Computación y Sistemas Vol. 4 No.4 pp. 298 - 305
© 2001, CIC - IPN. *ISSN 1405-5546* Impreso en México

CALMANT: A Systematic Method for the Execution of Hypercube Algorithms in Multiprocessor Systems

Luis Díaz de Cerio, Miguel Valero García, Antonio González and **Dolors Royo**
Departament d' Arquitectura de Computadors
Universitat Politècnica de Catalunya
C/Jordi Girona 1-3 Mòdul D6, 08034-Barcelona, Spain
Telf: +34 93 4017188, Fax: +34 93 4017055
e-mail: {ldiaz, miguel, antonio, dolors}@ac.upc.es

Article received on March 6, 2000; accepted on February 21, 2001

Abstract

In this work we present the CALMANT (CC-cube Algorithms on Meshes and Tori) method as a systematic method for the execution of a certain type of algorithms that are denominated CC-cube algorithms, on meshes and tori of several dimensions. It is very frequent to find CC-cube algorithms in literature (FFT, complete exchange, some methods for the single value decomposition and eigenvalue computation, etc) but the direct application of these algorithms does not allow to exploit efficiently the bandwidth that the interconnection network offers in meshes and tori. The CALMANT method allows us to reorganize the computation and communication of the CC-cube algorithms so that the efficiency increases remarkably. The importance of this method not only lies in the improvement of the efficiency but also it can be applied in a systematic way on different types of architectures.

Keywords: CC-cube algorithm, Communication Pipelining, Embedding, Hypercube, Mesh and Tori.

1 Introduction

One of the most important goals at the time of programming a multicomputer is the efficient use of the existent interconnection network between the nodes that compose the multicomputer. Depending on the interconnection network, the efficiency as far as the execution time of a given algorithm can be very different from one multicomputer to another.

The principal characteristics of an interconnection network are, among others, the topology, the number of ports that the nodes dispose of and the communication model used. In order to be able to make an efficient use of the interconnection network, it is necessary to consider these three characteristics.

Our work is focused concretely to interconnection networks with hypercube, mesh or tori topology of multiple dimensions. In anyone of the cases we have studied architectures with one port and with a maximum number of ports (all ports). With regard to the communication models, we have considered: *store-and-forward, wormhole, circuit-switching and virtual cut-through.*

Frequently, the algorithms designed for a determined multicomputer model cannot be executed (or they are not efficient) in a different multicomputer model. This work presents the CALMANT method as a systematic method to execute a determined type of algorithms on the different multicomputer models mentioned before, maintaining in anyone of the cases a high efficiency. On the other hand, even if it where at the reach of this work, the proposed method can be easily extensible to other types of architectures not so common but that can be interesting in the future. This is the case, for example, of meshes and tori of four or more dimensions and meshes and tori of six or eight neighbors.

The type of algorithms to which the work is focused is the one that are denominated as algorithms of *Computation and Communication with Hypercube topology (CC-cube)*. There are many algorithms that can be classified within this type. This is the case, for example, of the FFT, the Harley transformation, complete exchange, single value decomposition and eigenvalue computation, etc. From here comes the importance that a systematic method has like the one we propose, to be able to program these algorithms on multicomputers with different interconnection networks maintaining a good efficiency.

The efficiency of an algorithm not only depends on the characteristics of the interconnection network for which the algorithm has been designed, but that also depends on the relation that exists between the parameters of the architecture (i.e., computation time, transmission time) and the problem parameters (i.e., the problem size, number of operations per size unit). The CALMANT method also considers this and gives as a result algorithms that adapt to the relation between these parameters. For it, the work done has an important modeling component of the execution time of the algorithms.

The obtained execution time models allow us in each case to make decisions that optimize the efficiency of these algorithms.

The objective of this work is to make a presentation of the CALMANT method and to give some results. Concretely, in section 2 we will make a general vision of the method. In section 3 the architectures and the algorithms towards which this work is focused are described. The concept of communication pipelining is introduced in section 4. The embeddings that we will use are described in section 5. In section 6 we analyze how the scheduling of messages are carried out in the resulting algorithms and finally in section 7 we will show some efficiency figures.

2 General Description

The CALMANT method is based on the combination of two concepts clearly differentiated: the *communication pipelining* and the *embedding* of a CC-cube on hypercubes, meshes and tori.

The CC-cube algorithms are composed of 2^d processes, so that if we represent each process by a point and join by means of a line any pair of processes that communicate between each other (neighboring processes), the result is a hypercube.

The execution of these algorithms consists of a determined number of iterations. In each of the iterations, the processes perform a computation phase and another of communication (not necessarily in this order). The principal characteristic of these algorithms is that in each of the iterations all the processes communicate by a same hypercube dimension. This way, supposing that a connection exists between any pair of neighboring processes, these algorithms would only use $1/d$ of the total communication capacity, where d is the hypercube dimension. This is what would happen, for example, if we map a CC-cube on a multicomputer with hypercube topology.

By means of the communication pipelining technique we will obtain, until the point that is possible, to use all the hypercube dimensions simultaneously. This way we will be able to practically use the total communication capacity that can exist between the CC-cube processes.

As figure 1 shows, the idea is very simple. It deals with reorganizing the algorithm so that, instead of using the hypercube dimensions in sequential form, we can use them in parallel. The figure shows the hypercube dimension by means of a thicker outline that is used at every instant of time.

The resulting algorithm is denominated *pipelined CC-cube* algorithm.

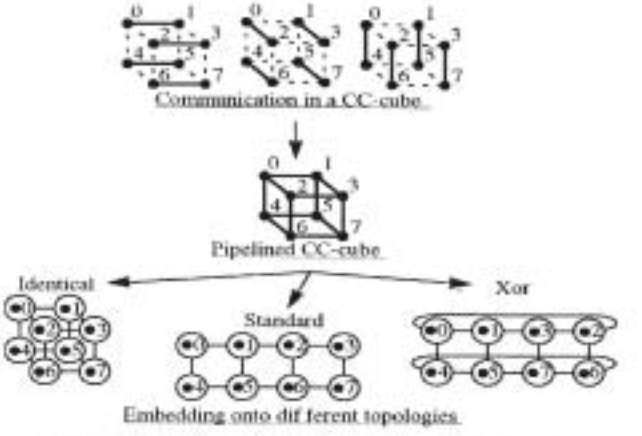

Figure 1: Basic elements of the CALMANT method: communication pipelining and embedding of a pipelined CC-cube algorithm on different topologies

It is not always possible to have a direct connection between all the pairs of neighboring processes. For example, in a mesh or tori, each node only has (at the most) four neighboring nodes. Therefore, if we want to execute a CC-cube algorithm whose dimension is d>4 it will be impossible that all the neighboring processes be located in neighboring nodes.

The embedding of a CC-cube on a mesh or torus is a function that allows us to map the pipelined CC-cube processes on the hypercube, mesh or torus nodes. The embeddings that we propose make this mapping so that, any pair of neighboring processes according to a same hypercube dimension are mapped in nodes located at the same distance in the host architecture. In addition, given a process located in a determined node, their neighboring processes will be mapped in nodes located the closest possible to this node, that is to say, that the existing average distance between a process and its neighbors is minimum.

As figure 1 shows, the embedding that we will use to perform the mapping of a CC-cube algorithm on a hypercube is the embedding that we denominated *identity*, since it deals with making the embedding of two identical graphs. For the mapping of CC-cubes on meshes and tori we will use the embeddings *standard* and *xor* respectively. These embeddings will be described later in more detail.

3 Considered Architectures and Algorithms

The CALMANT method can be applied on architectures of diverse topology, dimensionality, communication model and number of ports. Concretely, this work is focused to hypercubes, meshes and tori of two and three dimensions with bidirectional full-duplex communication under the models of a port (one port) and the maximum number of ports (all ports).

The difference between the one port model and the maximum number of ports model lies in that under the first model the nodes cannot send more than one message simultaneously, however, under the second model the nodes can send simultaneously as many messages as connections the node has. The communication models that we will consider are *store-and-forward* and *wormhole* (the results obtained for wormhole can be extended of equivalent form to *circuit-switching* and *virtual cut-through*.

With regard to the algorithms, we will focus in a determined type of algorithms that we will denominate *CC-cube* algorithms. A CC-cube is an algorithm with hypercube topology formed by 2^d processes, so that the code executed by each process has the following structure:

```
do i = 1, K
    Compute x[1:N] and other possible
          local data.
    Exchange x with the neighbor
          in dimension d_i.
enddo
```

where d_i is one of the hypercube dimensions ($d_i \in [0, d-1]$) and where d_i is not necessarily different to d_{i-1}.

The code consists of K iterations. Each one of these iterations is formed by a computation phase and a communication phase. In the computation phase the N data is computed. These data are represented by vector $x[1:N]$. After the computation phase, during the communication phase vector x is exchanged with one of the neighbors in the hypercube. The x vectors are local variables, that is to say, they are different for each process. In each computation phase, data that are not involved in the communication phase can also be computed. Note that in each iteration of the previous algorithm, the processes send only one message of length N to one of their neighbors.

4 Communication Pipelining

In order to explain the communication pipelining technique and to make a first qualitative evaluation of the improvement that contributes to the efficiency of the algorithms, we will assume in this section that the CC-cube is executed on a hypercube. We choose the hypercube for the sake of simplicity; nevertheless, the technique can be applied on meshes and tori as it will be seen later.

The direct application of a CC-cube algorithm on a computer with hypercube topology only uses a dimension in each iteration and the rest of dimensions remain inactive. In addition, because the message sent in an iteration is necessary to start the following iteration, there is no way of exploiting the possibility of overlapping computation and communications.

In figure 2.a we can observe the direct application of a CC-cube algorithm, with $K=d$, where $d=0$, $d=1$, $d=2$, for the case of a 3-dimensional hypercube. The algorithm consists of three iterations. In each iteration a computation phase is made (indicated with a different tone of gray). Once the computation phase is finished, a communication phase by a different dimension of the hypercube is made (indicated in the figure with a thicker outline).

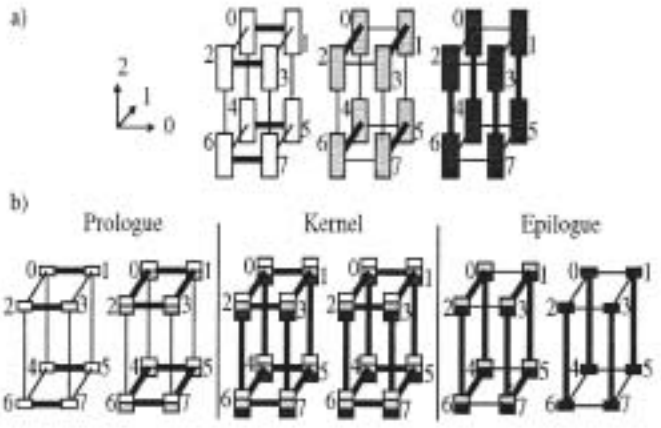

Figure 2: Execution of a CC-cube algorithm. (a) Without communication pipelining. (b) Applying the communication pipelining.

In order to reduce the communication cost we will use the *communication pipelining* technique. This technique is based on the idea of software pipelining proposed by Lamm (1988) and that was used first by Johnsson and Krawitz (1992). This technique allows us to reorganize the CC-cube so that instead of sending in each iteration a big message by a connection, many small messages are sent simultaneously by many connections. In order to this technique can be applied, the computation of each one of the vectors x for the CC-cube

algorithm must satisfy the following dependences:

do $j = 1, N$
  $x[j] = f(x[b], \text{datos\_locales})$
    with $a=i$, $a+b=j+i-1$
enddo

This means that the computation of $x[j]$ depends on only one part of the vectors received in previous iterations and possibly on some local data. The initial value of the vector for each process is x^0.

Figure 2.b shows how this technique for the case of a 3-dimensional hypercube is applied. The vectors on which the computation phase of each iteration is made have been divided in a determined number Q of packets (4 in this case). If the CC-cube algorithm fulfills the property mentioned previously, it is not necessary to completely finish the computation phase of one of the iterations to begin the following computation phase. Therefore, in the first iteration of the new algorithm, the first packet of x^1 is computed and is sent by dimension 0. In the second iteration, the second packet x^1 and the first packet x^2 can be computed since we dispose of the necessary data, which were computed in the previous iteration. Once the computation is finished, the second packet of x^1 is sent by dimension 0 and simultaneously the first packet x^2 is sent by dimension 1. Thus it is continued successively as shown in figure 2.b.

If Q is greater than the number of hypercube dimensions, from the iteration d of the pipelined algorithm and during Q-d+1 iterations we will be able to use the d hypercube dimensions simultaneously. The set of iterations in which the d hypercube dimensions are used in a simultaneous way will form what we call the *kernel*. The previous d-1 iterations to the kernel will form the *prologue* and the following d-1 iterations to the kernel phase will form the *epilogue*. On the other hand if Q is less than the number of hypercube dimensions, the prologue will be formed by Q-1 iterations, the kernel will be formed by d-Q+1 iterations, where we will be able to use Q hypercube dimensions simultaneously and the epilogue will also be formed by Q-1 iterations. The resulting algorithm of the application of the communication pipelining technique to a CC-cube will be denominated *pipelined CC-cube* (Díaz de Cerio *et al.*, 1996)

The communication pipelining technique also allows us to exploit the possibility of overlapping computation and communications. Note that in the iterations of the pipelined CC-cube algorithm several packets are computed and that once the computation of a packet is finished this one can begin to be transmitted at the same time that the following packet is being computed (Díaz de Cerio *et al.*, 1998).

5 Embeddings

The problem of programming a CC-cube algorithm in a mesh or tori can be seen as an embedding problem of graphs, being the algorithm the guest graph and the multicomputer the host graph. In this work we are interested in those embeddings in which is fulfilled that all the processes have their respective neighbors according to a determined hypercube dimension mapped in the multicomputer to the same distance. We will denominate this property as *constant distance* property.

Embeddings with constant distances have the property of which all the processes take the same time to transmit a message to their neighbor in a determined iteration of the algorithm. Due to this and that the duration of the computation phases is identical for each process, the delay times associated to imbalances between the nodes in the communication phases are avoided, especially under the store-and-forward communication model where the communication time is proportional to the distance.

The architectures in which this work is focused are: hypercubes, meshes and tori. The embeddings that represents the execution of the CC-cube on a hypercube is the *identity* embedding, since both graphs are identical. In meshes, the embedding that we will use is the one known as *standard* (Matic, 1990). In the case of tori, the embedding that we will use is denominated as *xor* embedding (González *et al.*, 1995), which is one of the proposals of this work, since in the case of tori, the xor embedding is better than the standard in the sense that the average distance between neighboring processes is smaller.

The application of the standard embedding of a d-dimensional hypercube on a c-dimensional mesh can be defined as figure 3 shows. Matic makes an exhaustive evaluation of this embedding.

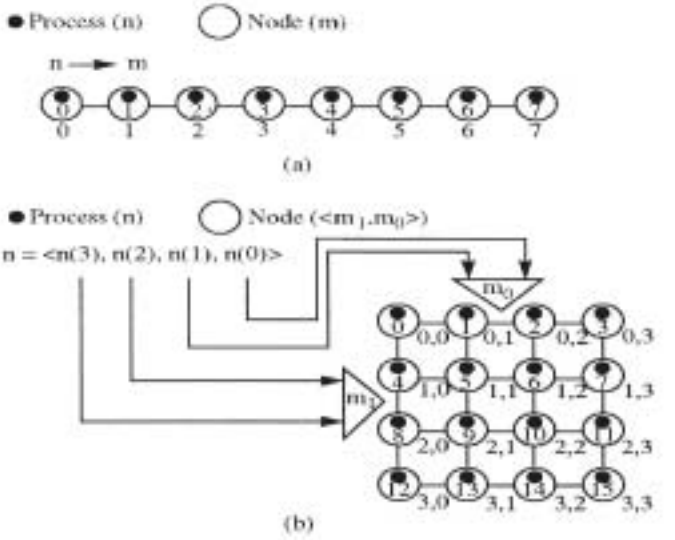

Figure 3: Standard embedding. (a) 3 dimension hypercube on a line. (b) 4 dimension hypercube on a mesh of 4x4 nodes.

The standard embedding fulfills the property of the constant distances. If we look at figure 3.b we can observe, for example, that any pair of neighboring processes according to dimension 1 is found at distance 2 (0-2, 1-3, 4-6, ...). It is necessary to emphasize that, so and as González *et al.* (1995) demonstrates, the standard embedding is optimal in the sense that it minimizes the average distance between a process and its neighbors.

The application of the xor embedding of a d-dimensional hypercube on a c-dimensional tori can be defined as figures 4 and 5 show, where (a XOR b) is the exclusive OR of bits a and b.

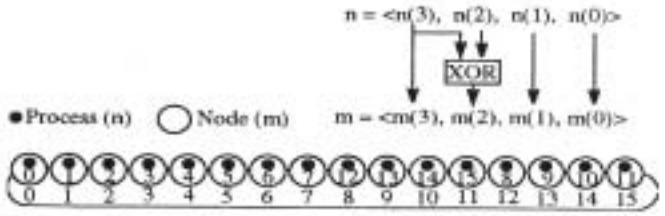

Figure 4: Xor embedding of a 4-dimensional CC-cube on a ring. The processes are labeled by means of n and the nodes by means of m.

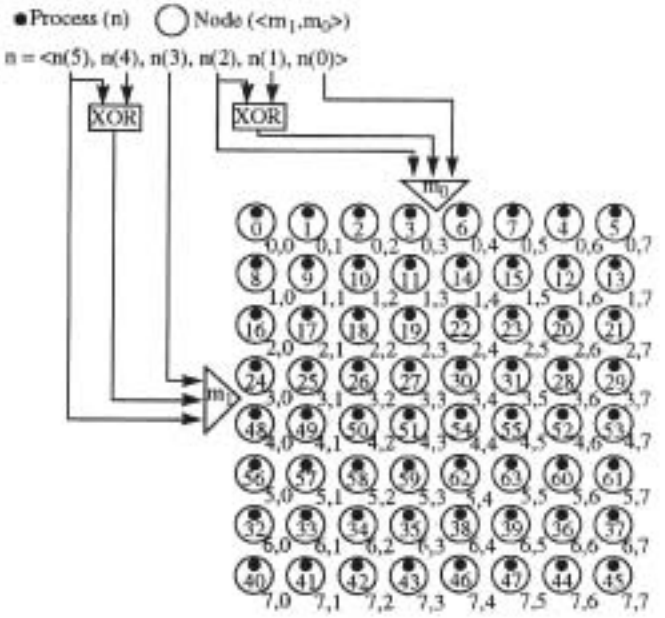

Figure 5: Xor embedding of a 6-dimensional hypercube on a torus of 8x8 nodes. The connections are not shown for the sake of clarity.

The xor embedding also fulfills the property of the constant distances. As it can be seen in figure 5, all the pairs of neighboring processes according to dimension 2 are found at distance 2 (0-4, 1-5, 2-6, ...). Note that, in the case of the standard embedding, the pairs of processes according to dimension 2 will be found at distance 4. González *et al.* (1995) also demonstrates that the xor embedding is optimal in the sense that it minimizes the average distance for the case of rings (tori of a dimension).

6 Message Scheduling

The embedding solely determines the location of the processes of the pipelined CC-cube algorithm in the mesh or tori. In order to completely specify the algorithm, it is necessary to determine how the messages that will be exchanged in each one of the iterations must be scheduled. It is in this point where aspects such as the number of ports, the communication model and the possibilities of overlapping the computation and communications acquire special relevance.

In our work we have made an exhaustive study of how to solve the scheduling problem in each one of the possible scenarios (mesh/tori, one port/maximum number of ports, etc) In general, the proposed schemes tend to make optimal use of the communication bandwidth that the connection network offers. In this section we describe the basic ideas through simple examples that allow us to give an idea of how to approach the problem. For it, we will suppose the wormhole communication model on one-port nodes. Because the message schedulings that we propose are free of conflicts, the examples can be applied in the same way under circuit switching or virtual cut-through communication models.

6.1 Embedding of a CC-cube

Figure 6.a shows how the standard embedding of a CC-cube algorithm has been made on an 8 node line. In this figure it can be seen that to communicate all and each one of the processes with its neighbor a distance 2^i, 2^i communication steps are necessary. In general, if we only consider the transmission time and suppose that the size of the messages is N, we can compute the communication time of the algorithm as:

$$t_1 = \sum_{i=0}^{d-1} 2^i NT_e = \left(2^d - 1\right) NT_e \approx 2^d NT_e \qquad (1)$$

Where T_e is the transmission time for each element of the message.

6.2 Embedding of a Pipelined CC-cube

We will again use the standard embedding to execute a pipelined CC-cube algorithm on a line. Let us take for example any iteration from the kernel of the pipelined CC-cube algorithm. At the end of the iteration all and each one of the

processes send a message to each one of their three neighbors in the hypercube. According to figure 6.b shown, each process has its neighbors a distance 1, 2 and 4. The vertical axis of the figure represents the time. Each one of the arrows represents the communication between two nodes whose processes are neighbors and the different tone of gray of the arrows indicates the dimension of the CC-cube that is used in the communication. The figure shows that during the first two communication steps there is a set of nodes that make the communication by dimensions 1 and 2 and in the following two steps the nodes that remained until now inactive make the communication also by dimensions 1 and 2. Once that all the nodes have made the communication by these two dimensions, the communication begins by dimension 0. In the case of dimension 0, only a step is necessary so that all the nodes make the communication. In the case of any number of dimensions we would take them in pairs beginning by those of greater distance and in descending order. Note that in the figure, in any communication step, most of the communications are passed through two messages (one in each direction) that is to say, that the interconnection network utilization is much more efficient. In general, to communicate all the processes of a d-dimensional CC-cube with their respective neighbors in dimensions i, $i+1$, ..., $i+M-1$, $(2^{i+M+1}-2^{i+1})/3$ communication steps will be necessary if M is even or $(2^{i+M+1}-2^{i})/3$ communication steps if M is odd.

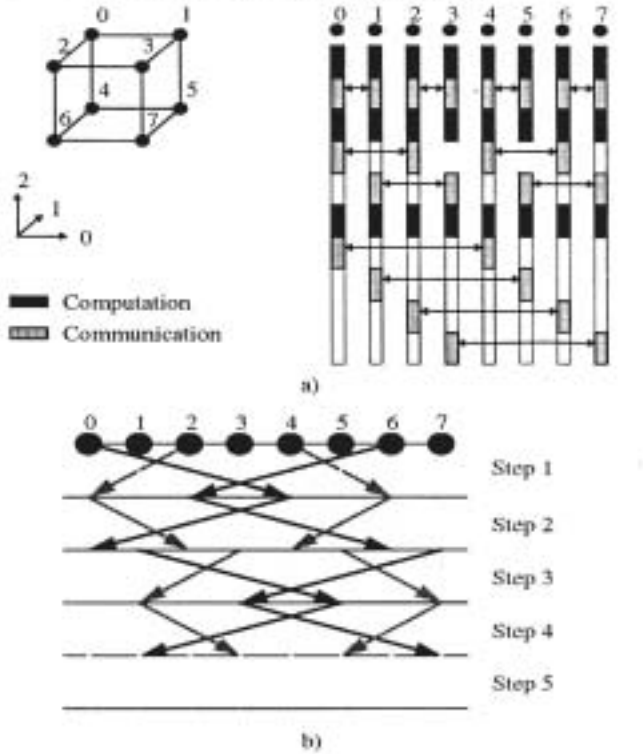

Figure 6: a) Computation and communication of a CC-cube on a line of 8 nodes. b) Execution of a kernel iteration of a pipelined CC-cube algorithm on a line.

If we account for the communication steps of the prologue, the kernel and the epilogue and suppose that $Q<d$, these altogether result to be:

$$\frac{(3Q+1)2^{d+2}-2^{d-Q+2}-9z}{18} \qquad (2)$$

where $z=Q$ if Q is even and $z=(Q+1)$ if Q is odd.

If we suppose that $Q=d$, the total number of communication steps are:

$$\frac{(3Q+1)2^{d+2}+3d-12Q-z}{18} \qquad (3)$$

where $z=4$ if d is even and $z=5$ if d is odd.

Now supposing that Q is sufficiently big $(Q>> d)$, we can suppose that the cost due to the prologue and the epilogue can be neglected with respect to the kernel cost. Considering only the transmission time and knowing that the size of the messages is N/Q we can express the communication time of the pipelined CC-cube algorithm as:

$$t_2 \approx (Q-d+1)\frac{2^{d+1}N}{3Q}T_e \approx \frac{2^{d+1}}{3}NT_e \qquad (4)$$

If we compare this result with the one previously obtained for the case of the CC-cube algorithm without communication pipelining, it results that we can express the improvement of the efficiency obtained when applying the CALMANT method as $r=t_1/t_2=1.5$, which means an important reduction in communication time. In general, for meshes of c dimensions the result is an efficiency improvement of order $r=1.5^c$.

The bigger parameter Q is, the better use is made of the interconnection network, however when increasing the number of packets to transmit, the cost due to the initialization of the messages also increases. A tradeoff exists that will have to be solved by means of an optimal Q selection and so minimizing the communication cost.

The example presented here is a simplified case that shows us how the message scheduling has an important relevance at the time of specifying the algorithms. The complete study of message scheduling for meshes and tori of multiple dimensions is much more extensive and can be found in the doctoral thesis of Díaz de Cerio (1998).

7 Efficiency Figures

The CALMANT method has been applied to problems such as the *FFT*, the *complete exchange* and some methods for the *single value decomposition and eigenvalue computation*, all of them on different types of architecture (Díaz de Cerio *et al.*, 1995, 1996, 1998; Royo *et al.*, 1998). In previous works, the CALMANT methodology has been presented in an initial version for tori, without applying the communication pipelining. Later, the communication pipelining technique was applied to synchronous and asynchronous hypercubes (the last ones with the possibility of overlapping computation and communication). In this section we will present some of the main contributions, related to the CALMANT application to meshes, making use of the communication pipelining technique. The graphs are obtained from analytical models of the execution time.

In the first place we will consider the application of the CALMANT method of the FFT. The efficiency of our method has been compared with the proposed method by Aykanat and Dervis (1991) for the resolution of the FFT on a hypercube of dimension d=10 with capacity to overlap computation and communications. The graph of figure 7.a shows the efficiency of both methods with respect to the execution time of the CC-cube algorithm based on the size of the problem (2^n). In the horizontal axis, the value of n is presented and in the vertical axis the efficiency that results from dividing the time of the CC-cube algorithm (T_{CC}) by the time of the different proposals ($T_{CALMANT}$, $T_{Aykanat-Dervis}$) is presented. In addition, the graph presents a level that cannot be outperformed ($T_{CC}/T_{Optimal}$) to show the distance to the optimal efficiency. We suppose an initialization time of the messages (T_{cs}) 10 times superior to the transmission time by the element and we suppose that this time is at the same time 100 times the computation time (T_c) of an arithmetic operation (with the purpose of obtaining a relation of computation-communication overlap). The proposal of Aykanat and Dervis, besides being a specific proposal for the FFT, it exploits the overlapping but not the communication pipelining. It is for this reason that our method results to be more efficient.

Interesting results have also been obtained when applying the CALMANT method to the complete exchange communication algorithm on meshes and tori. In figure 7.b we can find a graph for three-dimensional tori where our proposal is compared with the proposals of Takkella and Seidel (1994) and by Tseng y Gupta (1996). The graph shows the efficiency of the resulting algorithms based on the volume of data (2^m) that each node sends to another one. Again, in the horizontal axis the problem size (m) is presented and in the vertical axis the efficiency appears, as the execution time of the CC-cube algorithm (T_{CC}) over the time of the different proposals ($T_{CALMANT}$, $T_{Takkella}$, $T_{Tseng-Gupta}$). In this case we are supposing a relation of 1000 between the initialization time of the messages (T_{cs}) and the transmission time of an element (T_{tr}). It is necessary to emphasize that the application of our proposal is made in a systematic way on different architectures and obtains, in many occasions, better results than the specific proposals for concrete architectures that other authors do.

Finally, the proposed method has also been applied to the Jacobi algorithms for the single value decomposition and eigenvalue computation. Figure 7.c shows the relative communication cost when the CALMANT method is applied (*pipelined BR*) and when we apply this method on a modification of the algorithm also proposed by our group (*alpha-optimal*). In this graph two other curves appear that represent the Jacobi algorithm without communication pipelining (*BR algorithm*) and a minimum level (*perfect parallelism*). All of them based on the hypercube dimensionality.

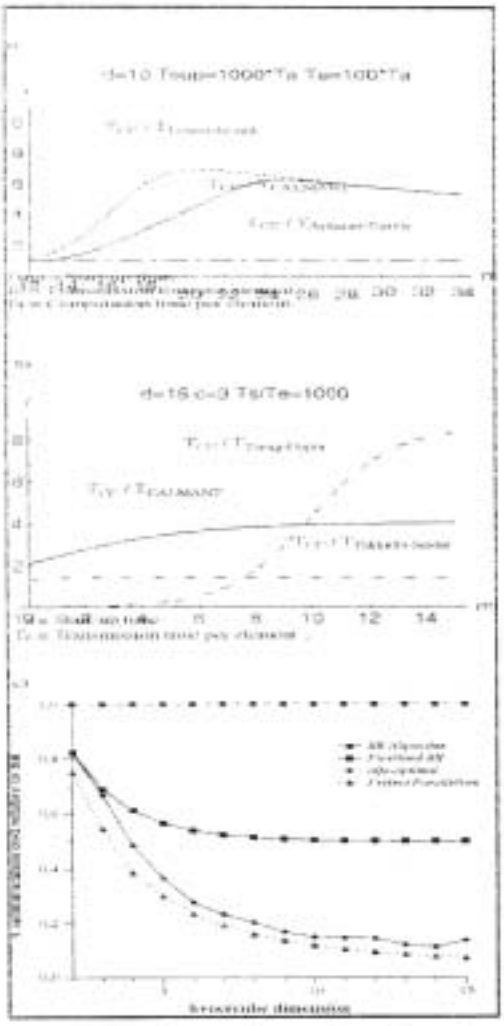

Figure 7: a) FFT algorithm on a hypercube. b) Complete exchange algorithm on a tori. c) BR algorithm on a hypercube

8 Conclusions

In this work the CALMANT method has been presented for the systematic mapping of algorithms with hypercube topology on multiprocessor systems. As an applicability example of the method and the obtained results, we have considered three different problems (*FFT, complete exchange* and *single value decomposition and eigenvalue computation*).

When comparing with the proposals of other authors for the solution of these problems, it can be verified that the CALMANT offers better results for many cases. We must not forget that our proposal is general and applicable to many different problems and architectures, and that the proposals from the other authors are specific for a problem and a concrete type of architecture. Throughout the work, we have tried to give an intuitive idea, avoiding to the maximum possible great formulas in the obtainment of the analytical models of the execution time. For those readers interested in more details, a complete study can be found at Díaz de Cerio's thesis (1998).

Acknowledgements

This work has been supported by the Ministry of Education and Science of Spain (CICYT TIC-98/0511) and the European Center of Parallelism in Barcelona (CEPBA).

References

Ayanat C. and Dervis A., "An Overlapped FFT Algorithm for Hypercube Multicomputers", in proceedings of *International Conference of Parallel Processing*, 1991, pp. II-316-III-317.

Díaz de Cerio L., "CALMANT: Un Método Sistemático para la Ejecución de Algoritmos con Topología Hipercubo en Multicomputadores", Tesis doctoral, ftp://ftp.ac.upc.es/pub/reports/DAC/2000/UPC-DAC-2000/2000-13. ps. Z, Diciembre, 1998.

Díaz de Cerio L., González A., and **Valero-García M.**, "Communication Pipeing in Hypercubes" *Parallel Processing Letters*, Vol. 6, No. 4, December, 1996, pp. 507-523.

Díaz de Cerio L., Valero-García M., and **González A.**, "A Study of the Communication Cost of the FFT on Torus Multicomputers", in *Proceedings of the IEEE First International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, April, pp. 131-140.

Díaz de Cerio L., Valero-García M., and **González A.**, "A Method for Exploiting Communication/Computation Overlap in Hypercubes", Parallel Computing, Vol. 24, No. 2, June, 1998, pp.221-245.

Royo D., González A., and **Valero-García M.**, "Jacobi Orderings for Multi-Port Hypercubes", in proceedings of the *12th. Int. Parallel Processing Symposium (IPPS'98) and 9th Symposium on Parallel and Distributed Processing*, March, 1998, pp. 88-98.

González A., Valero-García M., and **Díaz de Cerio L.**, "Executing Algorithms with Hypercube Topology on Torus Multicomputers", IEEE Trans. on *Parallel and Distributed Systems*, Vol. 6, No. 8, August 1995, pp. 803-814.

Johnsson S.L. and **Krawitz R. L.**, "Cooley-Tukey FFT on the Connection Machine", *Parallel Computing*, 18, 1992, pp. 1201-1221.

Lam M., "Software Pipeling: An Effective Scheduling Technique for VLIW machines", in proceedings of the *Conf. on Progremming Language Design and Impiementation*, June, 1988, pp. 318-328.

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

Takkella S. and **Seidel S.**, "Complete Exchange and Broadcast Algorithms for Meshes", in proceesings of the *IEEE Scalable High Performance Computing Conf.*, 1994, pp. 422-428.

Tseng Y. and **Gupta S.K.S.**, "All-to-All Personalized Communication in a Wormhole-Routed Torus", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No.5, May, 1996, pp. 498-505.

Luis Díaz de Cerio *received the engineering degree in telecommunications in 1993 and the Ph.D. degree in telecommunication engineering in 1998, both from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is currently an assistant professor in the Computer Architecture Department at UPC. His research interests focus on parallel algorithms for multiprocessor systems, distributed computing systems, the Grid and design techniques for low power microprocessors.*

Miguel Valero-García *received the engineering degree in Computer Science in 1986 and the Ph.D. degree in Computer Science in 1989, both from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at UPC. His research interests focus on parallel algorithms for multiprocessor systems and new techniques to improve the academic quality at university.*

Antonio González *received the engineering degree in Computer Science in 1986 and the Ph.D. degree in computer Science in 1989, both from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at UPC. His research interests focus on parallel algorithms for multiprocessor systems, memory organization, speculative execution and design techniques for low power microprocessors.*

Dolors Royo *received the engineering degree in Computer Science in 1989 and the Ph.D. degree in Computer Science in 1999, both from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. She is currently an associate professor in the Computer Architecture Department at UPC. His research interests focus on parallel algorithms for multiprocessor systems, distributed computing systems and the Grid.*