A Study of the Communication Cost of the FFT on Torus Multicomputers

Luis Díaz de Cerio, Miguel Valero-García and Antonio González

Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya - Barcelona - Spain
Email: {ldiaz,miguel,antonio}@ac.upc.es

Abstract

In this paper, the computation of a one-dimensional FFT on a c-dimensional torus multicomputer is analyzed. Different approaches are proposed which differ in the way they use the interconnection network. The first approach is based on the multidimensional index mapping technique for the FFT computation. The second approach starts from a hypercube algorithm and then embeds the hypercube onto the torus. The third approach reduces the communication cost of the hypercube algorithm by pipelining the communication operations. A novel methodology to pipeline the communication operations on a torus is proposed. Analytical models are presented to compare the different approaches. This comparison study shows that the best approach depends on the number of dimensions of the torus and the communication start-up and transfer times. The analytical models allow us to select the most efficient approach for the available machine.

1. Introduction

Distributed memory multiprocessors with an interconnection network based on point to point links (multicomputer for short) is a very suitable architectural model for massively parallel computers. Among different interconnection topologies, multidimensional meshes and tori are particularly attractive since they are scalable. Figure 1 illustrates such interconnection topologies. Some of the major supercomputer manufacturers have recently launched multicomputers with a mesh or a torus interconnection network (i.e., the CrayT3D system or the Convex SPP).

Programming a multicomputer is not easy. Not only computations but also data must be distributed among processors. There are several programming models that can be used on this type of computer architecture. One of them is the sequential programming model. In this case the programs are written in a sequential form and the parallelizing compilers perform the distribution among the processors. Some times the parallelizing compilers are helped by user directives, like in High Performance Fortran (HPF)[10], that facilitate the compilation task. On the other hand, a parallel programming model can be used. Two parallel programming models have been proposed: the virtual shared memory model and the message passing model [8]. On the virtual shared memory model the programmer regards the distributed memory as shared and he is not responsible for the data distribution and management but the system hardware/software. In this case, techniques for reducing and tolerating memory latency are crucial in order to obtain a good efficiency of the system [5]. In the message passing model, the communication between processors must be made explicit in the program. The programmer may use message passing interfaces like Parallel Virtual Machine (PVM) [12] or Message Passing Interface (MPI) to do this work [14]. Since in this model the data distribution and management is a responsibility of the programmer, it is widely accepted that message passing programs are more difficult to write than virtual shared memory programs. However, good message passing programs usually obtain better performance than good virtual shared memory programs.

This paper focuses on the message passing model. Figure 2 shows the basic ideas behind this programming model. The computations and data are mapped onto processes. Processes have direct access only to private data. Non private data are accessed through message passing. Since accessing data stored in remote memory modules can be very costly in time, data distribution and management are key issues in the design of message passing algorithms with a low communication overhead on a multicomputer.

This paper focuses on the problem of computing the Fast Fourier Transform (FFT) on a multidimensional torus multiprocessor. The FFT is the computational kernel of many scientific applications, and therefore, an efficient approach to compute it is crucial for such applications.

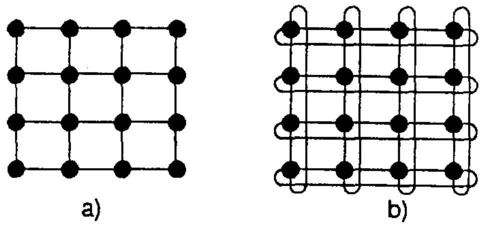

Figure 1: Scalable interconnection topologies: (a) 2-d mesh and (b) 2-d torus.

0-7803-2018-2/95/$4.00 © 1995 IEEE

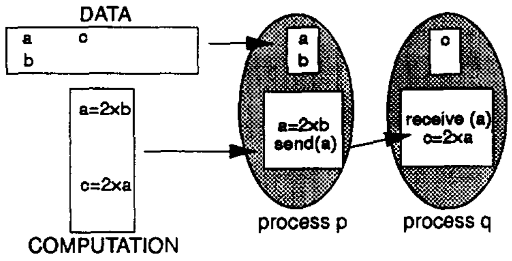

Figure 2: Message Passing Model.

Different algorithms for computing a one-dimensional FFT on a multidimensional torus are evaluated in this paper. These algorithms are based on a mixture of well-known techniques and new schemes proposed by the authors. The algorithms are equivalent in terms of computing cost and load balancing but differ in the way they use the communication bandwidth offered by the interconnection topology.

Three different approaches to compute the FFT are considered. The first one is based on the multidimensional index mapping technique to compute the FFT [1]. The resulting parallel algorithm can be easily mapped onto a multidimensional torus. The second approach starts from a parallel implementation of the radix-2 Cooley-Tukey algorithm for computing the FFT [2]. This parallel algorithm uses a hypercube communication topology and it is mapped onto a multidimensional torus by using the xor embedding of hypercubes onto tori, which has been proposed in [3,4]. The third scheme makes use of the same techniques as the second one but in addition it pipelines the communication operations in order to reduce the communication cost.

The evaluation of the different approaches is carried out by means of analytical models of the algorithms and the architecture. Only the communication component of the parallel algorithms is evaluated and compared, since the algorithms have the same amount of computation and equally load balanced.

The rest of this paper is organized as follows. Section 2 reviews the FFT algorithm. Three different approaches to compute the FFT on a torus are presented in section 3. Section 4 develops analytical model for the three approaches and presents some performance figures for several particular cases. The main conclusions of this study are drawn in section 5.

2. The Fast Fourier Transform (FFT)

The term FFT is used to refer to a class of algorithms to compute the Discrete Fourier Transform (DFT). Given a sequence $\{x_n\}$ of N complex numbers, its one-dimensional DFT is a sequence $\{X_n\}$ of the same size defined as:

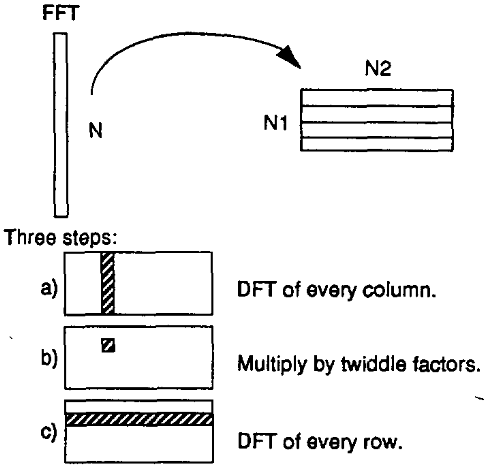

Figure 3: Multidimensional index mapping (decimation in frequency).

$$X_k = \sum_{n=0}^{N-1} W_N^{nk} x_n \qquad ; W_N = e^{-j2\frac{\Pi}{N}} \qquad ; k = 0 \ldots N-1$$

This expression can be regarded as a matrix by vector multiplication. For instance, in the case of N=4 it can be stated as:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^0 & W^2 \\ W^0 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

From this statement of the problem it is easy to see that the direct computation of the DFT requires O(N^2) complex additions and multiplications. A FFT algorithm reduces this amount of computation through the use of the multidimensional index mapping technique [1]. There are two strategies for applying this technique: either decimation in frequency or decimation in time. Since both require the same amount of computation cost and have almost the same dependency graph, this paper considers only decimation in frequency.

Figure 3 shows graphically the multidimensional index mapping for the particular case of a two-dimensional mapping. The original input vector (length N) is arranged as a $N1$-by-$N2$ matrix (N=$N1$x$N2$). The output sequence can be obtained through the following steps:

a) Compute the DFT of the $N2$ columns of the matrix (each DFT has length $N1$).

b) Multiply element (i,j) of the matrix by $W_N{}^{ij}$ (products by twiddle factors).

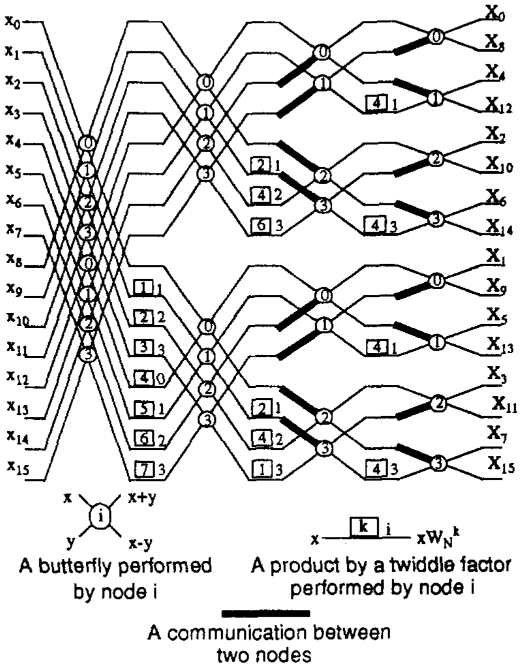
x0
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
X0
X8
X4
X12
X2
X10
X6
X14
X1
X9
X5
X13
X3
X11
X7
X15
x+y
x-y
A butterfly performed by node i
xW_N^k
A product by a twiddle factor performed by node i
A communication between two nodes

Figure 4: A radix-2 Cooley-Tukey FFT of length 16 (decimation in frequency).

c) Compute the DFT of the *N1* rows of the matrix (each DFT has length *N2*).

In this way, the amount of computation is reduced to O(*N*(*N1*+*N2*+*1*)).

The amount of computation can be further reduced if the same technique is applied to compute the DFT of each individual row or column. If *N* is a power of 2 and the technique is recursively applied until the problem is reduced to compute DFTs of size 2 (butterflies), the resulting algorithm is known as the radix-2 Cooley-Tukey algorithm, which is illustrated in figure 4. This algorithm requires O(*N log N*) operations. More details about the radix-2 Cooley-Tukey algorithm can be found in [2].

3. Computing the FFT on a Torus

In this section three different approaches to compute the one-dimensional FFT of length $N=2^n$ on a c-dimensional torus are presented. The particular case of a ring (one-dimensional torus) is considered first since it will be the main building block for the general case.

3.1 Parallel algorithms for a ring

Two different approaches for the parallel computation of the FFT are proposed below. A ring with $P=2^d$ nodes is assumed for the rest of this section.

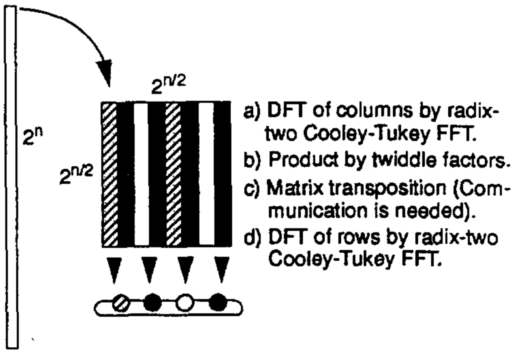
2^n
2^{n/2}
2^{n/2}
a) DFT of columns by radix-two Cooley-Tukey FFT.
b) Product by twiddle factors.
c) Matrix transposition (Communication is needed).
d) DFT of rows by radix-two Cooley-Tukey FFT.

Figure 5: Approach A on a ring of 4 processors.

Approach A

The original input sequence is arranged as a $2^{n/2}$-by-$2^{n/2}$ matrix (assume *n* even), following the two-dimensional index mapping technique. This matrix is distributed by columns among the nodes of the ring. The parallel algorithm is shown in figure 5 for the particular case of *P=4*. Steps (a) and (b) (DFT of columns and products by twiddle factors) can be done in parallel without any communication among nodes. Before step (d) (DFT of rows) the matrix is transposed. This is the only communication required by the algorithm, since after the transposition, the DFT of the rows can be done again in parallel without any communication. A simple approach (among others) for distributing the columns the $2^{n/2}$-by-$2^{n/2}$ matrix is to perform a cyclic data distribution of the original input vector on the ring (element *i* of the input vector is placed in node *(i mod* 2^d) of the ring). Notice that other data distribution schemes are also possible. For instance, consecutive columns of the matrix can be assigned to every node of the ring. The cyclic data distribution scheme has been chosen since it will be the required scheme when considering the generalization of approach A for c-dimensional tori.

To minimize the communication time of this approach, the matrix transposition is performed in an optimal way. As it is shown in figure 6, each processor of the ring has to send a part of each of its columns to every node of the ring. As an example, node 3 in figure 6 has to send blocks 0 to 2 to the left in the ring and blocks 4 to 7 to the right. Block 3 remains in the node. The data to be sent to the 2^{d-1} processors to the left is grouped into a single message and the same is done with the data that is to be sent to the 2^{d-1}*-1* processors to the right. Then, each node sends two messages in parallel, in opposite directions of the ring. When a processor receives a message it extracts the data that was directed to it and forwards the remaining data, again in a single message, to the next processors. As an example, node 2 in figure 6 extracts block 2 from the message received through the right link and forwards the rest of the mes-

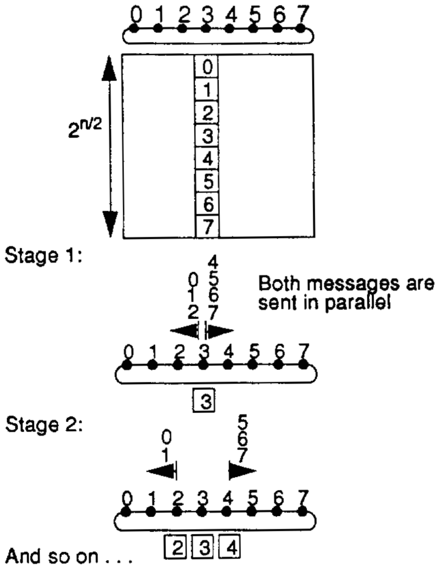

Figure 6: Efficient matrix transposition for approach A.

sage to the left. After 2^{d-1} communication steps, all the nodes have received all their data and the transposition is completed.

Approach B

The second approach consists in computing the radix-2 Cooley-Tukey FFT by using an algorithm with a hypercube topology and then embedding the hypercube onto the ring. Figure 4 shows how the radix-2 Cooley-Tukey FFT of length *16* can be mapped onto a hypercube of $P = 2^d = 4$ nodes. The input sequence is distributed using a cyclic scheme (x_i is allocated to node *i mod P*). The code executed by every node is:

```
do i=0, n-d-1
  compute $2^{n-d-1}$ butterflies
  perform $2^{n-d-1}$ products by twiddle factors
end do
do i=0, d-2
  exchange half of the local data with
    neighbor in dimension i
  compute $2^{n-d-1}$ butterflies
  perform $2^{n-d-1}$ products by twiddle factors
end do
exchange half of the results with neighbor
in dimension d-1
compute $2^{n-d-1}$ butterflies
```

The above algorithm is known as bi-section in [7] or i-cycles in [13]. We can see that the parallel algorithm consists of *d* stages. The first stage performs *n-d* iterations of the FFT algorithm and it does not require any communication. In each of the last *d*-1 stages each node communicates with one of its *d* neighbors in the hypercube. Figure 4 shows which operations are executed by each node and when communication between neighbor nodes is required. Note that not all the nodes must perform 2^{n-d-1} products by twiddle factors in every stage, since some twiddle factors are equal to *1*. Only nodes *1* and *3* in the example execute exactly the code given above.

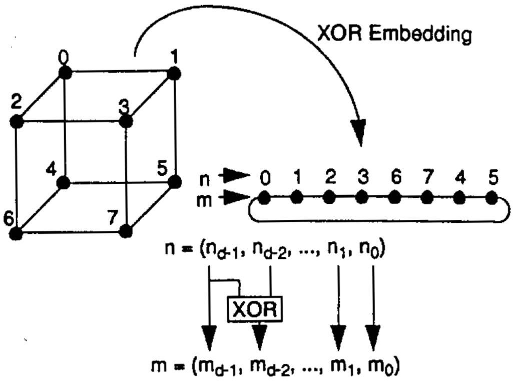
XOR Embedding
0
1
2
3
4
5
6
7
n
m
0 1 2 3 6 7 4 5
$n = (n_{d-1}, n_{d-2}, ..., n_1, n_0)$
XOR
$m = (m_{d-1}, m_{d-2}, ..., m_1, m_0)$

Figure 7: XOR embedding of a 8-node hypercube on a ring.

The hypercube algorithm can be executed on the ring by embedding the hypercube graph onto the ring graph. The embedding function determines in which node of the ring each node of the hypercube is mapped. When neighbor nodes of the hypercube are not mapped onto neighbor nodes on the ring, messages between them are routed through intermediate nodes. A good embedding must keep close in the ring those nodes which are neighbors in the hypercube.

Different approaches have been identified for embedding hypercube algorithms onto meshes and tori [9], [11]. In [3,4] the xor embedding is proposed, which is proved to be optimal for rings. Therefore, we use this embedding to execute the above hypercube algorithm onto the ring. In the following, the xor embedding is defined.

Let *H* be the graph which represents a hypercube of 2^d nodes and *T* be the graph representing the ring with the same number of nodes. Assume that the vertices of *T* are labelled from *0* to 2^d-1 clockwise. Let $(n_{d-1}, n_{d-2}, ..., n_1, n_0)$ be the label (in binary code) of vertex *n* in *H*. This vertex is mapped onto vertex $m = f_{xor}(n)$ in *T*, whose label in binary code $(m_{d-1}, ..., m_0)$ is:

$$m_i = n_i \qquad i \in [0, d-1], i \neq d-2$$

$$m_{d-2} = XOR\ (n_{d-1}, n_{d-2})$$

where *XOR (a,b)* is the exclusive-or of bits *a* and *b*. Figure 7 shows an example for *d=3*.

Note the simplicity of function $f_{xor}(n)$. This function, which is used very frequently for routing messages during the execution of the FFT algorithm, consists of simple bit operations and its computational cost is negligible.

3.2 Parallel algorithms for a c-dimensional torus.

In this section, approaches A and B are generalized for a c-dimensional torus. For clarity, the two-dimensional case is briefly considered first and then the general form is presented.

Approach A

If we have a $2^{d/2}$-by-$2^{d/2}$ two-dimensional torus, the input vector is arranged as a $2^{n/2}$-by-$2^{n/2}$ matrix. This matrix is distributed among the nodes of the torus so that every row is placed in a ring along one dimension of the torus and every column is placed in a ring along the other dimension of the torus. This can be achieved by using a bidimensional cyclic data distribution scheme (element (i,j) of the matrix is stored in node $(i \; mod \; 2^{d/2}, j \; mod \; 2^{d/2})$ of the torus). Assuming that $n/2$ is even it is possible to compute the DFT of each column of the matrix using the approach A described in the previous section to compute a DFT on a ring. After the product by the twiddle factors, the DFT of each row can be computed using again approach A for rings.

To generalize approach A for a c-dimensional torus, the input vector must be arranged as a c-dimensional matrix and distributed using a c-dimensional cyclic scheme. Assuming n/c even, the DFT in each dimension is computed following the approach A for rings.

Approach B

The hypercube algorithm to compute the FFT has the input sequence distributed in cyclic scheme (element i is allocated to node $i \; mod \; P$). Then, the hypercube algorithm is mapped onto the c-dimensional torus using the general form of the xor embedding, which maps a d-dimensional hypercube onto a $(2^{k1}, 2^{k2}, ..., 2^{kc})$ c-dimensional torus such that $k_1 + k_2 + ... + k_c = d$. It is described below.

Given a positive integer x, let $x(i)$ denote the i-th bit of the binary representation of x. The least significant bit is considered to be the *0th* bit. We also define K_j in the following way. $K_1 = 0$, and for every $1 < j \leq c+1$ we have that:

$$K_j = \sum_{i=1}^{j-1} k_i$$

Let H be the graph which represents the d-cube and T be the graph which represents the torus.Then, vertex n of H is mapped onto vertex $(m_1, m_2, ..., m_c) = f_{xor}(n)$ in T as follows:

$$m_j(i) = n(i+K_j) \qquad i \in [0, k_j - 1], i \neq k_j - 2$$

$$m_j(k_j - 2) = XOR\,(n(K_{j+1} - 1), n(K_{j+1} - 2))$$

Figure 8 shows an example of embedding a 64-node hypercube onto a (8,8) two-dimensional torus.

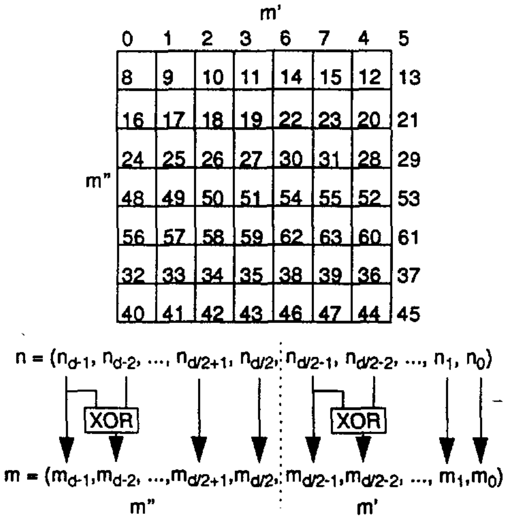

Figure 8: XOR embedding of a 64-node hypercube onto a two dimensional torus.

3.3 A Preliminary Comparison Between Approaches A and B

In general, approach A is expected to outperform approach B since the former uses more efficiently the interconnection network of the torus (both approaches are equally good in terms of load balance). In approach A, communication is required only to transpose a matrix in a ring. Such matrix transposition is performed using at the same time two out of the *2c* links of the tours.

On the other hand, in each communication step of the hypercube algorithm (approach B), every node sends a message to one of its neighbors. Therefore, every node sends a single message along one of its *2c* links in the torus, making a poorer use of the interconnection network than approach A. In the next section a modification of the hypercube algorithm is proposed which allows for a better utilization of the interconnection network.

3.4 Pipelining Hypercube Communication (Approach C)

Pipelining the communication operations can reduce the communication cost of hypercube algorithms. The basic idea is to change the ordering of computations in such a way that every node can send, in each iteration, several messages in parallel, along different dimensions of the hypercube. The communication pipelining technique was used in [7] to improve the efficiency of the FFT computation on the CM-2. In this paper we will use a slightly different scheme for communication pipelining, which takes into account that the hypercube algorithm will be finally executed on a torus.

The communication pipelining technique can be applied to hypercube algorithms in which the code executed by every process *p* has the following structure.

```
do i=0, K-1
  compute $x^p_i$(1..N)
  exchange $x^p_i$ with neighbor in
  dimension $d_i$
enddo
```

where d_i is one of the dimensions of the hypercube ($d_i \in [0,d\text{-}1]$).

The code consists in *K* steps (*K* is equivalent to the dimension of the hypercube algorithm for the particular case of the FFT), each one composed of a computation phase and a communication phase. In the computation phase, *N* data items are computed. These data are represented by vector $x^p_i(1..N)$. After the computation step there is a communication step in which the computed data x^p_i are exchanged with one of the neighbors in the hypercube.

To apply communication pipelining, it is also required that the computation of x_i can be written as follows:

```
do j=1, N
  $x^p_i$(j) = f ($x^q_{i-1}$(j), local_data)
enddo
```

where *p* and *q* are neighbor processes. This means that the computation of $x^p_i(j)$ is a function of $x^q_{i-1}(j)$ (which was computed in step *i-1* by neighbor in dimension d_{i-1}), and possibly some local data.

The idea of communication pipelining is based on the fact that, in order to compute $x^p_i(1)$ it is not necessary to receive the whole vector x^q_{i-1} from the neighbor in dimension d_{i-1} but simply element $x^q_{i-1}(1)$. Therefore, every vector x_i can be decomposed into *B* packets. In a first iteration every node computes the first packet of x_1 and sends the result to neighbor in dimension d_1. In a second iteration, every node computes the second packet of x_1 and the first packet of x_2 (it has all the information required to perform these computations). At the end of this second iteration, each node sends in parallel two messages, one of them to neighbor in dimension d_1 containing the second packet of x_1, and the other one to neighbor in dimension d_2, containing the first packet of x_2. Proceeding in this way, at the end of the third iteration every node can send three messages in parallel. In the steady state, at the end of every iteration, in case of *B>d* each node sends *d* messages of length *N/B* through the *d* different links and in case of *B<d* each node sends *B* messages of length *N/B* through *B* different links. In any way, the communication bandwidth of the hypercube is used better than in approach B.

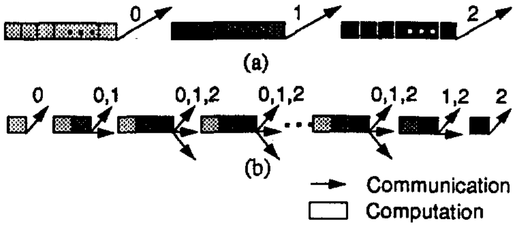

Figure 9: An example of communication pipelining when *B>d* (*d=3*). (a)Hypercube algorithm without communication pipelining. (b) Hypercube algorithm with communication pipelining.

Figure 9 shows the basic idea of communication pipelining on hypercubes when *B>d*. Figure 9.a represents the execution of the hypercube algorithm without communication pipelining for *d*=3. After the computation phase of iteration *i*, there is a exchange of information along dimension *i* of the hypercube. The size of the message sent in every iteration is $N=2^{n-d-1}$ complex numbers. Figure 9.b shows how the same algorithm is executed with communication pipelining. The computation in every iteration is decomposed into *B* blocks. After the computation of the first block of the first computation phase, a first data exchange can be carried out along dimension *0*. The length of the message is *N/B*. In the second iteration of the pipelined algorithm, the second block of the first computation phase and the first block of the second computation phase can be done. At the end of the iteration, two messages of length *N/B* can be sent in parallel along dimensions *0* and *1* of the hypercube. In general, the pipelined algorithm has *d-1* iterations to load the pipeline (load phase). In iteration *i* of the load phase, data is sent in parallel along *i* dimensions of the hypercube. The next *B-d+1* iterations constitute the steady phase. At the end of each iteration of the steady phase, *d* messages are sent in parallel along *d* dimensions of the hypercube. Finally, there are *d-1* iterations to unload the pipeline. In the unload phase, the number of messages sent in parallel decreases in every iteration.

Figure 10 shows the idea of communication pipelining on hypercubes when *B<d*. Figure 10.a represents the execution of the hypercube algorithm without communication pipelining but this time for *d*=6 and *B*=3. Figure 10.b shows how the same algorithm is executed with communication pipelining. Now there are *B-1* iterations to load the pipeline. In iteration *i* of the load phase, data is sent in parallel along *i* dimensions of the hypercube. The next *d-B+1* iterations constitute the steady phase. At the end of each iteration of the steady phase, *B* messages are sent in parallel along *B* different dimensions of the hypercube. Finally, there are *B-1* iterations to unload the pipeline.

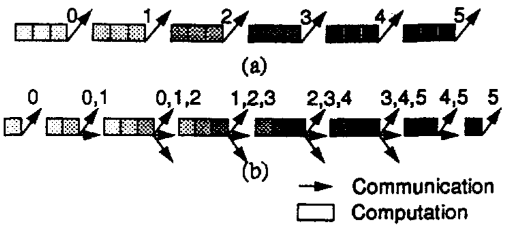

Figure 10: An example of communication pipelining when *B<d* *(d=6)*. (a)Hypercube algorithm without communication pipelining. (b) Hypercube algorithm with communication pipelining.

A pipelined hypercube algorithm can be executed on a c-dimensional torus using the xor embedding. However, every node of the torus will be able to send at most *2c* messages in parallel. Therefore, it does not make any sense that *B > 2c*. On the other hand, for communication pipelining to be effective, every node must have any group of *B* consecutive neighbors in *B* different directions of the tours, so that it is possible to send *B* messages in parallel. This cannot be achieved in the case of a ring since, when using xor embedding, there is always a node which has all its neighbors except one in the same direction of the ring. However, communication pipelining can be used across rings and, therefore, it is useful when *c*>1. In particular, the xor embedding can easily map every group of *c* consecutive neighbors along the *c* different dimensions of the torus. As an example, in figure 8 the xor embedding has mapped the neighbors in dimensions *0*, *1* and *2* along the horizontal dimension of the torus and the neighbors in dimensions *3*, *4* and *5* along the vertical dimension. Through a simple node renaming, neighbors in dimensions *0*, *2* and *4* can be mapped along the horizontal dimension and neighbors in dimensions *1*, *3* and *5* along the vertical dimension. In this way every group of *2* consecutive neighbors are mapped along different dimensions of the torus.

Therefore, we will take *B=c*. The resulting algorithm (xor embedding with communication pipelining) will be called in the following sections approach C. In general, approach C is expected to have a lower communication cost than approach A when *c* is greater than *2*.

4. Evaluation

In this section, the three approaches described in previous sections are evaluated. Only the communication component of the algorithms is evaluated and compared since the computation cost is the same for all three approaches. To carry out the evaluation, an analytical model is developed first for each algorithm. Using these analytical models, some performance figures are given for some particular cases.

To build an analytical model for the cost of the communication component of the algorithms we assume that the time required to send a message consisting in *m* real data items to a neighbor node in the torus is:

$$Tsup + m \times Te$$

where *Tsup* is the start-up time and *Te* is the transmission time per real data item. For sending a message to a non neighbor node, a store-and-forward scheme is assumed.

Again, the case of a ring is considered before analysing the general case.

4.1 Analytical models for the ring

Let $N=2^n$ be the size of the input sequence and $P=2^d$ the number of nodes of the ring. Analytical models for approaches A (two-dimensional index mapping) and B (xor embedding) are developed below. As shown before, approach C (xor embedding with communication pipelining) can only be applied for higher dimensionality tori.

Approach A

The communication component of the algorithm is due to the matrix transposition. This matrix transposition is carried out (as described in section 3.1) in 2^{d-1} steps. In step *i* every node receives two messages containing 2^{d-1}-i+1 and 2^{d-1}-i blocks of data, each block containing $2^{n/2-d}$-by-$2^{n/2-d}$ complex numbers, takes one block from each message and sends the rest of both messages to its neighbors, again in parallel.

Taking into account that, when sending two messages in parallel, only the cost of the longest one must be considered. The communication cost for this algorithms can be modelled as follows:

$$Tcom_A = \sum_{i=1}^{2^{d-1}} [Tsup + (2^{d-1} - i + 1)\,(2^{n-2d+1})Te] =$$
$$= 2^{d-1}Tsup + (2^{n-2} + 2^{n-d-1})Te$$

Approach B

In iteration *i* of the hypercube algorithm, each node sends a message containing 2^{n-d-1} complex numbers to its neighbor along dimension *i* of the hypercube. When using the xor embedding, neighbor along dimension *i* is at a distance 2^i if $i \in [0,d-2]$ or at a distance 2^{d-2} if $i=d-1$. Therefore, the communication cost can be modelled as follows:

$$Tcom_B = \left(\sum_{i=0}^{d-2} 2^i + 2^{d-2}\right)(Tsup + 2^{n-d}Te) =$$
$$= (3 \cdot 2^{d-2} - 1)\,Tsup + (3 \cdot 2^{n-2} - 2^{n-d})\,Te$$

As expected, from the above models it is evident that approach A outperforms approach B. The start-up cost in approach A is *2/3* of the start-up cost in approach B. Moreover, the transmission cost in approach A is over *1/3* of the transmission cost in approach B.

4.2 Analytical models for the general case

The analytical models for the three approaches in the general case are given now. Assume we have to compute a FFT of length $N=2^n$ on a square c-dimensional torus of $P=2^d$ nodes ($2^{d/c}$ nodes in every dimension).

Approach A

As described is section 3.2., data is distributed by following a c-dimensional index mapping and a c-dimensional cyclic data distribution scheme. In each of the *c* steps of the algorithm, FFTs of length $2^{n/c}$ have to be computed on rings of size $2^{d/c}$. In particular, every ring computes $2^{(n-d)(c-1)/c}$ FFTs of length $2^{n/c}$. Each of these FFTs is computed following approach A for rings. However, these FFTs are computed in an interleaved way, that is, after performing steps (a) and (b) of approach A for each FFT (see section 3.1) all the matrices are transposed simultaneously (step (c)). Then, step (d) is done for each FFT. In that way the communication cost of matrix transpositions is minimized since the nodes of the rings exchange long messages containing data of all the FFTs they are computing in every step, amortising the communication start-up cost. Taking into account the analytical model for the computation of a FFT of length 2^n on a ring with 2^d nodes, following approach A, the following general form can be derived:

$$Tcom_A = c\,[2^{d/c-1}Tsup +$$
$$+\,(2^{n/c-2}+2^{(n-d)/c-1})\,2^{(c-1)\,(n-d)/c}Te\,] =$$
$$= c2^{d/c-1}Tsup + c\,(2^{n+d/c-d-2}+2^{n-d-1})\,Te$$

Approach B

When using the general form of the xor embedding, each node of the hypercube has *c* neighbors at a distance 2^i in the torus ($i \in [0,\ d/c-2]$) and *c* more neighbors at a distance $2^{d/c-2}$. Since in each of the *d* communication steps every node sends a message of length 2^{n-d}, the analytical model for the general form is:

$$Tcom_B = c\left(\sum_{i=0}^{d/c-2} 2^i + 2^{d/c-2}\right)(Tsup + 2^{n-d}Te) =$$
$$(3c\cdot 2^{d/c-2}-c)\,(Tsup+2^{n-d}Te) =$$
$$= (3c\cdot 2^{d/c-2}-c)\,Tsup + c\,(3\cdot 2^{n+d/c-d-2}-2^{n-d})\,Te$$

Approach C

The approach described in section 3.4 is considered now. In the general case, the hypercube algorithm is pipelined into *c* stages. The first *c-1* iterations of the algorithm constitute the pipeline load phase, the next *d-c+1* iterations are the steady phase and the last *c-1* iterations are the pipeline unload phase.

In iteration *i* of the pipeline load phase, every node sends in parallel *i* messages of size $2^{(n-d)}/c$. All these messages go to neighbor nodes in the torus. Therefore, the communication cost of the pipeline load phase is:

$$(c-1)\,(Tsup + \frac{2^{n-d}}{c}Te)$$

In every iteration of the steady phase, every node sends *c* messages in parallel, along the *c* dimensions of the torus. The size of every message is $2^{(n-d)}/c$. In this case, some of the messages must travel a longer distance than others. Therefore, we take into account the cost of the message to be sent to the longest distance. The communication cost for the steady phase is:

$$\left(1 + c\sum_{i=1}^{d/c-2} 2^i + c\cdot 2^{d/c-2}\right)(Tsup + \frac{2^{n-d}}{c}Te) =$$
$$= (1 + c\cdot 2^{d/c-1} + c\cdot 2^{d/c-2} - 2c)\,(Tsup + \frac{2^{n-d}}{c}Te)$$

Finally, in the pipeline unload phase, all messages go to a distance $2^{d/c-2}$. The communication cost for this phase is:

$$(c-1)\,2^{d/c-2}\,(Tsup + \frac{2^{n-d}}{c}Te)$$

Putting all together, the communication cost for approach C is:

$$Tcom_C = [\,(4c-1)\,2^{d/c-2} - c - 1]\,(Tsup + \frac{2^{n-d}}{c}Te) =$$
$$= [\,(4c-1)\,2^{d/c-2} - c]\,Tsup +$$
$$+\,(\frac{(4c-1)}{c}2^{n+d/c-d-2} - 2^{n-d})\,Te$$

From the above expressions we can see that approach A is always better than B as expected. For c-dimensional tori with *c =2* or *c=3*, approach C may outperform approach A, in particular when *d* is low (a small number of nodes). When *c > 3*, approach C may outperform approach A for any value of *d*. However, the previous conclusions are affected by the value of *Tsup* and *Te*. When the cost of the start time is increased in relation to the transfer time, approach C is most degraded, especially for small problems (small value of *n*).

4.3 Performance figures

Based upon the previous models, some performance figures are presented in this section. To that purpose, two different scenarios are considered. The first one assumes *Tsup* = 4 µsec and *Te* =5 µsec. These values are close to those of the Transputer T800 processor, which is a very suitable processor for the implementation of tori [6]. Note that, since communication in the T800 is highly optimized, the value of *Tsup* is quite low. On the other hand, the value of *Te* is relatively high since the point-to-point links are serial. The second scenario assumes a higher *Tsup/Te* ratio: *Tsup* = 40 μsec and *Te* =5 μsec (Notice that the relative performance of the different approaches depends only on the ratio *Tsup/Te* and it does not depend on the concrete values that each one of the factors takes).

The performance of the three previously presented approaches is compared in figure 11. This figure shows the communication cost of approaches B and C in relation to the cost of approach A. The main conclusions that can be drawn from this figures are presented in the next section.

5. Conclusions

This paper presents two novel approaches to execute the FFT on a torus multicomputers: a) approach B: a XOR embedding of a hypercube onto a torus and b) approach C: applying communication pipelining to the previous approach. Analytical models of their performance are developed and used to compare them with the standard approach based on a multidimensional index mapping (approach A). The results, which are depicted in figure 11, can be summarized as follows.

Although the three approaches require the movement of the same amount of data among processors, their performance is significantly different.

Despite of the good properties of the XOR embedding [3,4], approach A is more effective than approach B because the former exploits more parallelism in the communication operations and in addition, it reduces the number of messages.

When combining the XOR embedding with communication pipelining (approach C), the start-up cost (term on *Tsup*) increases by a factor of about 2 in relation to approach A while the transmission cost (term on *Te*) experiences a variation by a factor between $1/c - 1/2c^2$ (when d is equal to c) and $4/c - 1/c^2$ (when d is much greater than c). The net effect is that for a two-dimensional torus, the best approach is in general approach A although approach C may be better for a very small number of nodes (d close to c). For a three-dimensional torus, the conclusions are similar, but now, approach C outperforms approach A for a wider range of values of d. However, from c greater than 3, approach C is the most efficient provided that *Tsup* is not extremely higher than *Te*.

Acknowledgments

This work has been supported by the Ministry of Education and Science of Spain (CICYT TIC-92/880 and TIC-91/1036)

References

[1] R.E. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley Publishing Company 1985.

[2] J.C. Cooley and J.W. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series", *Math. Comp*, 19:291-301, 1965.

[3] A. González and M. Valero-García, "The Xor Embedding: An Embedding of Hypercubes onto Rings and Toruses", *Proc. Int'l Conf. on Application Specific Array Processors*, pp. 15-28, 1993.

[4] A. González, M. Valero-García and L. Díaz de Cerio, "Executing Algorithms with Hypercube Topology on Torus Multicomputers", *IEEE Trans. on Parallel and Distributed Systems*, to appear.

[5] A. Gupta et al., "Comparative Evaluation of Latency Reducing and Tolerating Techniques", *Int'l Symp. on Computer Architecture*, p. 254, 1991.

[6] M. Homewood et al., "The IMS T800 Transputer", *IEEE Micro*, Vol. 7, No. 5, pp. 10-26, Oct. 1987.

[7] S.L. Johnsson, M. Jacquemin and R.L. Krawitz, "Communication Efficient Multiprocessor FFT", *Thinking Machine Corp. Technical Report*, TR-220, Oct. 1991.

[8] A. H. Karp, "Programming for Parallelism", *IEEE Computer*, p. 43, May 1987.

[9] T.H. Lai and A.P. Sprague, "Placement of the Processors of a Hypercube", *IEEE Trans. on Computers*, Vol. 40, No. 6, pp. 714-722, 1991.

[10] D. D. Loveman, "High Performance Fortran", *IEEE Parallel & Distributed Technology*, p. 25, February 1993.

[11] S. Matic, "Emulation of Hypercube Architecture on Nearest-Neighbor Mesh-Connected Processing Elements", *IEEE Trans. on Computers*, Vol. 39, No. 5, pp. 698-700, May 1990.

[12] V.S. Sunderam, G.A. Geist, J. Dongarra and R. Manchek, "The PVM Concurrent Computing System: Evolution, Experiences, and Trends", *Parallel Computing*, Vol. 20, No. 4, pp. 531-546, 1994.

[13] C. Tong and P.N. Swarztrauber, "Ordered Fast Fourier Transform on a Massively Parallel Hypercube Multiprocessor", *Journal of Parallel and Distributed Computing*, Vol. 12, pp. 50-59,1991.

[14] D.W. Walker, "The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers", *Parallel Computing* Vol. 20, No. 4, pp. 657-674, 1994.

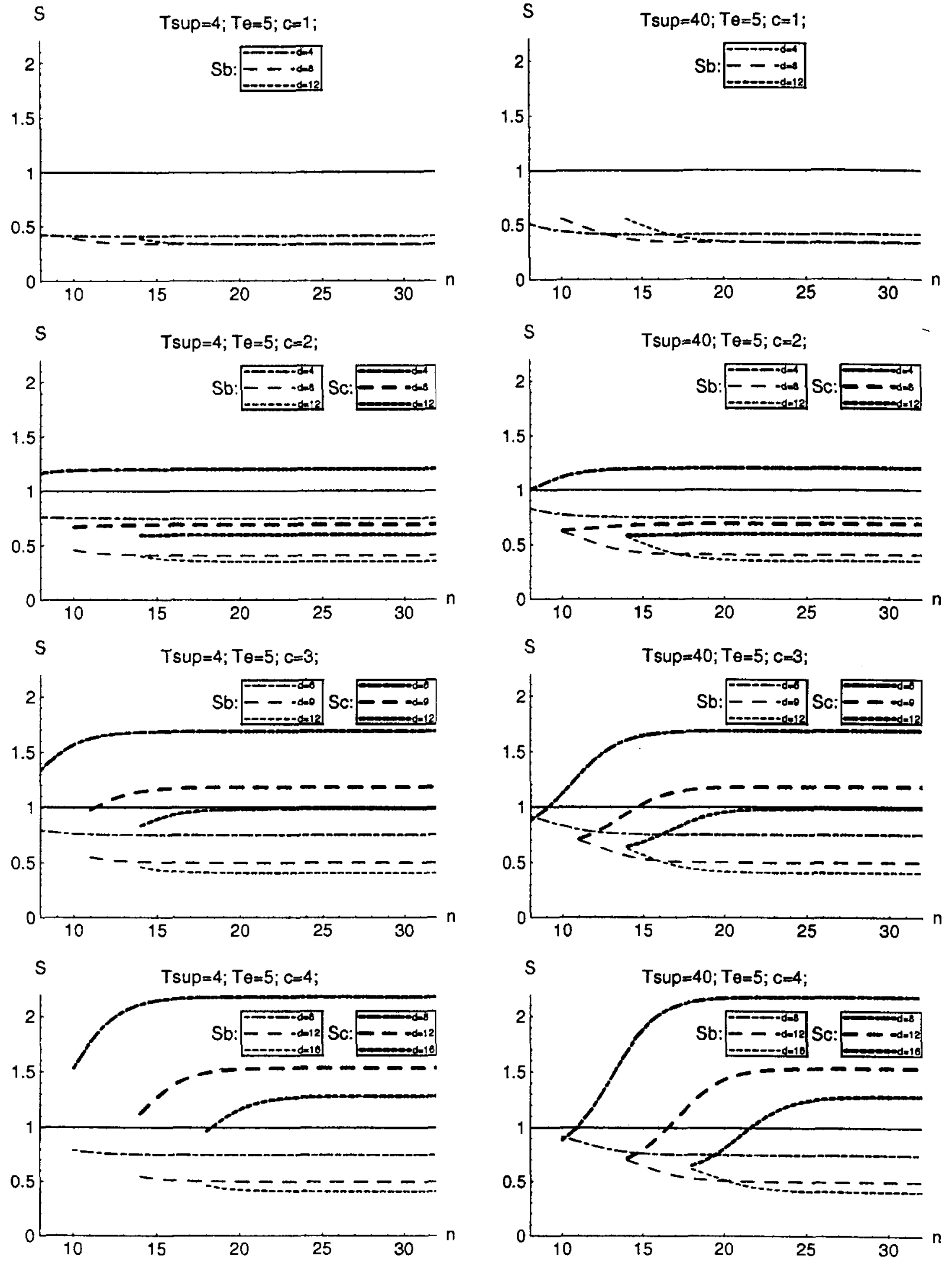

Figure 11: Performance figures. S depicts the communication cost of approaches B (curves labelled with Sb) and C (curves labelled with Sc) divided by the communication cost of approach A.