cell Interconnection Network", Journal of American Science, 7(12). 2011.

[13] M. Qatawneh. "New Efficient Algorithm for Mapping Linear Array into Hex-Cell Network", International Journal of Advanced Science and Technology, 90, 2016.

[14] M. Saadeh, H. Saadeh, M. Qatawneh, "Performance Evaluation of Parallel Sorting Algorithms on IMAN1 Supercomputer", International Journal of Advanced Science and Technology Vol.95 (2016), pp.57-72.

[15] M. Qatawneh, A. Alamoush, J. Alqatawna. "Section Based Hex-Cell Routing Algorithm (SBHCR)", International Journal of Computer Networks & Communications (IJCNC), 7(1). 2015.

[16] Qatawneh Mohammad, Hebatallah Khattab. "New Routing Algorithm for Hex-Cell Network", International Journal of Future Generation Communication and Networking, 8(2). 2015.

[17] M. Qatawneh, A. Sleit, W. Almobaideen. "Parallel implementation of polygon clipping using transputer", American journal of Applied Science 6 (2). 2009.

[18] A.Grama, A. Gupta, G.Karypis and V.Kumar. . "Introduction to Parallel Computing," second edition, Addison Wesley,January 16, 2003. ISBN: 0-201-64865-2.