A Parallel Search on Hypercube Interconnection Network

Masumeh Damrudi, Kamal Jadidy Aval

Department of Computer Science, Firoozkooh Branch, Islamic Azad University, Firoozkooh, Iran
m.damrudi@gmail.com, k.jadidy@gmail.com

***Abstract*:** Parallel processing is one of the significant research subjects in the field of computer science. Today's real time applications expect higher throughput approaches than years away. There are various problems that require search algorithms as a part of their base operations. Some parallel search algorithms are available albeit we always need to find faster ones due to the new born technologies. This paper introduces a new searching algorithm on hypercube architecture with a tradeoff between the numbers of processor elements and the running time.

***Keywords*:** Parallel processing, Hypercube, Search, SIMD**.**

1. Introduction

Parallel processing is a significant field of study in today's digital world in computer science whilst consumers of real time applications attend to receive the results as soon as possible. Topological property of interconnection networks are utilized in this paper, which leads to more speed in different operations including search, sort, etc.

Cube interconnection network is used in many systems such as iPSC[1] and SGI Origin 2000 [2], which are parallel. Since hypercube has appropriate properties with regard to other topologies, it is one of the most popular interconnection networks [3].

Search algorithms are one of the essential elements of applications. These algorithms are used to solve various problems in computer science. Many applications are using these algorithms such as expert systems, database systems, Game systems, and robot control systems. Some of the search algorithms are discussed in [1].

A parallel searching algorithm on sorted matrices is issued by Sarnath and Xin with *O(1oglogn)* execution time using *O(n/ loglog n)* processors on a CREW PRAM[2], albeit their selection is assumed to be a $n \times m$ sorted matrix. A parallel search having time of *O(log (n+1))* was also proposed by Berman and Paul using tree architecture[4]. Two parallel algorithms with different complexities that are *O(logN)+ O(n/N)* using *N* processors in CREW model and *O(N1/2)* on mesh in EREW model are issued in [3]. CS algorithm on Centralized Diamond architecture is issued in [5] with constant time.

In this paper, we present a new fault tolerant searching algorithm on hypercube interconnection network. The rest of this paper is organized as following. Section 2 explains the hypercube architecture. Section 3 presents the new search algorithm. Subsequently, in section 4 the parallel search algorithm is analyzed. Eventually, Section 5 gives the conclusion.

2. Hypercube Interconnection Network

Hypercube is a topology that has low communication diameter and high bisection width. The communication diameter is logarithmic in the number of processors the same as the tree, pyramid, and mesh-of-trees. Although, the bisection width of the hypercube is linear that is a significant improvement over the pyramid, mesh, and mesh-of-trees [6]. These properties as well as highly symmetric recursive structure of the hypercube support a variety of efficient parallel algorithms [7].

The limitation of this architecture is the number of its node, which is $N=2^n$ for a *n*-dimensional hypercube [8]. It employs *nN/2* links to connect its nodes, and each node has *n* links. The hypercube topology guarantees that no two nodes are more than *n* links apart [9]. The degree and diameter of a hypercube is *logN* [7]. Despite of the mesh, tree, pyramid, and mesh-of-trees, the hypercube is not a fixed-degree network. A node in a hypercube of size *n* is connected to exactly *log n* other nodes[6]. Further, Figure 1 presents a hypercube using two cubes.

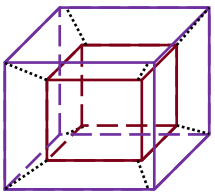

Figure 1. Interconnection network of hypercube, 1st scheme

In the hypercube topology each node is connected to a node, which has just one different bit in the binary form of the index of node. For instance, the node with index zero (000) can connect to nodes with the index of (001), (010), and (100). Hypercube can be demonstrated as Figure **2** for a better scheme.

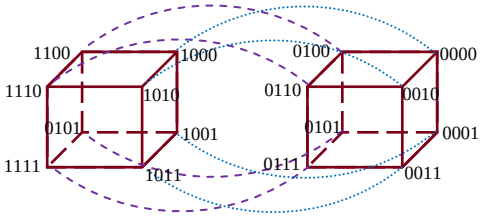

Figure 2. Interconnection network of hypercube, 2nd scheme

The connection of nodes in hypercube have the advantages of different topologies such as mesh and tree all together, which can cooperate to achieve better performance.

3. Parallel Search on Hypercube

Assume *n* is the number of data elements and *N* is the number of Processor Elements (PE) in this algorithm. Each node is a processing element, which just does comparison operation. It is considered that all data elements are located in processing elements previously that are nodes of hypercube. The aim is to search the key value of *k* among data elements that are available and distributed in the nodes of hypercube.

The hypercube is assumed to have two cubes for this example, which have 16 processing element. The following steps will be performed the algorithm to be carried out.

Step 1: At first, *k* is fed into PE0 and this PE compare its data eleements with the value of *k*, if it has a value equal to *k*, it sends one with *k* to PEs 1, 2, 4, and 8. If PE0 doesn't have the value *k*, it will send zero to the mentioned PEs. These steps are demonstrated in Figure 3.

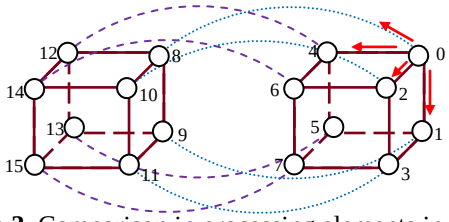

Figure 3. Comparison in processing elements in step one

Step 2: PEs 1, 2, 4, and 8 compare their data elements with *k*. If these two data were equal. they send one, which means the data exists; otherwise they send zero. PE1 sends the results to PEs 3, 5, and 9. PE2 transfers the results to PEs 3, 6, and 10. PE4 gives its results to PEs 5,6, and 12. PE8 sends the results to PEs 9 , 10, and 12. All these operations are carried out at the same time. These steps are illustrated in Figure 4.

Figure 4. Comparison in processing elements in step two

Step 3: PEs 3, 5, 6, 9, 10, and 12 do the same operation and compare their data elements with *k*. PE3 transfers its results to PEs 7, and 11. PE5 sends the results to PEs 7, and 13. PE6 transfers the results to PEs 7, and 14. PE9 gives its results to PEs 11, and 13. PE10 transfers the results to PEs 11, and 14. PE12 sends the results to PEs 13, and 14. These activities are performed simultaneously.

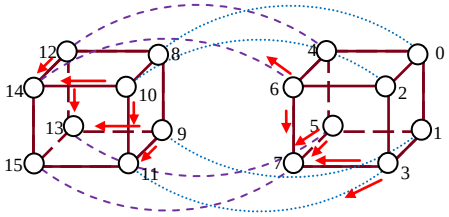

Figure 5. Comparison in processing elements in step three

Step 4: PEs 7, 11, 13, and 14 accomplish the same operation and compare their data elements with *k*. If each of them had the data, they will transfer one otherwise will send zero. These PEs transfer their results to PE15.

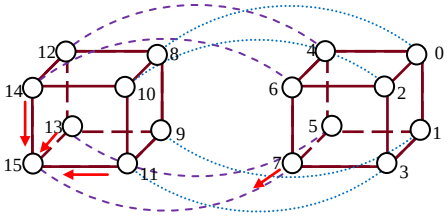

Figure 6. Comparison in processing elements in step four

Step5: PE15 receives the results from mentioned processing elements. These data, which are ones or zeros comprise a number. If this number is zero, it means the key *k*, doesn't exist and if the number becomes greater than zero, it means that data exists. This number also specify which processing elements had the value of *k*.

The indicated steps employ two cubes for hypercube benchmark. Generalizing thses operations for all hypercubes, the following pseudo code can be applied.

$$if\ \ i\ mod\ 8=0; i \to i+1, i \to i+2, i \to i+4, if\ \ i < N-8; i \to i+8$$
$$if\ \ i\ mod\ 8=1; i \to i+2, i \to i+4, if\ \ i < N-8; i \to i+8$$
$$if\ \ i\ mod\ 8=2; i \to i+1, i \to i+4, if\ \ i < N-8; i \to i+8$$
$$if\ \ i\ mod\ 8=3; i \to i+4, if\ \ i < N-8;\ \ i \to i+8$$
$$if\ \ i\ mod\ 8=4; i \to i+1, i \to i+2, if\ \ i < N-8;\ \ i \to i+8$$
$$if\ \ i\ mod\ 8=5; i \to i+2, if\ \ i < N-8;\ \ i \to i+8$$
$$if\ \ i\ mod\ 8=6; i \to i+1,\ \ if\ \ i < N-8; i \to i+8$$
$$if\ \ i\ mod\ 8=7 and\ \ i < N-8; i \to i+8$$

The above operations peresent the relation between PE index and its results' destination. It can be utilized for hybercubes with any number of cubes. The way of analyzing the result will be discussed in the next section.

4. Analysis and Cost

The total execution time for the proposed searching algorithm is *logN*. In the search algorithm, each PE just does one comparison, which is *O(1)*. The total steps are as the same of the diameter of the hypercube. Total execution time is *O(logN)*.

Let's to show that how the algorithm works in steps one to five. Each processing elements have three registers as following. One register have the key value located in PE, the other one is the value of *k*, and the last one is the value, which is received from and sent to processing elements that indicates whether value of *k*, exists or not. This value is displayed with *w*. PE15 produce the final result.

Each PE receives the *k*, and compares with it's value. If there were equal, the PE will compute *or* operation with *w* and a value that have one in the bit equal to the index of PE and and zero in other bits. For instance, if PE3 has the value *k*, it will do the *or* operation with 8 =$(0000000000001000)_2$.

Assume that we are looking for *k=14* among data elements. It's considered in this example that *N=16* and *n=16*. Figure 7 presents that PE6 has this value.

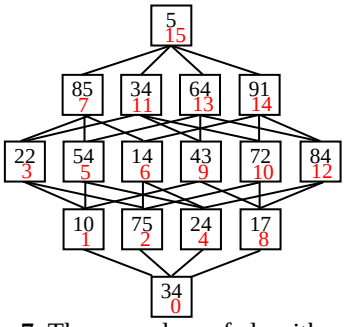

Figure 7. The procedure of algorithm for *k=14*

Starting from step 1, PE0 value is not equal to 14,then sends *w=0* with *k* to PEs 1, 2, 4, and 8. In step 2, none of PEs have equal value to 14 too. Therefore, they send *w*=0 to the PEs of next step. In the step 3, PE6 has the value. Hence, it performs *or* operation with *w=0* and value of 64. Value of 64 has one in sixth bit. The PEs of next step don't have value 14, they just transfer the *w* and *k*. In the step 5, the final result will be *w*=64(0000000001000000), which demonstrates that PE6 has this value.

All these parts in each step work parallel. In the last step, a number will be produced. This number is greater than or equal to zero, which will be different based on the place of key *k*.

5. Simulation and Analysis

The architecture of the proposed search algorithm is simulated using Matlab Simulink R2008a to achieve the results proving the proper work of the algorithm. Figure 8 demonstrates this architecture and the results for mentioned example. This simulation employs logic blocks to perform the proposed algorithm with the number of inputs for each PE assigned based on the connections that a cube requires. In this figure, PE6 has a value equal to *k*. The value of *k* is considered to be 14. The value that PE6 will execute *or* operation with *w* is 64. The result is shown using a display block.

The issued architecture for this search algorithm is a homogeneous architecture, which is desirable due to the time of execution. All PEs in each step will start at the same time and finish concurrently. There is no delay due to waiting for the results of other PEs in each step. The complexity of this algorithm is *O(logN)*.

The pipelined approach of this search algorithm can be applied, which leads to better results and performance. Utilizing pipelining, the PEs that were idle in mentioned algorithm, will be busy and the performance an throughput will be increased. In the pipelining approach all the PEs will always be busy.

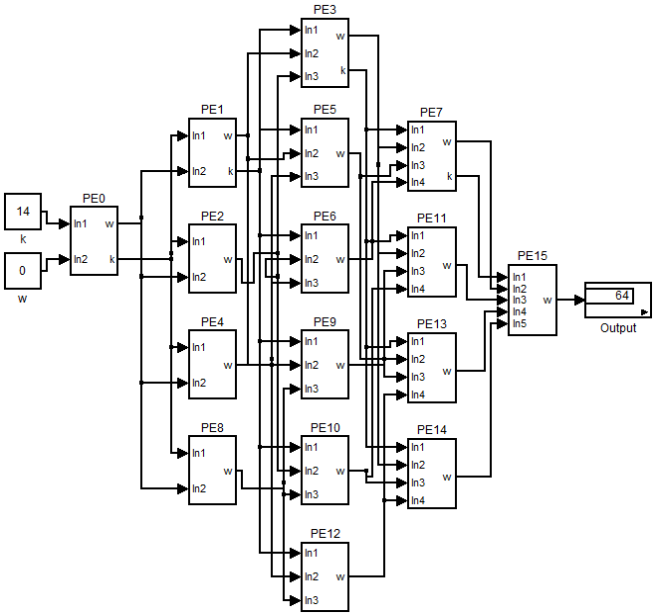

Figure 8. The architecture of hypercube in Matlab Simulink

The throughput of pipeline approch to the approch without pipelining can be computed as following.

$$Throughput_{pipeline} = logN\ Throughput_{without\_pipeline} \quad (1)$$

The work of Sarnath and Xin with *O(1oglogn)* execution time is limited the input file to be sorted while the proposed search algorithm doesn't have this limitation. The CS[5] search algorithm has a constant time, which is better than proposed algorithm. The excellence of the hypercube based search algorithm is because it's a fault tolerant algorithm. If one PE doesn't work, the results of other PEs will be obtained. It's due to transferring the result from more than one PE. For instance, the results of PE5 will be transferred via PEs 7, and 13.

6. Conclusion

We proposed a new fault tolerant search algorithm on hypercube using a new technique. Many search algorithms are used nowadays, which are necessary in respect of importance of time in digital environment.

This search algorithm has an acceptable order using a appropriate hardware cost where execution time is *O(logN)* with fault tolerrance property. The pipelined of this search algorithm will acquire more throughput which leads to better performance.

References

[1] H. El-Rewini and M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing*. USA: John Wiley & Sons Publishing, 2005.

[2] D. Jiang and J. P. Singh, "A methodology and an evaluation of the SGI Origin2000," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, pp. 171-181, 1998.

[3] D. K. Guo, *et al.*, "KCube: A novel architecture for interconnection networks," *Information Processing Letters*, vol. 110, pp. 821-825, Sep 2010.

[4] K. A. Berman and J. L. Paul, *Algorithms: sequential, parallel and distributed*. USA: Thomson, 2005.

[5] M. Damrudi and K. Jadidy Aval, "Searching Data Using Centralized Diamond Architecture," *Journal of Communication and Computer*, vol. 8, pp. 807-811, 2011.

[6] R. Miller and L. Boxer, *Algorithms Sequential and Parallel: A Unified Approach*, Second Edition ed. Hingham, Massachusetts, 2005.

[7] B. Parhami, *Introduction to Parallel Processing Algorithms and Architectures*. Santa Barbara, California: KLUWER ACADEMIC PUBLISHERS, 2002.

[8] R. K. Katare and N. S. Chaudhari, "Study of topological property of interconnection networks and its mapping to sparse matrix model," *International Journal of Computer Science and Applications*, vol. 6, pp. 26-39, 2009.

[9] J. P. Hayes and T. Mudge, "Hypercube supercomputers," *Proceedings of the IEEE*, vol. 77, pp. 1829-1841, Dec. 1989 1989.