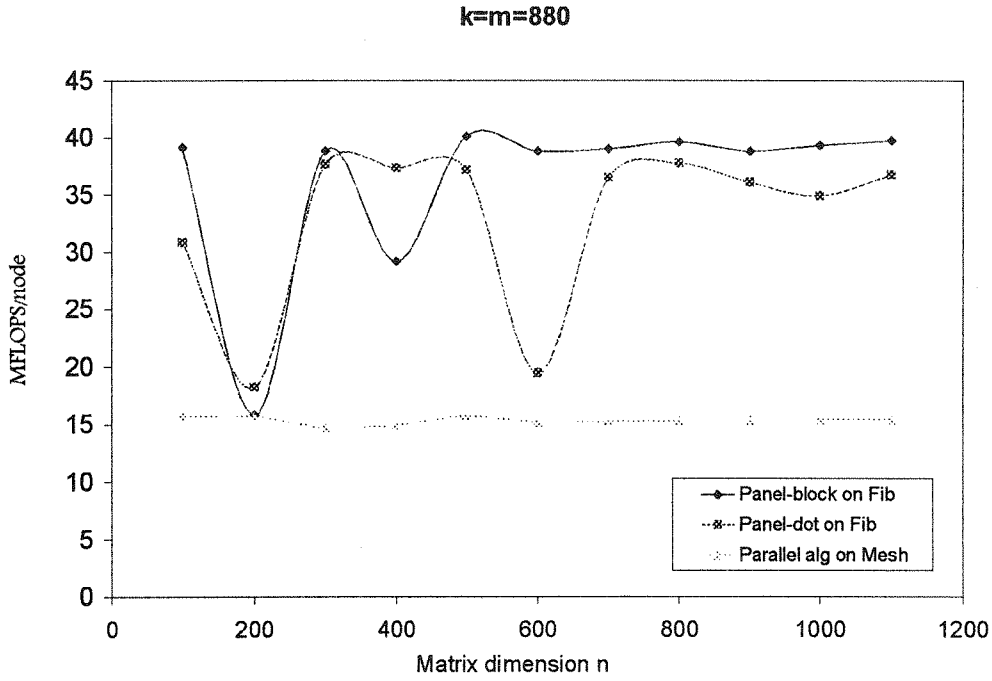

Figure 7: Performance of various parallel matrix multiplication algorithms

6. REFERENCES

[1] S. G. Akl, The *design and Analysis of Parallel Algorithms*, Queens University, Prentice Hall International, Inc, 1989.

[2] A. V. Anisimov, "Fibonacci Hypercube", *International Journal of computer mathematics*, Vol. 25,pp. 221_227, 1997.

[3] A. Chtchekanova, J. Gunnels, G. Morrow, J. Overfelt, R. geijn, "Parallel Implementation of BLAS : General Techniques for level3 Blas", TR_95_40, Department of Computer Sciences, University of Texas, *Intel Research council*, 1995.

[4] G. C. Fox , M. A. Johnson, G .A. Lyzenga, S. Wotto, J. K.Salmon and D. W. Walker , *Solving Problems on Concurrent Processors*, Prentice Hall, Englewood cliffs, N. J. Vol.1, 1998.

[5] J. Gunnels, C. Lin, Greg Morrow, R. Geijn, "Analysis of a Class of parallel Matrix Multiplication Algorithms", *A Technical Paper Submitted to IPPS 98*, 1998.

[6] R. Geijn, "Scalable Universal Matrix Multiplication Algorithms , Concurrency" : *Practice and Experience* , Vol. 9(4), pp. 255_274 ,1995.

[7] R. V. Geijn,"Using PLAPACK : Parallel Linear Algebra Package", *The MIT Press*, 1997.

[8] B. Grayson, A. Pankaj shah, R. Geijn, "A High Performance Strassen Implementation", *Intel Research council*, June13. 1995.

[9] J. M. Jadish, *Parallel Algorithms and Matrix Computation*, University of Cambridge, Clarn Don Pess. Oxford , 1998.

[10] L. Jokar, "Parallel Algorithms for Matrix Multiplication", *M.S Thesis*, Department of mathematics and computer science, Tehran university, 2002.

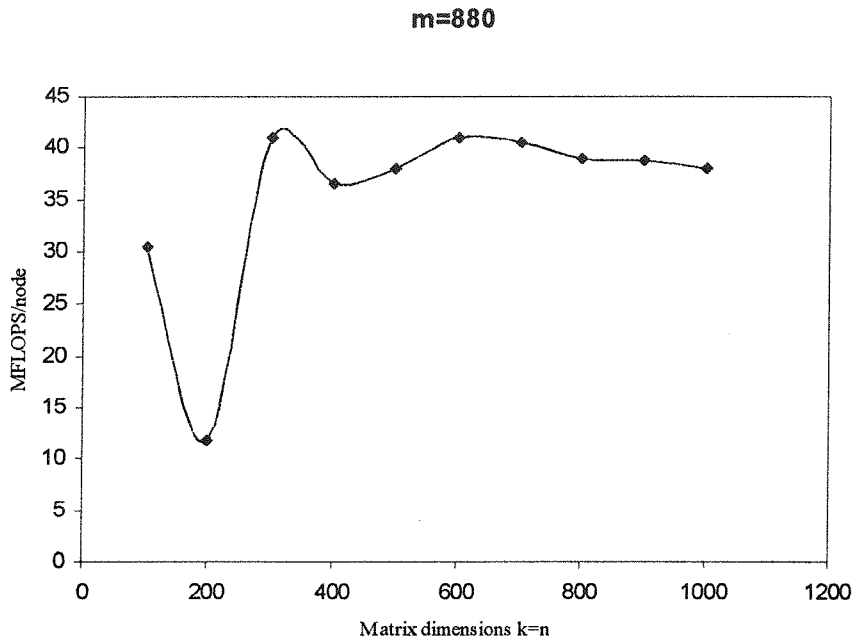

Figure 5: Performance of parallel matrix multiplication algorithm based on panel-block that designed to run on $C_{F,4}$

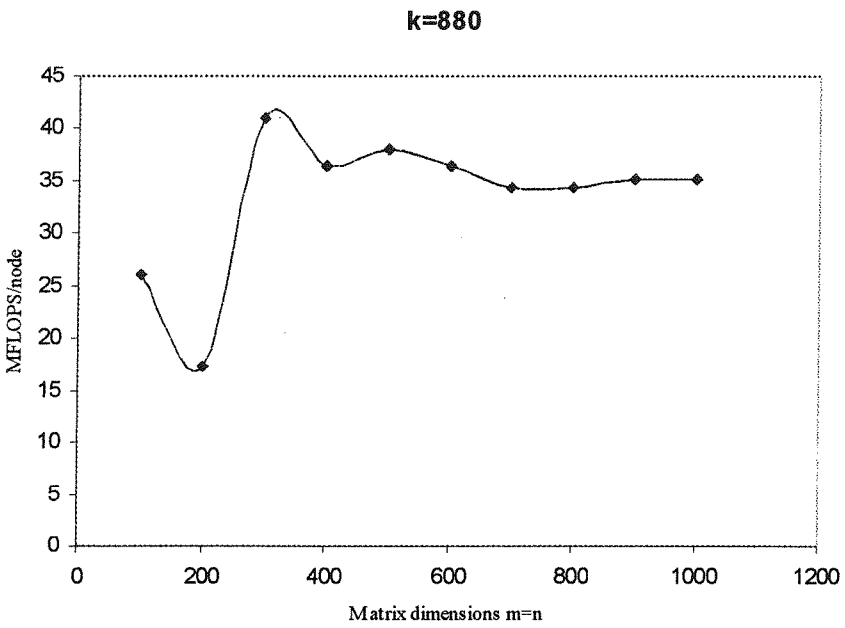

Figure 6: Performance of parallel matrix multiplication algorithm based on panel-dot that designed to run on $C_{F,4}$.

matrix multiplication algorithm based on panel-block that are designed to run on $C_{F,4}$. For the case where $m >> n_1 f_{n_1+2}$, this algorithm is cost-optimal, we present results for the case where m dimension is fixed and large. It is clear from the graph (in accordance to the Figure 5) that to increase dimension of k in comparison to the dimension of m, the speed of performance of algorithm will be reduced. Therefore if dimension of m is fixed and large this algorithm will be an improved algorithm.

In Figure 6, we show the performance of parallel matrix multiplication algorithm based on panel-dot that are designed to run on $C_{F,4}$. Considering, for the case where $k >> n_1 f_{n_1+2}$, this algorithm is cost-optimal, we present results for the case where k dimension is fixed and large. It is clear from the graph (in accordance to the Figure 6) that to increase dimension of m in comparison to the dimension of k, the time of algorithm performance will be increased. Therefore if dimension of k is fixed and large this algorithm will be an improved algorithm.

In Figure 7, the results of the performance of panel-block and panel-dot algorithms that are designed to run on $C_{F,4}$ and parallel multiplication algorithm that is designed to run on a 4×2 mesh structure [1] were compared. We present results for the case where both dimensions of k and m are fixed and large. It is clear from the graphs that the panel-block algorithm has a better performance time in respect to the other two algorithms. Also, the performance time of both panel-block and panel-dot algorithms on the Fibonacci hypercube structure were much better than performance time of the parallel multiplication algorithm on the mesh structure.

5. Conclusion

The main drawback of the hypercube structure, is the necessary increase of each vertex valence while increasing a hypercube dimension. This means that the value of the communication hardware grows faster than a hypercube dimension. Taking into consideration that increase of the hypercube dimension and vertices of higher dimensional mesh will lead to a considerable rise in the number of connections of processors and consequently this will lead to a fast growth in the hardware. By using the presented algorithms on the Fibonacci hypercube structure, the cost of connection amongst the processors will be reduced (because the Fibonacci hypercube structure has the same property as hypercube structure but with fewer connectors plus the same number of vertices).

The panel-block algorithm is designed to run on C_{F,n_1} structure. As we know the number of processors in this structure is f_{n_1+2}, where n_1 is Fibonacci hypercube dimension. As shown in the previous sections, the time complexity of this algorithm is $O(knn_1t_w + \frac{knm}{f_{n_1+2}})$ and for the case where $m >> n_1 f_{n_1+2}$, this algorithm is cost-optimal. In accordance to the procedure of this algorithm, the steps of algorithm will be repeated by $n*k/b^2$ times, therefore, if the dimension of m is fixed and large, this algorithm would have a better performance time in comparison to the panel-dot algorithm.

The panel-dot algorithm on the Fibonacci hypercube structure with dimension of n_1 uses f_{n_1+2} processors. As shown in the previous sections, the time complexity of this algorithm is $O(nmn_1t_w + \frac{knm}{f_{n_1+2}})$ and for the case where $k >> n_1 f_{n_1+2}$, this algorithm is cost-optimal. Since the steps of this algorithm will be repeated by $n*m/b^2$ times, therefore, if the dimension of k is fixed and large, this algorithm in comparison with panel-block algorithm has a better performance time.

Comparing the graph of the charts in the Figures 5 and 6 it can be said in the points where $k > m$, the speed of performance of panel-dot algorithm is higher than panel-block algorithm. And in the points where $m > k$, the panel-block algorithm has a better performance time than pane-dot algorithm.

Considering dimension of m has a fixed large quantity, the panel-block algorithm in comparison with the panel-dot algorithm will take less time to process and if considering dimension of k has a fixed and large quantity, the panel-dot algorithm in comparison to the panel-block algorithm will take less time to process.

As it is seen in the Figure 7, when both dimensions of k and m are fixed and large, the panel-block algorithm has a better performance time than panel-dot algorithm and parallel multiplication algorithm on mesh structure. Also, the performance time of both panel-block and panel-dot algorithms on the Fibonacci hypercube structure are much better than the performance time of the parallel multiplication algorithm on the mesh structure.

$$Cost=TP*P=O(knn_1 f_{n_1+2}+knm). \quad (10)$$

We know the condition of cost- optimality is:

$$E=\frac{T_{Seq}}{P*T_P}=O(1). \quad (11)$$

Therefore to have cost -optimal algorithm we must have:

$$knn_1 f_{n_1+2} << knm \quad (12)$$

$$\Rightarrow n_1 f_{n_1+2} << m.$$

Therefore, for the case where $n_1 f_{n_1+2} << m$ the algorithm is cost-optimal.

B. Parallel Matrix Multiplication Algorithm Based on Panel-dot

Consider C_{F,n_1} structure, as we know the number of processors in this structure is f_{n_1+2}, where n_1 is Fibonacci hypercube dimension. Assume A and B are two matrices with $m\times k$ and $k\times n$ dimension. Matrix A will be divided to row panel with b size, and Matrix B will be divided into column panel with $b\times b$ dimensions. Therefore, we would have:

$$\begin{bmatrix} C_{0,0} & \cdots & C_{0,n/b} \\ \vdots & \vdots & \vdots \\ C_{m/M,0} & \cdots & B_{m/b,n/b} \end{bmatrix} = \begin{bmatrix} \hat{A}_0 \\ \hat{A}_1 \\ \vdots \\ \hat{A}_{m/b} \end{bmatrix} (B_0 \mid B_1 \mid \ldots \mid B_{n/b}) = \begin{bmatrix} \hat{A}_0 B_0 & \cdots & \hat{A}_0 B_{n/b} \\ \vdots & \vdots & \vdots \\ \hat{A}_{m/b} B_0 & \cdots & \hat{A}_{m/b} B_{n/b} \end{bmatrix}. \quad (13)$$

This algorithm is designed based on multiplication of row panels of A by column panels of B. This multiplication is called multiplication of panel-dot. In this algorithm, from multiplication of row panel $\hat{A}_S$ by column panel B_h, block $C_{s,h}$ of matrix C will be obtained.

In this algorithm, the following steps will be repeated by $n/b \times m/b$ times.

1) Members of panel $\hat{A}_S$ will be distributed amongst all processors in the vector based method.

2) Members of panel B_h will be distributed amongst all processors in the vector based method.

3) The result of $C(i)=A(i)*B(i)$ will be computed by all the processors simultaneously.

4) With sum to one processor, the result will be saved in $C_{s,h}$.

Algorithm procedure is shown as follows:

Procedure Fib_Panel_dot_Multip(A, B, C)

```
Begin
  For h=0 to m/b Do
    For s=0 to n/b Do
      Begin
        For i=0 to f_{n1+2}  Do in Parallel
          Begin
            A(i)←A_{h,i} ;
            B(i)←B_{i,s} ;
            C(i)=A(i)×B(i) ;
          end
        Sum_to_one( C_{h,s} );
      end
  end.
```

Figure 4: Panel _dot Algorithm in C_{F,n_1} structure

Cost-Optimality:

The first and second steps of algorithm take constant time. In the third step, the multiplication of two matrices with the dimensions of $b\times k/f_{n_1+2}$ and $k/f_{n_1+2}\times b$ will be computed by all the processors simultaneously. Therefore, this multiplication takes $(b^2 * k/f_{n_1+2})$ time. The fourth step of algorithm takes $(t_S+b^2 t_W)n_1$ time. There are $n/b\times m/b$ iterations of steps 1 to 4 [10]. Thus, the algorithm time complexity is equal to:

$$T_P = O(n/b * m/b (b^2 * k/f_{n_1+2} + (t_s + b^2 t_w)n_1)) = O(\frac{knm}{f_{n_1+2}} + nmn_1 t_w). \quad (14)$$

Then cost of algorithm will be equal to:

$$Cost=TP*P=O(knm+nmn_1 f_{n_1+2}). \quad (15)$$

We know the condition of cost -optimality is:

$$E=\frac{T_{Seq}}{P*T_P}=O(1). \quad (16)$$

Therefore, to have cost -optimal algorithm we must have:

$$nmn_1 f_{n_1+2} << knm$$

$$\Rightarrow n_1 f_{n_1+2} << k. \quad (17)$$

Therefore, for the case where $n_1 f_{n_1+2} << k$ the algorithm is cost-optimal.

4. PERFORMANCE RESULTS

This section reports the performance of the algorithms that we presented in the previous sections, on a system with 8 nodes.

In Figure 5, we show the performance of parallel

If P_0 did not receive the data then $P_0 \leftarrow x$;
If n=0 Return();
If n=1 then P_0 Send Data to P_1;
Else
{ /*adjacent function returns the adjacent processor P_0 that is in bit nth differ.*/
P_0 Send Data to adjacent(P_0,n);
do in parallel
{
Broadcast_Fib(n-1,x);
Broadcast_Fib(n-2,x);
}
}
}

Figure 2: Broadcasting algorithm in Fibonacci hypercube $C_{F,n}$

E. Model of Communication Cost

We will assume that in the absence of network conflict, sending a message of length n between any two nodes can be models by:

$$t_s + nt_w. \tag{7}$$

where t_s represents the latency (startup cost) and t_w represents the cost per byte transferred.

3. PARALLEL ALGORITHMS

In this part, two optimal parallel algorithms are given for the matrix multiplication on the Fibonacci hypercube structure. We show these algorithms are cost-optimal.

Parallel Matrix Multiplication Algorithm Based on Panel-block

If we consider C_{F,n_1} structure, as we know the number of processors in this structure is f_{n_1+2} that n_1 is Fibonacci hypercube dimension. Assuming A and B are two matrices with $m\times k$ and $k\times n$ dimension, Matrix A will be divided into column panel each with b size, and Matrix B will be divided into blocks with $b\times b$ dimensions. Therefore, we would have:

$$(C_0|C_1|...|C_{n/b}) = (A_0|A_1|...|A_{k/b}) \begin{bmatrix} B_{0,0} & \cdots & B_{0,n/b} \\ \vdots & \vdots & \vdots \\ B_{k/b,0} & \cdots & B_{k/b,n/b} \end{bmatrix} \tag{8}$$
$$= (A_0B_{0,0} + \cdots + A_{k/b}B_{k/b,0} | \cdots | A_0B_{0,n/b} + \cdots + A_{k/b}B_{k/b,n/b}).$$

This algorithm is designed based on multiplication of column panels of A by blocks of B. This multiplication is called multiplication of panel-block. In this algorithm, from the sum of multiplication of the column panels of matrix A by block of ith column of matrix B, panel C_i of matrix C will be obtained.

In this algorithm the following steps will be repeated by $n/b \times k/b$ times.

1) Members of panel A_s will be distributed amongst all the processors in the vector based method.

2) A block from matrix B, will be distributed in every processor.

3) The result of $C(i)=C(i)+A(i)*B(i)$ will be computed by all the processors simultaneously.

4) After the steps 1 to 3 repeated by k/b times. In gathering operation, a panel of matrix C will be computed.

Algorithm procedure is shown as follows:

Procedure Fib_Panel_block_Multip(A, B, C)
Begin
For $h=0$ to n/b Do
Begin
For i=0 to f_{n_1+2} Do in Parallel
$C(i) \leftarrow 0$;
For $s=0$ to k/b Do
Begin
For $i=0$ to f_{n_1+2} Do in Parallel
$A(i) \leftarrow A_{i,s}$;
Broadcast_Fib(n_1,$B_{s,h}$)
For i=0 to f_{n_1+2} Do in Parallel
$C(i)=C(i)+A(i)\times B(i)$;
end
Gather to C_h;
end
end.

Figure 3: Panel-block Algorithm in C_{F,n_1} structure

Cost-Optimality:

In the first step of algorithm, A_s in the constant time will be distributed amongst all the processors in the vector based method. The second step of algorithm takes $(t_s + b^2 t_w)n_1$ time. In the third step, the multiplication of two matrices with the dimensions of $m/f_{n_1+2} \times b$ and $b\times b$ will be computed by all the processors simultaneously. Therefore, this multiplication takes $(m/f_{n_1+2} * b^2)$ time. The fourth step takes constant time. There are $n/b \times k/b$ iterations of steps 1 to 4 [10]. Thus, the algorithm time complexity is equal to:

$$T_P = O(k/b * n/b((t_s + b^2 t_w)n_1 + m/f_{n_1+2} * b^2)) \tag{9}$$
$$= O(knn_1 t_w + \frac{knm}{f_{n_1+2}}).$$

Then cost of algorithm will be equal to:

dimension. We assume that the $C_{F,n}$ shows a Fibonacci hypercube with a number of f_{n+2} vertices which in that index, each vertex can be shown with n bit. Fibonacci hypercube with n dimension can be presented as n_Fibonacci hypercube.

If we show the connections of the processors as a graph, therein the set of all edges in $C_{F,n}$ presented as E_n and the set of all labels of vertices of $C_{F,n}$ structure presented as FC_n . Also the number of vertices of $C_{F,n}$ can be presented by $|FC_n|$.

In Figure 1, Fibonacci hypercubes $C_{F,0}$, $C_{F,1}$, $C_{F,2}$, $C_{F,3}$ are depicted.

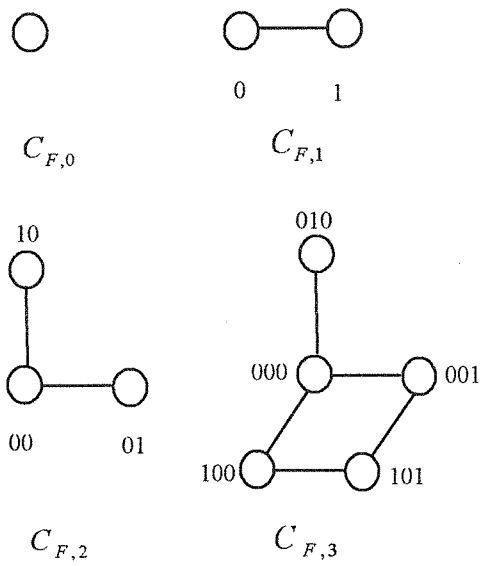

Figure 1: Fibonacci hypercubes $C_{F,0}$, $C_{F,1}$,$C_{F,2}$, $C_{F,3}$

B. Properties of the Fibonacci Hypercube

1st Property: $FC_{n+1}=0FC_n+10FC_{n-1}$. (1)

2nd Property:

$$FC_{n+k}=FC_k0FC_{n-1}+FC_{k-1}010FC_{n-2} \quad (2)$$

3rd Property: Number of vertices of $C_{F,n}$ is equal to f_{n+2}.

4th Property: If a and b are adjacent vertices in $C_{F,n}$ then $H(a,b)=1$ [2] and if $a,b\in FC_n$ and $H(a,b)=1$ then a and b are the adjacent vertices in $C_{F,n}$.

C. Data Decomposition

For simplicity, we will assume that the dimensions m, n and k are integer multiples of the algorithmic block size b. When discussing these algorithms, we use the following partitioning A , B and C:

$$X=(X_0|X_1|\cdots|X_{n_x/b})=\begin{pmatrix}\hat{X}_0\\ \hat{X}_1\\ \vdots\\ \hat{X}_{n_x/b}\end{pmatrix}. \quad (3)$$

where $X\in\{A,B,C\}$ and m_x and n_x are the row and column dimension of the indicated matrix.

Also,

$$X=\begin{pmatrix}X_{0,0} & X_{0,1} & \cdots & X_{0,n_x/b}\\ X_{1,0} & X_{1,1} & \cdots & X_{1,n/b}\\ \vdots & \vdots & \vdots & \vdots\\ X_{m/b,0} & X_{m/b,1} & \cdots & X_{m_x/b,n_x/b}\end{pmatrix}. \quad (4)$$

In the above partitioning, the block size b is chosen to maximize the performance of the local matrix-matrix multiplication operation.

D. Data Distribution

I): *Vector based matrix distribution*

It is more natural to start by distributing the problem to nodes, we partition vector x and assign portions of this vector to nodes. The matrix A is then distributed to nodes so that it is consistent with the distribution of the vectors, as we describe below.

Assume that we have p processors. We use the following partitioning

$$x=\begin{pmatrix}x_0\\ x_1\\ \vdots\\ x_{p-1}\end{pmatrix}. \quad (5)$$

This vector is distributed by assigning x_i to P_i. We call such matrix distribution, vector based.

II): *Broadcasting a datum to Fibonacci hypercube architecture*

We broadcast a datum to Fibonacci hypercube $C_{F,n}$ with $O(n)$ time complexity. According to the first property and structure of Fibonacci hypercube the following identities for time complexity can be obtained:

$$\left.\begin{array}{l}T(C_{F,n})=T(C_{F,n-1})+1=T(C_{F,n-2})+2=\ldots=T(C_{F,0})+n\\ T(C_{F,n})=T(C_{F,0})+n\\ T(C_{F,0})=1\end{array}\right\}\Rightarrow T(C_{F,n})=O(n). \quad (6)$$

The algorithm for broadcast to $C_{F,n}$ is as follows:

```
Broadcast_Fib(n,x)
{
```

Optimal Parallel Matrix Multiplication Algorithms On a Fibonacci Hypercube Structure

L. Jokar[i]; H. Ahrabian [ii]

ABSTRACT

In this paper, we give two new optimal parallel algorithms for matrix multiplication which are designed to run on a Fibonacci hypercube structure. At first, we present a broadcast algorithm on Fibonacci hypercube with $O(n)$ time complexity. We use this broadcast algorithm and algorithms presented by Gunnels, Lin, Morrow and Geijn on a mesh structure, in order to present two optimal parallel matrix multiplication algorithms on a Fibonacci hypercube structure. We show these algorithms are cost-optimal. The performance of the algorithm has been tested on a simulative parallel system.

KEYWORDS

Broadcast, Cost-Optimal, Fibonacci Hypercube, Matrix Multiplication.

1. INTRODUCTION

In the last three decades, numerous parallel algorithms were implemented for the matrix multiplication. The most common parallel matrix multiplication algorithms were Fox's algorithm, Cannon's algorithm, Strassen's algorithm and parallel algorithms which were designed to run on Mesh and Hypercube structures [1], [4], [8], [9]. In 1998, Gunnels, Lin, Morrow and Geijn with regard to various dimension sizes of matrixes, presented 5 parallel algorithms for the multiplication of matrixes on a mesh structure [3], [5], [6], [7].

As you know, all the parallel computers consist of many types of Mesh, Hypercube and Perfect shuffle structures. Even though the Hypercube structures are very useful, they also have their own problems. More of Hypercube dimensions will lead to a rise in the number of processors and connection between them. Consequently, this will lead to a fast growth in the hardware. For this reason, in recent years researchers were searching for a new structure which could substitute the hypercube and to eliminate this problem to some extent. In 1997, the Fibonacci hypercube structure was recommended by Anisimov [2]. Anisimov proved that the Fibonacci hypercube has the same property as hypercube but with fewer connectors plus the same number of vertices.

In this paper, we give two new optimal parallel algorithms for matrix multiplication which are designed to run on this structure. We show these algorithms are cost-optimal.

2. BACKGROUND

A. The Fibonacci Hypercube Structure

The Fibonacci hypercube was introduced by Anisimov [2] initially in 1997. This Fibonacci hypercube is a new topological structure that is obtained recursively using formulas similar to the relation of Fibonacci numbers. It has properties very close to the hypercube. In this part, a very brief explanation is given:

Assume that we have p processors $P_0, P_1, \cdots, P_{P-1}$

Knowing that p is a Fibonacci number and for each processor a binary index is attributed which the binary representation of numbers is substituted by the representation of numbers by sums of Fibonacci numbers. Each processor will be connected to the vertices that their index differs by just in one position. They called this structure the Fibonacci hypercube. In this structure, the number of bits which form the index of vertices is called

i. L. Jokar was M.S Student, Department of Computer Science, School of Mathematics, Statistics, and Computer Science, University of Tehran, Tehran, Iran and is an Educator, Department of ComputerEngineering, Islamic Azad University, University of Rodehen, Tehran, Iran.

ii. H. Ahrabian is an Assistant Professor, Department of Computer Science, School of Mathematics, Statistics, and Computer Science, University of Tehran, Tehran, Iran.