

# Parallel 3-Dimensional DCT Computation on $k$ -Ary $n$ -Cubes

Mehdi Modarressi & Hamid Sarbazi-Azad

IPM School of Computer Science & Sharif University of Technology

Tehran-Iran

modarressi@mehr.sharif.edu , azad@ipm.ir

## Abstract

The three dimensional discrete cosine transform (3D DCT) has been widely used in many applications such as video compression. On the other hand, the  $k$ -ary  $n$ -cube is one of the most popular interconnection networks used in many recent multicomputers. As direct calculation of 3D DCT is very time consuming, many researchers have been working on developing algorithms and special-purpose architectures for fast computation of 3D DCT. This paper proposes a parallel algorithm for efficient calculation of 3D DCT on the  $k$ -ary  $n$ -cube multicomputers. The time complexity of the proposed algorithm is  $O(N)$  for an  $N \times N \times N$  input data cube while direct calculation of 3D DCT has a complexity of  $O(N^6)$ .

## 1. Introduction

The discrete cosine transform (DCT) is widely used in signal processing and especially in image and speech compression. The three dimensional discrete cosine transform (3D DCT), also has been used in many 3D applications such as video compression. Direct calculation of 3D DCT is very time consuming and in real time applications can not be used. Thus, many algorithms and hardware architectures have been proposed by researchers for fast computation of 3D DCT, especially for real time applications.

2D DCT has been already studied well (e.g. see [2]). In [1], a parallel algorithm for computation of 3D DCT, based on butterfly calculation, is introduced. In this paper, a parallel algorithm is proposed for calculating 3D DCT on  $k$ -ary  $n$ -cube interconnection networks. The  $k$ -ary  $n$ -cube is one of the most popular interconnection networks that have been used in most

recent multicomputers, such as Cray T3E and Cray T3D.

## 2. Preliminaries

### 2.1. The 3D DCT transform

The 3D DCT of input data cube  $x$ , with size  $N_1 \times N_2 \times N_3$ , is defined as [3]:

$$X_{k_1, k_2, k_3} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1} x(n_1, n_2, n_3) \cdot \cos(\alpha_1 k_1) \cdot \cos(\alpha_2 k_2) \cdot \cos(\alpha_3 k_3) \quad (1)$$

Where  $k_i = 0, 1, \dots, N_i - 1$  and  $\alpha_i = \frac{\pi(2n_i + 1)}{2N_i}$ .

A fast algorithm for calculating Equation 1 is given in [1]. For lucidity, we assume  $N_1 = N_2 = N_3 = N = 2^l$  in our presentation. The  $N \times N \times N$  point 3D-DCT can be first decomposed into eight  $N/2 \times N/2 \times N/2$  point 3D-DCT. Each  $N/2 \times N/2 \times N/2$  point 3D-DCT is then divided further until we get  $2 \times 2 \times 2$  point transforms. Figure 1 illustrates the phases used to calculate a  $N \times N \times N$  point 3D-DCT using this method.

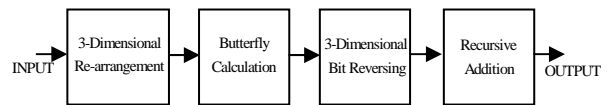
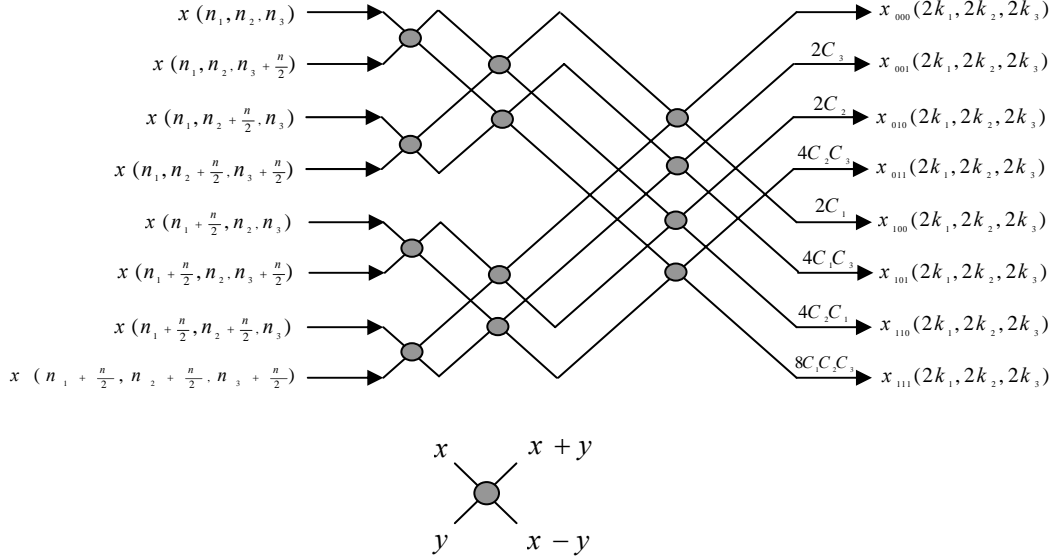


Figure 1- The different phases in calculating 3D DCT

Since this method is based on dividing the  $N \times N \times N$  point 3D-DCT into  $N/2 \times N/2 \times N/2$  point 3D-DCTs until getting to



**Figure 2 - Butterflies calculation,**  $C_i = \cos(\Phi_i)$ ,  $\Phi_i = (4n_i + 1)\pi / 2N$ .

$2 \times 2 \times 2$  transforms, the butterfly calculation and recursive addition phases will be carried out in  $\log(N)$  steps.

The re-arrangement phase performs according to the following code to prepare a proper data arrangement before starting the butterfly calculations. To this end, each data  $x(i, j, k)$  replaces  $x(2N-i-1, j, k)$  if  $i > N/2$  and  $x(2i, j, k)$  otherwise, it replaces  $x(i, 2N-j-1, k)$  if  $j > N/2$  and otherwise  $x(i, 2j, k)$ , and replaces  $x(i, j, 2N-k-1)$  if  $k > N/2$  and  $x(i, j, 2k)$  otherwise.

The structure of butterfly calculation and the equation of recursive additions are depicted in Figure 2 and Equation 2. After butterfly calculation, the 3D bit-reversing phase must be carried out. By migrating the bit-reversing operation to the beginning and after input re-arrangement phase and merging these two phases in the initialization phase [4] and by a very limited modification in the order of butterfly steps, we can eliminate the bit-reversing phase without affecting the computational complexity of the algorithm [5].

In the last phase, recursive addition is realized to get the final results as follows.

$$\begin{bmatrix} x(2k_1, 2k_2, 2k_3) \\ x(2k_1, 2k_2, 2k_3 + 1) \\ x(2k_1, 2k_2 + 1, 2k_3) \\ x(2k_1, 2k_2 + 1, 2k_3 + 1) \\ x(2k_1 + 1, 2k_2, 2k_3) \\ x(2k_1 + 1, 2k_2, 2k_3 + 1) \\ x(2k_1 + 1, 2k_2 + 1, 2k_3) \\ x(2k_1 + 1, 2k_2 + 1, 2k_3 + 1) \end{bmatrix} = \begin{bmatrix} x_{000}(2k_1, 2k_2, 2k_3) \\ x_{001}(2k_1, 2k_2, 2k_3) \\ x_{010}(2k_1, 2k_2, 2k_3) \\ x_{011}(2k_1, 2k_2, 2k_3) \\ x_{100}(2k_1, 2k_2, 2k_3) \\ x_{101}(2k_1, 2k_2, 2k_3) \\ x_{110}(2k_1, 2k_2, 2k_3) \\ x_{111}(2k_1, 2k_2, 2k_3) \end{bmatrix} + \begin{bmatrix} 0 \\ -x(2k_1, 2k_2, 2k_3 - 1) \\ -x(2k_1, 2k_2 - 1, 2k_3) \\ a \\ -x(2k_1 - 1, 2k_2, 2k_3) \\ b \\ c \\ d \end{bmatrix} \quad (2)$$

Where:

$$\begin{aligned} a &= -x(2k_1, 2k_2 + 1, 2k_3 - 1) - x(2k_1, 2k_2 - 1, 2k_3 - 1) \\ &\quad - x(2k_1, 2k_2 - 1, 2k_3 + 1) \\ b &= -x(2k_1 - 1, 2k_2, 2k_3 + 1) \\ &\quad - x(2k_1 + 1, 2k_2, 2k_3 - 1) - x(2k_1 - 1, 2k_2, 2k_3 - 1) \\ c &= -x(2k_1 - 1, 2k_2 + 1, 2k_3) \\ &\quad - x(2k_1 + 1, 2k_2 - 1, 2k_3) - x(2k_1 - 1, 2k_2 - 1, 2k_3) \\ d &= -x(2k_1 + 1, 2k_2 + 1, 2k_3 - 1) - \\ &\quad x(2k_1 + 1, 2k_2 - 1, 2k_3 + 1) - x(2k_1 + 1, 2k_2 - 1, 2k_3 - 1) - \\ &\quad x(2k_1 - 1, 2k_2 + 1, 2k_3 + 1) - x(2k_1 - 1, 2k_2 + 1, 2k_3 - 1) \\ &\quad - x(2k_1 - 1, 2k_2 - 1, 2k_3 + 1) - x(2k_1 - 1, 2k_2 - 1, 2k_3 - 1). \end{aligned}$$

In Equation 2, the second matrix is the output of butterfly calculation phase. In next sections, we show how to realize the above calculation in parallel on a  $k$ -ary  $n$ -cube. To this end, we first describe the parallel algorithm for a  $k$ -ary 3-cube and then extend it for a  $k$ -ary  $n$ -cube.

## 2.2. The $k$ -ary $n$ -cube

An  $nD$   $k_1 \times k_2 \times \dots \times k_n$  torus,  $T_{k_1, k_2, \dots, k_n}$ , has  $\prod_{i=1}^n k_i$  nodes arranged in  $n$  dimensions with  $k_i$  node at dimension  $i$ ,  $1 \leq i \leq n$ . Node  $A$  in  $T_{k_1, k_2, \dots, k_n}$  is labelled with a distinct  $n$ -digit mixed-radix vector  $[a_1, a_2, \dots, a_n]$ , where  $a_i, 0 \leq a_i \leq k_i - 1, 1 \leq i \leq n$ , indicates the position of the node in the  $i^{\text{th}}$  dimension.

In a torus  $T_{k_1, k_2, \dots, k_n}$ , defined over radix vector  $K = [k_1, k_2, \dots, k_n]$ , two nodes  $A = [a_1, a_2, \dots, a_n]$  and  $B = [b_1, b_2, \dots, b_n]$  are interconnected if and only if there is an  $i$ ,  $1 \leq i \leq n$  such that  $a_i = b_i \pm 1 \pmod{k_i}$  and  $a_j = b_j$  for  $1 \leq j \leq n$ ,  $j \neq i$ . Thus, in  $T_{k_1, k_2, \dots, k_n}$  each node is adjacent with two nodes in each dimension, hence  $2n$  nodes in total. A  $k$ -ary  $n$ -cube,  $C_k^n$ , is a variation of torus where each dimension  $i$ ,  $1 \leq i \leq n$ , is of size  $k$ ; i.e.  $C_k^n = T_{\underbrace{k, k, \dots, k}_n}$  [6].

**Theorem 1** [7, 8]. The  $nD$  torus and  $k$ -ary  $n$ -cube networks are Hamiltonian.

### 2.3. Product of networks

Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be two undirected graphs defined on vertex and edge sets  $V_1$ ,  $E_1$ , and  $V_2$ ,  $E_2$ , respectively. The cross product of  $G_1$  and  $G_2$ , denoted as  $G_1 \times G_2$ , is a graph  $G = G_1 \times G_2 = (V, E)$  where

$$V = \{(u, v) \mid u \in V_1, v \in V_2\}, \text{ and}$$

$$E = \{ \langle (u_1, v_1), (u_2, v_2) \rangle \mid ((u_1, u_2) \in E_1 \text{ and } v_1 = v_2) \text{ or } ((v_1, v_2) \in E_2 \text{ and } u_1 = u_2) \} \text{ [9].}$$

Let  $C_k$ ,  $k > 2$ , denote a cycle of  $k$  nodes numbered as 0, 1, ..., and  $k-1$ . Then, the  $k$ -ary  $n$ -cube,  $C_k^n$ , and more generally the  $nD$  torus,  $T_{k_n, k_{n-1}, \dots, k_1}$ , can be defined in terms of cycles as  $C_k^n = \underbrace{C_k \times C_k \times \dots \times C_k}_n$ , and

$$T_{k_1, k_2, \dots, k_n} = C_{k_1} \times C_{k_2} \times \dots \times C_{k_n}.$$

**Corollary 1.** The  $k^{n/3}$ -ary 3-cube is a sub-graph of the  $k$ -ary  $n$ -cube.

### 3. The proposed parallel algorithm

The  $k$ -ary  $n$ -cube, where  $k$  is referred as the radix and  $n$  as dimension, has  $k^n$  processors arranged in  $n$  dimensions with  $k$  processors per dimension. Each processor in the network can be identified by an ' $n$ ' digit address  $(a_1, a_2, \dots, a_n)$ . Two processors  $A(a_1, a_2, \dots, a_n)$  and  $B(b_1, b_2, \dots, b_n)$  are connected if and only if there exists  $i$ ,  $1 \leq i \leq N$ , such that

$a_i = (b_i \pm 1) \pmod{k}$  and  $a_j = b_j$  for  $i \neq j$  and  $1 \leq j \leq N$ . Most of current supercomputers employ  $k$ -ary  $n$ -cubes as their underlying network topology including J-Machine, Cray T3E, Cray T3D and Cray X1.

In the rest of this section, we explain the derivation of the algorithm for a  $k \times k \times k$ -point DCT on a  $k$ -ary 3-cube network. Each point of the input will be allocated on a node in the network. Each node has at least one register to hold the initial value named **value** and one register named **temp** to hold intermediate values arriving from other relevant nodes during the different steps of the algorithm. Let symbol  $\leftarrow$  indicate data movement inside a processor and symbol  $\Leftarrow$  define data communication between two adjacent processors.

#### 3.1. The initialization phase

In this phase, the input data after bit reversing and reordering steps are placed in their proper processing node  $P_{i,j,k}$  located at the  $i$ -th place in dimension  $x$ , the  $j$ -th place in dimension  $y$ , and the  $k$ -th place in dimension  $z$ . The complexity of this step depends on the technique used. This can be easily done in pipelined fashion or in parallel from nodes located at the borders. Let us assume that each node  $P_{i,j,k}$  contains its input data value in variable **value** and variable **temp** is initialized to 0 in each node. The input reordering should be performed according to the following pseudo code.

For all nodes  $P_{i,j,k}$  do in parallel

$$P_{f(i)f(j)f(k)}(x) \Leftarrow P_{i,j,k}(x);$$

where function  $f$  is given by

$$f(t) = \begin{cases} 2N - t - 1, & \text{if } t > N/2 \\ 2t, & \text{otherwise} \end{cases}.$$

Note that the above reordering and bit reversal could be done even when initializing processors just by changing the order of input data to be placed on network processors.

#### 3.2. Butterflies calculations

The process of butterfly calculations is illustrated in Figure 2. This butterfly structure can be mapped onto a

cube as shown in Figure 3. As mentioned before, the  $N \times N \times N$  3D-DCT is divided into some  $N/2$ -point DCTs until we get  $2 \times 2 \times 2$ -point DCTs. Thus, the butterfly calculation phase of the 3D-DCT can be accomplished in  $\log(N)$  steps, for each the procedure (graphically) depicted in Figure 3 must be performed. The distance between the adjacent nodes in each embedded  $2 \times 2 \times 2$  cube is given by  $n$ , which changes from  $N/2$  to 1 (by a factor of  $1/2$  in each step). The steps of this phase (graphically shown in Figure 3), can be declared in pseudo code as follows:

```

n ← N;
while (n > 1) do
Step 1: for all processors  $P_{i,j,k}$  do in parallel
    if  $k < n/2$  then  $P_{i,j,k+n/2}(temp) \leftarrow P_{i,j,k}(value)$ 
    else  $P_{i,j,k-n/2}(temp) \leftarrow P_{i,j,k}(value)$ ;
Step 2: for all processors  $P_{i,j,k}$  do in parallel
    if  $k < n/2$  then  $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) + P_{i,j,k}(temp)$ 
    else  $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) - P_{i,j,k}(temp)$ ;
Step 3: for all processors  $P_{i,j,k}$  do in parallel
    if  $j < n/2$  then  $P_{i,j+n/2,k}(temp) \leftarrow P_{i,j,k}(value)$ 
    else  $P_{i,j-n/2,k}(temp) \leftarrow P_{i,j,k}(value)$ ;
Step 4: for all processors  $P_{i,j,k}$  do in parallel
    if  $j < n/2$  then  $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) + P_{i,j,k}(temp)$ 
    else  $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) - P_{i,j,k}(temp)$ ;
Step 5: for all processors  $P_{i,j,k}$  do in parallel
    if  $i < n/2$  then  $P_{i+n/2,j,k}(temp) \leftarrow P_{i,j,k}(value)$ 
    else  $P_{i-n/2,j,k}(temp) \leftarrow P_{i,j,k}(value)$ ;
Step 6: for all processors  $P_{i,j,k}$  do in parallel
    if  $i < n/2$  then  $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) + P_{i,j,k}(temp)$ 
    else  $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) - P_{i,j,k}(temp)$ ;
Step 7: for all processors  $P_{i,j,k}$  do in parallel
    if  $i \geq n/2$  then
         $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) \times 2\cos(\phi_1)$ ;
    for all processors  $P_{i,j,k}$  do in parallel
    if  $j \geq n/2$  then
         $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) \times 2\cos(\phi_2)$ ;
    for all processors  $P_{i,j,k}$  do in parallel
    if  $k \geq n/2$  then

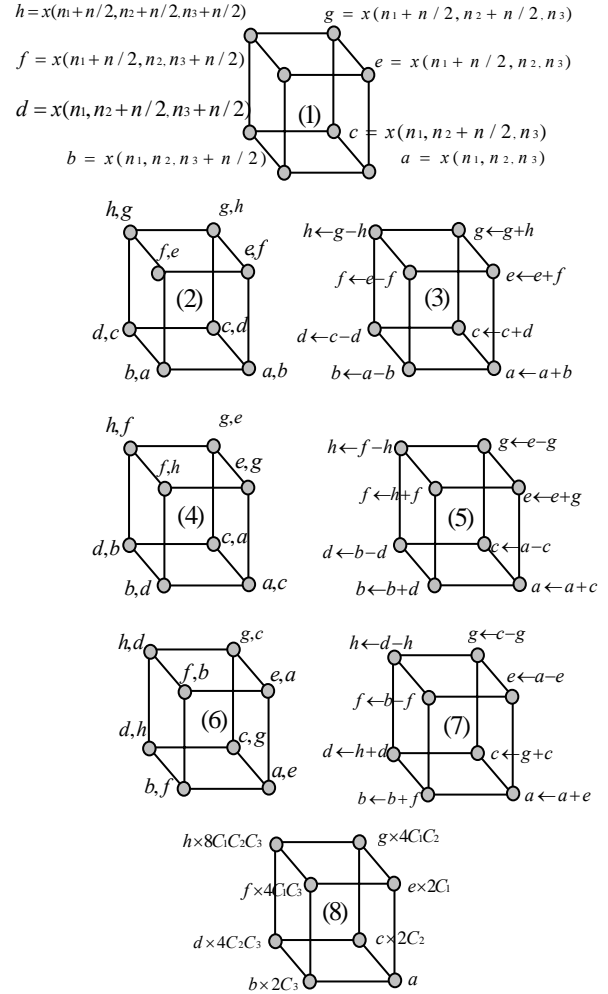
```

```

         $P_{i,j,k}(value) \leftarrow P_{i,j,k}(value) \times 2\cos(\phi_3)$ ;
n ← n/2;
end while;
```

Where  $\phi_i = (4n_i + 1)\pi / 2N$  in the above code.

Now the data is ready for recursive additions.



**Figure 3 - Mapping Butterflies Calculation on a Cube; 1) Addressing in a cube and initial values, 2) send data to adjacent node in k direction, 3) Calculate New Value of Node, 4) send data to adjacent node in j direction, 5) Calculate New Value of Node, 6) send data to adjacent node in i direction, 7) Calculate New Value of Node, 8) Multiply cosine functions.  $C_i$  is a cosine function shown in Figure 2.**

### 3.3. Recursive additions

After butterfly calculation phase, the bit-reversal phase should be performed. Each processor sends its data to the node whose address is the bit-reversal of its address. Bit reversal can be carried out by some fast methods e.g. [10] and [11]. However, we can transfer this step to initial phase as mentioned earlier

In recursive additions phase, according to the algorithm, the network is divided into some partitions in which some  $2 \times 2 \times 2$  cubes are embedded. The recursive additions can be carried out in parallel in all partitions. We first carry out recursive addition in  $2 \times 2 \times 2$  partitions and multiply the size of partitions by 2 at each step (of the total  $\log(N)$  steps). The distance between two peer nodes in the adjacent cubes in a partition referred as 'n' changes from 'N' in the first step, and divided by two in subsequent steps until it becomes 1 in the last step. Exactly similar to the butterfly calculations, every node with address (i, j, k) can have different positions in each  $\log(N)$  steps. This position can be identified by a partition, a  $2 \times 2 \times 2$  sub-cube embedded in the partition, and the vertex in the  $2 \times 2 \times 2$  sub-cube, where the node exists. In each step, we have  $(n/2)^3$  partitions of size

$(2N/n) \times (2N/n) \times (2N/n)$  and each partition has  $(N/n)^3$  cubes where the distance between two adjacent vertices in that cube is  $n/2$ . Thus, in each step, node address is decomposed into a partition base address (x, y, z) such that  $x, y, z \in \{\alpha n | \alpha n < N, \alpha = 0, 1, 2, \dots\}$ , a cube base address in the partition (a, b, c) such that  $0 < a, b, c < n/2 - 1$ , and a vertex address in the cube (p, q, r) such that  $p, q, r \in \{0, n/2\}$ . For example, when  $n=4$ , the node (5, 6, 1) can be decomposed as  $(n \times 1 + 1 + n/2 \times 0, n \times 1 + n/2 \times 1 + 0, n \times 0 + n/2 \times 0 + 1)$  and logically belongs to the node (1,1,0) in the cube (1,0,1) in a partition with base address (0,1,0).

This type of address partitioning indicates that every address which is a multiple of (n, n, n) is a base address of a partition in every step. The value of every node in this phase, according to equation 2, depends on the values of one or more nodes in the adjacent cubes in the same partition. We should note that equation 2 is written for a  $2 \times 2 \times 2$  point DCT. For  $n \times n \times n$  partitions, we should replace '1' and '2' in equation (2) with proper numbers. For example, the second row of equation (2) could be rewritten as

$$x(\alpha k_1, \alpha k_2, \alpha k_3 + n/2) = x_{001}(\alpha k_1, \alpha k_2, \alpha k_3) - x(\alpha k_1, \alpha k_2, \alpha k_3 - n/2),$$

$\alpha = a + b$  such that  $a = \beta n$   $|\beta n < N, \beta = 0, 1, 2, \dots$  and  $0 < b < n/2$ .

We can write  $x(\alpha k_1, \alpha k_2, \alpha k_3 - n/2)$  as  $x(\alpha k_1, \alpha k_2, \alpha k_3 - n + n/2)$  that means the value of node (0,0,1) in any cube in any partition depends on the value of node (0,0,1) in the cube whose third dimension radix is 'n' units less than that in the current cube in the same partition. This cube is adjacent to the current cube in the same partition because the distance between the two peer nodes in adjacent cubes in any partition is 'n'.

By using equation 2, some negative indices appear in the relations. Regarding to the properties of the DCT, we only use the absolute value of node numbers (or indices) in the equations when a negative node address is generated.

The pseudo code for this phase is shown below. Each node can calculate its value immediately after receiving the results of calculations in related nodes according to equation (2). 'Token' in the pseudo code is a variable (or register) that holds the number of values received. When 'Token' indicates that all necessary values for calculating the node value are received, the calculation can be started. After calculation, the data should be sent to those nodes needing it to calculate their value and nodes receiving the data should increment their 'Token' by one.

In this pseudo code, only the code for one class of nodes, nodes having the relative address (0,1,1) in the cubes that their values should be calculated according to the fourth row in equation (2), is presented. The codes for other classes are implemented similarly according to equation (2).

```

if  $P_{i,j,k}$  (value) can be calculated directly
due to negative indices then
{
    Calculate it;
    Ready = TRUE;
}
else if any data is received into temp
then
{
     $P_{i,j,k}$  (value) =  $P_{i,j,k}$  (value) -  $P_{i,j,k}$  (temp);
     $P_{i,j,k}$  (token) =  $P_{i,j,k}$  (token) + 1;
    If  $P_{i,j,k}$  (token) = 3 then Ready = TRUE;
};

if Ready = TRUE then
//send your data to processors needing it according to
equation 2
{
    if  $j+n < N$  then  $P_{i,j+n,k}$  (temp)  $\Leftarrow$   $P_{i,j,k}$  (value);
    if  $k+n < N$  then  $P_{i,j,k+n}$  (temp)  $\Leftarrow$   $P_{i,j,k}$  (value);
    if  $k+n < N$  and  $j+n < N$  then
         $P_{i,j+n,k+n}$  (temp)  $\Leftarrow$   $P_{i,j,k}$  (value);
}

```

#### 4. Complexity analysis and extension for $k$ -ary $n$ -cubes

In this section, we investigate the complexity of the proposed algorithm and suggest some ways to increase the performance of the algorithm and making it faster.

The three dimensional discrete cosine transform (3D DCT) has been widely used in many 3D applications such as video compression. Computing an  $N \times N \times N$  point 3D DCT using the direct formula requires  $3 \times N^6$  multiplications and  $N^6$  additions. This complexity grows fast when the radix  $N$  increases. Parallel computing is the most promising way to compute such a large computational problem (for large  $N$ s) in a reasonable time. In this paper, we first introduced a parallel algorithm for calculating 3D DCT on a 3D torus network. The 3D torus has been widely used by current multicomputers for message passing medium and has been extensively studied in the literature. We then extended the algorithm for  $k$ -ary  $n$ -cubes and also mentioned some suggestions to improve performance when implementing the algorithm. The complexity of the algorithm is as follows: In initialization phase for distributing input data we require  $N^3$  communication steps in the worst case. In butterflies phase, the steps depicted in Figure 3 are performed in parallel over  $2 \times 2 \times 2$  embedded cubes. Since the distance between the adjacent nodes in each embedded  $2 \times 2 \times 2$  cube changes from  $N/2$  to 1, a total running time of  $T_a$  is required for this phase, where  $T_a$  is the addition latency,  $T_m$  is the multiplication latency, and  $T_c$  is the communication latency. In recursive addition phase, a total time of  $(N/2 - 1) \times (7T_c + 3T_a) + T_m$ , is spent. This is the time required for the nodes which have relative address  $(n-1, n-1, n-1)$  in their  $n \times n \times n$  partitions to receive all required values and calculate their values. Clearly, the proposed algorithm is time optimal.

As mentioned before, the 3D-DCT is divided into some  $N/2$  point DCT's until we get  $2 \times 2 \times 2$  point DCT's. Also, it has been observed that the calculation phases of 3D-DCT needs  $\log(N)$  steps. For each step, each node should interact with some other nodes in an embedded  $2 \times 2 \times 2$  cube. The distance between adjacent nodes in the embedded  $2 \times 2 \times 2$  cubes will change from  $N/2$  to 1 (by factor of 0.5) for each step of the phase. So, the positions of nodes vary in each step. Since the scenario repeats for any input sequence, it is wise to calculate the positions of each node in each step of the butterfly calculations and recursive additions and store them in the nodes during an initialization process, when the system starts working. Also the cosine functions multiplied by node values in

the end of butterfly calculation stage only depends on the position of node and can be stored in each node during initialization rather than calculating them for any input sequence.

Using corollary 1, we can realize the proposed algorithm on a  $k$ -ary  $n$ -cube for calculating a radix- $k^{n/3}$  3D DCT.

It is also easy to see, from the definition of  $k$ -ary  $n$ -cubes in section 2, that  $C_k^n = \underbrace{C_k \times C_k \times \dots \times C_k}_n = C_k^{n-3} \times C_k^3$ . Thus, a  $C_k^n$

contains  $k^{n-3}$  identical copies of  $C_k^3$ 's. This means a  $k$ -ary  $n$ -cube can realize  $k^{n-3}$  set of radix- $k$  3D DCT calculations in parallel.

#### 5. Conclusions

Fast computation of the 3D DCT has been widely used in many applications such as video compression and many researchers have been working on developing algorithms and special-purpose architectures for fast computation of 3D DCT.

The  $k$ -ary  $n$ -cube is one of the most popular interconnection networks used in many recent multicomputers. As direct calculation of 3D DCT is very time consuming, in this paper, we introduced a parallel algorithm for efficient calculation of 3D DCT on a  $k$ -ary  $n$ -cubes. The time complexity of the proposed algorithm is of  $O(N)$  for an  $N \times N \times N$  input data cube while direct calculation of 3D DCT has a complexity of  $O(N^6)$ .

Working on parallel algorithms for calculating 3D DCT on other important network topologies such as star graphs is a potential future research in this line.

#### References

- [1] O. Alshibami and S. Boussakta, "Fast Algorithm for the 3D DCT", *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2001, pp. 1945-1948.
- [2] S.C. Chan and K.L. Ho, "A new two dimensional fast cosine transform algorithm", *IEEE Trans. on Signal Processing*, 39(2), 1991, pp. 481-485.
- [3] A. Oppenheim and R. Schaffer, *Discrete Time Signal Processing*, New Jersey: Prentice-Hall, 1999.
- [4] A. N. Skodras and A. G. Costantinides, "Efficient input-reordering algorithms for fast DCT", *Electronics Letters*, 27(21), October 1991, pp. 1973-1975.

- [5] C. A. Christopoulos and A. N. Skodras, "Pruning the Two-Dimensional Fast Cosine Transform", *European Signal Processing Conference (EUSIPCO 94)*, Edinburgh, Scotland, 1994, pp. 596-599.
- [6] H. Sarbazi-Azad, M. Ould-khaoua, L. Mackenzie and S.G. Akl, "On some topological properties of k-ary n-cubes", *Journal of Interconnection Networks*, 4(1), 2004, pp. 79-92.
- [7] H Sarbazi-Azad, M Ould-Khaoua, and L. M. Mackenzie, "Employing  $k$ -Ary  $n$ -Cubes for parallel Lagrange interpolation", *Parallel algorithms and applications*, 16, 2001, pp. 283-299.
- [8] Y Ashir ,and I.A. Stewart, "Fault tolerant of Embedding Hamiltonian circuits in k-ary n-cubes", *Technical Report*, Comp. Sci. Dept., Univ. Leicester, Leicester, U.K.
- [9] K. Day and A. Al-Ayyoub, "The cross product of interconnection networks", *IEEE Transactions on Parallel and Distributed Systems*, 8(2), 1997, pp.109-118.
- [10] A.C. Elster, "*Fast bit-reversal algorithms*", *Proc.ICASSP-89*, pp.1099-1102, Glasgow, Scotland, May 1989.
- [11] J.Jeong and W. J.Williams. "A fast recursive bit\_reversal algorithm", *International Conference on ASSP*, volume 3, 1990, pp. 1511-1514.