Overlapping Communication and Computation in Hypercubes*

Luis Díaz de Cerio, Miguel Valero-García and Antonio González
Dept. d'Arquitectura de Computadors - Univ. Polit. de Catalunya
c/ Gran Capitán s/n, Campus Nord - D6, E-08071 Barcelona, Spain
E-mail: {ldiaz,miguel,antonio}@ac.upc.es

Abstract. This paper presents a method to derive efficient algorithms for hypercubes. The method exploits two features of the underlying hardware: a) the parallelism provided by the multiple communication links of each node and b) the possibility of overlapping computations and communications, which is a feature of machines supporting an asynchronous communication protocol. The method can be applied to a generic class of hypercube algorithms. Many examples of this class of algorithms are found in the literature for different problems. The paper shows the efficiency of the method using two of these problems as an example: FFT and Vector Add. The results show that the reduction in communication overhead is very significant in many cases and the algorithms produced by our method are always very close to the optimum in terms of execution time.

1 Introduction

Hypercube multicomputers are interesting because there are many problems for which parallel algorithms with a hypercube communication topology are obtained in a natural way [7], [14].

Communication overhead is a crucial issue when considering the performance of a multicomputer. In some cases, communication overhead is the most significant factor in the execution time. To help in reducing communication overhead a) the nodes of a multicomputer can send several messages in parallel through different links (communication parallelism), and/or b) communication through one or several links can be overlapped with computation in the node (communication/computation overlapping)

Designing parallel algorithms which are able to exploit features (a) and/or (b) is not an easy task. Many of those natural hypercube algorithms found in [7], [14] cannot exploit these features efficiently. In any case, some papers can be found in the literature which propose hypercube algorithms for particular problems which are efficient for particular machine configurations. Examples are [1],[2], [4], [9], [12], and [13], just to mention a few.

In this paper we propose a method to derive hypercube algorithms which are able to exploit features (a) and (b). The method takes as an starting point a hypercube algorithm to solve the problem and transforms it in a systematic way, using a technique that we call communication pipelining. The starting algorithm must belong to a class of hypercube algorithms which we call CC-cube algorithms. Many numerical and symbolic computation problems can be solved using a CC-cube algorithm. FFT [9], Hartley transform [3], All-to-All personalized communications [8] and Jacobi methods for singular value decomposition and eigenvalue computation [11] are just some examples.

* This work has been supported by the Ministry of Education and Science of Spain (CICYT TIC-92/880 and TIC-91/1036) and the European Center for Parallelism in Barcelona (CEPBA).

In this paper, performance figures are given for two concrete application examples: FFT and Vector Add. Hypercube algorithms for FFT have been extensively studied in the literature. Among others, [9] and [2] are two concrete examples of algorithms which try to exploit communication parallelism ([9]) and communication/computation overlapping ([2]). In both cases the degree of communication parallelism and overlapping is fixed and, therefore, the results are only efficient for some machine configurations.

The communication pipelining technique, which is the basis of the proposed method, has been used in a previous paper [5], in which only communication parallelism is considered. The main contribution of this paper is the extension of the method in order to include the overlapping of communications and computations.

The rest of the paper is organized as follows. Next section describes the architecture assumptions. Section 3 defines the characteristics of a CC-cube algorithm. Section 4 reviews the communication pipelining technique. Section 5 establish the overlapping method that is based on the previously described concept of communication pipelining. Section 6 shows the performance figures that the method provides. Finally, the conclusions are found in section 7.

2 Target architecture

This study assumes a distributed memory multicomputer consisting of 2^d processors connected by bidirectional point-to-point links in a d-dimensional hypercube topology. Every node can send messages to any of its d neighbors following an asynchronous protocol. This means that, after initiating a communication operation through one or several of its links, a processor can continue performing computations in parallel with the transmission of the data. It is also assumed that every node can send messages in parallel along different links of the hypercube. However, the start-up times for the different communications cannot be overlapped (we assume that the start-up time corresponds mostly to time spent by the processor to initiate each transmission). Therefore, the cost of sending c messages in parallel along c different dimensions of the hypercube and performing afterwards C computations is considered to be:

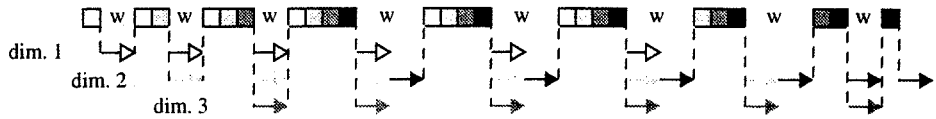
$$cT_{sup} + \text{Max}\,(CT_a, L_{max}T_e)$$

where T_a is the time to perform a computation, T_{sup} is the communication start-up, T_e is the communication time per size unit and L_{max} is the size of the longest message to be sent. This model is in fact an upper bound that will be exact only in the case that the longest message is the last one to be sent. Notice that this model is valid for any control flow method (store-and-forward, wormhole, circuit switching) since the communications always take place between neighbor nodes.

3 CC-cube algorithms

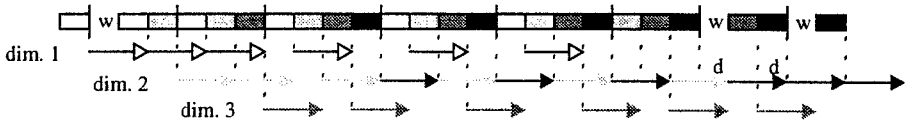
A CC-cube algorithm consists of 2^d processes that perform some computation and exchange data among them. Each process communicates only with other d processes following a hypercube communication topology. Every process executes the same code, which has the following structure:

```
do i = 1, K
  compute x_i[1:N] plus some local data
  exchange x_i with neighbor in dimension d_i
enddo
```



w: Node is waiting for data.

Fig. 1. An example of communication pipelining. The packets with the same gray level belong to the same iteration of the original CC-cube algorithm.



w: Node is waiting for data. d: Communication is delayed due to link contention.

Fig. 2. Pipelining and overlapping of communications and computations.

where d_i is one of the dimensions of the hypercube ($d_i \in [1, d]$). Note that it is not necessary that each iteration uses a different dimension. The computation of $x_i[j]$ is a function of $x_{i-1}[j]$ (which was computed in iteration i–1 by the neighbor in dimension d_{i-1}) and possibly some local data.

4 Communication pipelining

The communication pipelining technique is inspired in the software pipelining approach used to generate code for VLIW processors [10]. Communication pipelining is based on the fact that, in order to compute $x_i[j]$ it is not necessary to have received the whole vector x_{i-1} from the neighbor in dimension d_{i-1} but simply element $x_{i-1}[j]$. In this situation, the algorithm is rewritten in as follows. Every vector x_i is decomposed into Q packets. As Fig. 1 shows, in the first iteration every node computes the first packet of x_1 and sends the result to neighbor through dimension d_1. In the second iteration, every node computes the second packet of x_1 and the first packet of x_2 (it has all the data required to perform these computations). At the end of this second iteration, each node sends two messages, one of them to neighbor through dimension d_1 containing the second packet of x_1, and the other one to neighbor through dimension d_2, containing the first packet of x_2. If $d_1 \neq d_2$, both packets can be sent in parallel; otherwise, they are combined into a single message and sent to its destination. Proceeding in this way, at the end of the third iteration every node can send three messages in parallel (if the involved dimensions are different). Following this approach, a parallel algorithm that makes use of all the links of the hypercube at the same time can be designed.

5 Overlapping communication and computation

After a packet is computed it can be sent to its destination at the same time that the following packet is computed. As Fig. 2 shows, the nodes do not send the data at the end of every iteration but they send the data after the computation of every packet (if the link is busy the communication is delayed until it becomes idle). In parallel with the transmission, the nodes compute consecutive packets if they have the necessary data.

The complete study of the execution time and the corresponding analytical models can be found in [6]. These models have been used to obtain the performance figures presented in the next section.

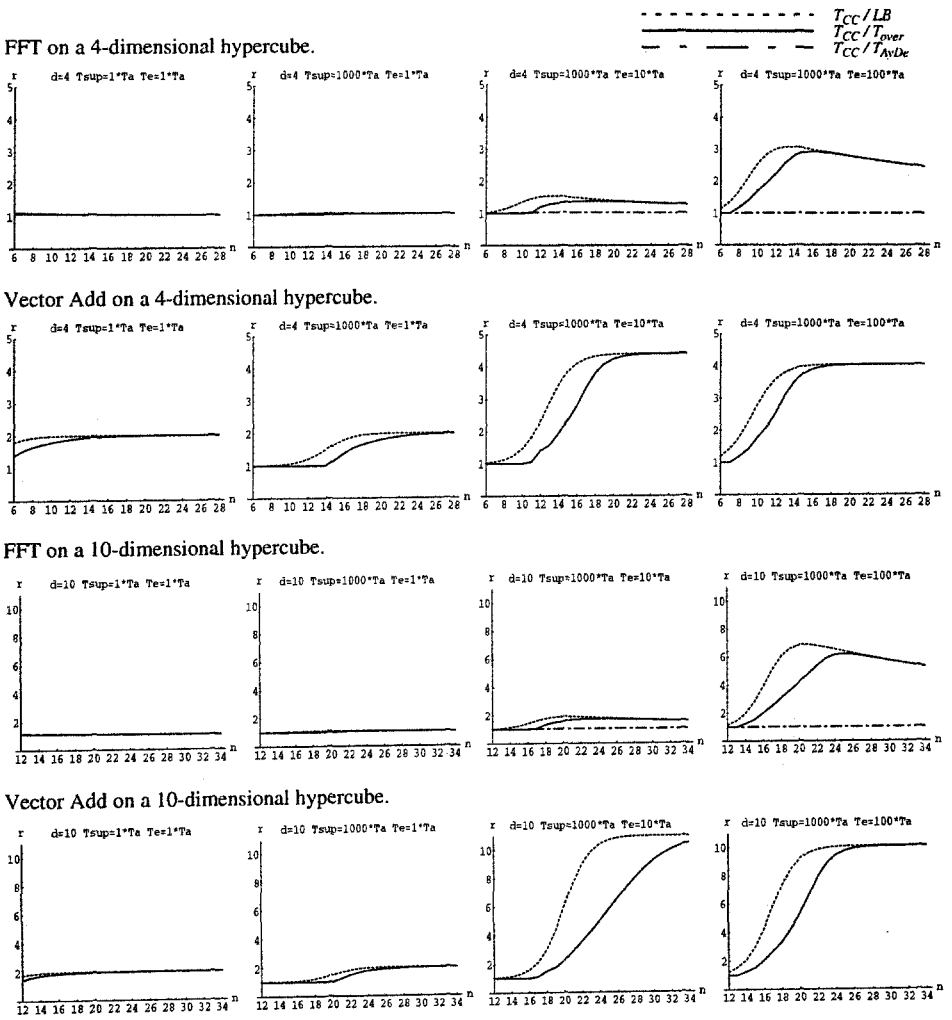


Fig. 3. Performance improvement of the overlapping scheme in relation to the CC-cube algorithm.

6 Performance figures

The plots in figure 3 show the performance improvement of the overlapping scheme in relation to the CC-cube algorithm (T_{CC}/T_{ov}, where T_{ov} is the execution time when using the overlapping method). In addition, they also show the performance improvement of a hypothetical algorithm that achieves an execution time equal to a lower bound (T_{CC}/LB). This lower bound represents the execution time under the assumption that an ideal overlapping of computations and communications can be achieved. The objectives of these plots are twofold: it is intended to show that overlapping provides a very important improvement and that it is very close to the optimum. These figures combine both problems (FFT and Vector Add) for two different sizes of the hypercube ($d = 4, 10$), varying the size of problems ($n \in [6, 34]$) and the communication parame-

ters (T_e/T_a = 1, 10, 100 and T_{sup}/T_a = 1, 1000). In case of the FFT problem, the figures also show the relative improvement of the method proposed by Aykanat and Dervis (T_{CC}/T_{AyDe}) [2].

7 Conclusions

The main conclusion of the performance figures is that the proposed overlapping scheme gives results very close to the lower bound for all machine configurations and problem sizes. In addition, they show that a significant improvement is achieved in many cases (more than a factor 10 in some of the examples).

The main features of the proposed overlapping method are: a) It is a method that can be applied to a wide range of algorithms for hypercubes. In the literature, there are particular solutions to specific problems that are difficult to adapt to other problems. We have illustrated the method by means of two real problems: FFT and Vector Add. There are many other problems on which the method can be applied. b) The method permits to tune the degree of overlapping for a given problem and a given architecture (due to lack of space we have omitted the derivation of this parameter). Other proposals, as the one of Aykanat and Dervis, besides being a particular solution to a specific problem, they cannot vary the amount of overlapping and therefore the performance of the algorithm is much lower than that of our scheme, for some machine configurations.

8 References

1. Agarwal, R. C., Gustavson, F. G., Zubair, M.: An Efficient Algorithm for the 3-D FFT NAS Parallel Benchmark. Scalable High-Performance Computing Conf. (1994) 129-133
2. Aykanat, C., Dervis, A.: An Overlapped FFT Algorithm for Hypercube Multicomputers. ICPP (1991) III-316 - III-317
3. Aykanat, C., Dervis, A.: Efficient Fast Hartley Transform Algorithms for Hypercube - Connected Multicomputers. IEEE Transactions on Parallel and Distributed Systems, vol. **6**, no. **6** (1995) 561-577
4. Clement, M. J., Quinn, M. J.: Overlapping Computations, Communications and I/O in Parallel Sorting. Journal of Parallel and Distributed Computing **28** (1995) 162-172
5. Díaz de Cerio, L., González, A., Valero-García, M.: Communication Pipelining in Hypercubes (submitted for publishing)
6. Díaz de Cerio, L., Valero-García, M., González, A.: Overlapping Communication and Computation in Hypercubes. DAC/UPC Research Report No. RR-96/02 (1996)
7. Fox, G. et al.: Solving Problems on Concurrent Processors. Englewood Cliffs, N. J. Prentice - Hall (1988)
8. Johnsson, S. L., Ho, C. T.: Optimum broadcasting and Personalized Communication in Hypercubes. IEEE Trans. Comput. **38** (1989) 1249-1268
9. Johnsson, S. L., Krawitz, R. L.: Cooley-Tukey FFT on the Connection Machine. Parallel Computing **18** (1992) 1201-1221
10. Lam, M.: Software Pipelining: An Effective Scheduling Technique for VLIW machines. Conf. on Programming Language Design and Implementation (1988) 318-328
11. Mantharam, M., Eberlein, P. J.: Block Recursive Algorithm to Generate Jacobi-sets. Parallel Computing **19** (1993) 481-496
12. Sahay, A.: Hiding Communication Costs in Bandwidth-Limited Parallel FFT Computation. Report: UCB/CSD 93/722, University of California (1993)
13. Suarez A., Ojeda-Guerra, C.: Overlapping Computations and Communications in Tours Networks. 4th Euromicro Workshop on Parallel and Distributed Processing (1996) 163-169
14. Thomson Leighton, F.: Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes. Morgan Kaufmann Publishers (1992)