Divide-and-Conquer Algorithms on Two-Dimensional Meshes*

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

Dept. d'Arquitectura de Computadors - Universitat Politcnica de Catalunya
c/Jordi Girona 1-3, Campus Nord - D6, E-08034 Barcelona (Spain)
{miguel, antonio, ldiaz, dolors}@ac.upc.es

Abstract. The Reflecting and Growing mappings have been proposed to map parallel divide-and-conquer algorithms onto two - dimensional meshes. The performance of these mappings has been previously analyzed under the assumption that the parallel algorithm is initiated always at the same fixed node of the mesh. In such scenario, the Reflecting mapping is optimal for meshes with wormhole routing and the Growing mapping is very close to the optimal for meshes with store-and-forward routing. In this paper we consider a more general scenario in which the parallel divide-and-conquer algorithm can be started at an arbitrary node of the mesh. We propose and approach that is simpler than both the Reflecting and Growing mappings, is optimal for wormhole meshes and better than the Growing mapping for store-and-forward meshes.

1 Introduction

The problem of mapping divide-and-conquer algorithms onto two - dimensional meshes was addressed in [2]. First, a binomial tree was proposed to represent divide-and-conquer algorithms. Then, two different mappings (called the Reflecting and Growing mappings) were proposed to embed binomial trees onto two-dimensional meshes. It was shown that the Reflecting mapping is optimal for wormhole routing since the required communication can be carried out in the minimum number of steps (there are not conflicts in the use of links). On the other hand, the Growing mapping was shown to be very close to the optimal for the case of store-and-forward routing.

The communication performance of the Reflecting and the Growing mappings was analyzed in [2] under the assumption that the divide-and-conquer algorithm is always started at a fixed node of the mesh. In the following, we use the term fixed-root to refer to this particular scenario. In this paper, we consider a more general scenario in which a divide-and-conquer algorithm can be started at any arbitrary node of the mesh. The term arbitrary-root will be used to refer to this scenario. It will be shown that the Reflecting mapping is still optimal for wormhole routing in the arbitrary-root scenario but the performance of the Growing mapping for store-and-forward routing can be very poor in some common cases.

* This work was supported by the Ministry of Education and Science of Spain (CICYT TIC-429/95).

An alternative solution for the arbitrary-root scenario is proposed in this paper, which is inspired in a previous work on embedding hypercubes onto meshes and tori [1]. The proposed scheme, which will be called DC-cube embedding, has the following properties: a) it can be applied to meshes with either wormhole or store-and-forward routing, b) it is significantly simpler than the Reflecting and Growing mappings, c) it is optimal for wormhole routing, and d) it is significantly faster than the Growing mapping in some common cases, for store-and-forward routing. A more detailed explanation of the results presented in this paper can be found in [4].

The rest of this paper is organized as follows. Section 2 reviews the Reflecting and Growing mappings [2], which are the basis for our proposal. In section 3, these mappings are extended to the arbitrary-root scenario. Section 4 presents our proposal. Finally, section 5 presents a performance comparison of the different approaches and draws the main conclusions.

2 Background

Rajopadhye and Telle [2] propose the use of a binomial tree to represent a divide-and-conquer algorithm. Every node of the binomial tree represents a process that performs the following computations:

1. Receive a problem (of size x) from the parent (the host, if the node is the root of the tree).
2. Solve the problem locally (if 'small enough"), or divide the problem into two subproblems, each of size αx), and spawn a child process to solve one of these parts. In parallel, start solving the other part, by repeating step 2.
3. Get the results from the children and combine them. Repeat until all children's results have been combined.
4. Send the results to the parent (the host, if the node is the root).

Step 2 is referred to as the division stage. It has a number of phases corresponding to the different repetitions of step 2 (the number of levels of the binomial tree). The division stage is followed by the combining stage in which the results of the different subprobems are combined to produce the final result. For the sake of simplicity, as in [2], only the division stage will be considered from now on. Two different values for parameter α (see step 2) will be considered: $\alpha = 1$ (the problem is replicated) and $\alpha = 1/2$ (the problem is halved).

The execution of a binomial tree on a two-dimensional mesh can be specified in terms of the embedding of the tree onto the mesh. Two different embeddings were proposed in [2]: the Reflecting mapping and the Growing mapping. It was shown that the Reflecting mapping is optimal for wormhole routing since the communications are carried out without conflicts in the use of the mesh links. On the other hand, the Growing mapping is close to the optimal under store-and-forward routing. These performance properties were derived assuming a fixed-root scenario, that is, the root of the tree is always assigned to the same mesh node.

3 Extending the Reflecting and Growing Mappings to the Arbitrary-Root Scenario

In the following, we consider the case in which the divide-and-conquer algorithm can be started at an arbitrary node (a, b) of the mesh (this is called the arbitrary-root scenario).

We have first considered two straightforward approaches to extend the Reflecting and Growing mappings to the arbitrary-root scenario. In approach A we build a binomial tree which is an isomorphism of the binomial tree used in [2] for the fixed-root scenario (every label of the new tree is obtained by a fixed permutation of the bits in the old label). The isomorphism is defined so that the root of the binomial tree is mapped (by the corresponding embedding Reflecting or Growing) onto node (a, b). In approach B, the whole problem is first moved from node (a, b) to the starting node according to the original proposal (for the fixed-root scenario). These approaches will be compared with our proposal, that is described in the next section.

4 A New Approach for the Arbitrary-Root Scenario

Our approach to perform a divide-and-conquer computation on a mesh, under the arbitrary-root scenario is inspired in the technique proposed in [1] to execute a certain type of parallel algorithms (which will be referred to as CC-cube algorithms) onto multidimensional meshes and tori.

A d-dimensional CC-cube algorithm consists in 2^d processes which cooperate to perform a certain computation and communicate using a d-dimensional hypercube topology (the dimensions of the hypercube will be numbered from 0 to $d-1$). The operations performed by every process in the CC-cube can be expressed as follows:

```
do i=0, d-1
    compute
    exchange information with neighbour in dimension i
enddo
```

The above case corresponds to a CC-cube which uses the dimensions of the hypercube in increasing order. However, any other ordering in the use of the dimensions is also allowed in CC-cubes.

A CC-cube can executed in a mesh multicomputer by using an appropriate embedding. In particular, the standard and xor embeddings were proposed to map the CC-cube onto a multidimensional meshes and tori respectively. The properties of both embeddings were extensively analyzed in [1, 3].

A divide-and-conquer algorithm can be regarded as a particular case of CC-cube algorithm. This particular case will be referred as DC-cube (from Divide-and-Conquer) and has the following peculiarities with regard to CC-cubes:

- a) In every iteration, only a subset of the processes are active (all the processes are active in every iteration of a CC-cube).

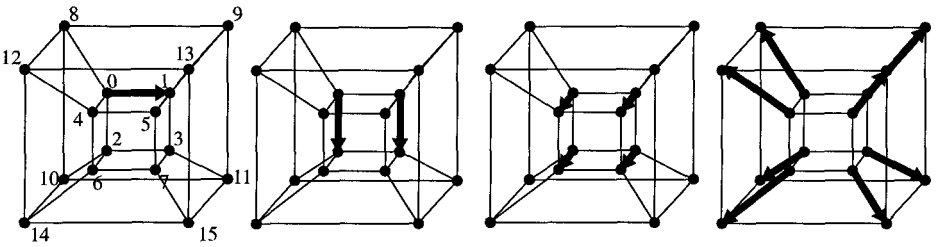

Fig. 1. The four iterations of a 4-dimensional DC-cube starting at process 0 and using the hypercube dimensions in ascending order.

- b) Communication between neighbour nodes is always unidirectional (instead of the bidirectional exchange used by CC-cubes).

A particular DC-cube is characterized by: (a) a process which is responsible for starting the divide-and-conquer algorithm, and (b) a certain ordering of the hypercube dimensions, which determine the order in which the processes of the DC-cube are activated. Figure 1 shows an example of a 4-dimensional DC-cube starting at process 0, and using the dimensions in ascending order. In this figure, the arrows represent the communications that are carried out in every iteration of the DC-cube. Finally, it can be shown that a binomial tree with the root labelled as l is equivalent to a d-dimensional DC-cube initiated at process l and using the hypercube dimensions in descending order.

The standard embedding has been proposed to map a $2k$-dimensional hypercube onto a $2^k \times 2^k$ mesh. The function S which maps a node i ($i \in [0, 2^{2k} - 1]$) of the hypercube onto a node (a, b) ($a, b \in [0, 2^k - 1]$) of the mesh is defined as[1]:

$$S(i) = \left(\left\lfloor \frac{i}{2^k} \right\rfloor , i \bmod 2^k \right) . \tag{1}$$

The properties of the standard embedding that are more relevant to this paper are: a) It is an embedding with constant distances (this property means that the neighbors in dimension i of the hypercube are found at a constant distance in the mesh, for any pair of neighbor nodes, and b) It has a minimal average distance. Property (a) is very attractive for the purpose of using the embedding on the arbitrary-root scenario. Property (b) is attractive from the performance point of view.

To start a divide-and-conquer algorithm from an arbitrary node (a, b) of the wormhole mesh, we use a DC-cubeinitiated at process $S^{-1}((a, b)) = a \cdot 2^k + b$, and using the dimensions of the hypercube in descending order. It is easy to see that the standard embedding of such a DC-cube is conflict free, and therefore the approach is optimal. The formal proof of this property is very similar to the

[1] The standard embedding can be easily extended to the general case of C-dimensional meshes. This extension is however out of the scope of this paper.

Table 1. Comparision of approaches

	t_s	t_e (general α)	t_e ($\alpha = 1$)	t_e ($\alpha = 0.5$)
G_A	$2^{k+1} - 2$	$(\alpha + \alpha^2)2^{k-1} + (\alpha^3 + \alpha^4)\frac{(2\alpha^2)^{k-1}-1}{2\alpha^2-1}$	$2^{k+1} - 2$	$\frac{3}{8}\left(2^k + 1 - \frac{1}{2^{k-1}}\right)$
G_B	$3 \cdot 2^{k-1} - 1$	$2^{k-1} - 1 + \alpha + \alpha^2 + (\alpha^3 + \alpha^4)\frac{(2\alpha^2)^{k-1}-1}{2\alpha^2-1}$	$3 \cdot 2^{k-1} - 1$	$2^{k-1} + \frac{1}{8} + \frac{3}{2^{k+2}}$
S	$2^{k+1} - 2$	$(\alpha + \alpha^2)\frac{2^k\alpha^{2k}-1}{2\alpha^2-1}$	$2^{k+1} - 2$	$\frac{3}{2}\left(1 - \frac{1}{2^k}\right)$

proof given in [2] for the case of the Reflecting mapping under the fixed-root scenario.

To start a divide-and-conquer algorithm from an arbitrary node (a, b) of the store-and-forward mesh, we use a DC-cube initiated at process $S^{-1}((a, b)) = a \cdot 2^k + b$, and using the dimensions of the hypercube in ascending order. In this way, the distances corresponding to the first iterations of the computation (involving larger messages) are smaller.

5 Comparison of Approaches and Conclusions

As a general consideration, it can be said that the standard embedding is significantly simpler than both Reflecting and Growing mapping.

When considering the wormhole arbitrary-root scenario, both the approach A for Reflecting mapping and the standard embedding of DC-cubes are optimal.

The comparison of approaches for the store-and-forward arbitrary-root scenario are summarized in table 1, in terms of the average communication cost (G_A for approach A, G_B for approach B and S for the standard embedding of DC-cube). The average cost G_A is defined as:

$$G_A = \frac{1}{2^{2k}} \sum_{a=0}^{2^k-1} \sum_{b=0}^{2^k-1} G^{a,b} \quad , \tag{2}$$

where $G^{a,b}$ is the cost of the division stage when the computation is initiated at node (a, b). This cost is defined in terms of the startup cost incurred in every communication between neighbor nodes (denoted by t_s) and the transmission time per message size unit (denoted by t_e). The average costs G_B and S are defined in a similar way. Two particular cases of the term affecting t_e are distinguished, corresponding to $\alpha = 1$ and $\alpha = 0.5$.

The expressions in table 1 can be compared in two different cases. In the "small volume" case, the communication cost is assumed to be dominated by the term affecting t_s (this will happen when t_s/t_e is large and/or the problem is small). In the "large volume" case, the cost is assumed to be dominated by the term affecting t_e (this will happen when t_s/t_e is small and/or the problem is large).

The conclusions drawn from table 1 are:

- a) In the "small volume" case, approach B is the best, since: $G_A = S = (4/3)G_B$.

- b) In the "large volume" case and $\alpha = 1$, the conclusion is exactly the same as case (a).
- c) In the "large volume" case and $\alpha = 0.5$, approach S is significantly better than the rest, since:

$$G_A = \left(2^{k-2} + \frac{1}{2}\right) S \text{ and } G_B = \left(\frac{2^k}{3} + \frac{5}{12}\right) S \ . \tag{3}$$

Note that cases (a) and (c) are expected to be the most frequent since a parallel computer is targeted to solve large problems. Besides, they are also the most relevant since they may be very time consuming.

Note that in case (c) the improvement of the standard embedding of DC-cube over approaches A and B is proportional to the number of nodes and therefore it is very high for large systems.

References

1. González, A., Valero-García, M., Díaz de Cerio L.: Executing Algorithms with Hypercube Topology on Torus Multicomputers. IEEE Transactions on Parallel and Distributed Systems **8** (1995) 803–814
2. Lo, V., Rajopadhye, S., Telle, J.A.: Parallel Divide and Conquer on Meshes. IEEE Transactions on Parallel and Distributed Systems **10** (1996) 1049–1057
3. Matic, S.: Emulation of Hypercube Architecture on Nearest-Neighbor Mesh-Connected Processing Elements. IEEE Transactions on Computers **5** (1990) 698–700
4. Valero-García, M., González, A., Díaz de Cerio, L., Royo, D.: Divide-and-Conquer Algorithms on Two-Dimensional Meshes. Research Report UPC-DAC-1997-30, `http://www.ac.upc.es/recerca/reports/INDEX1997DAC.html`