Dynamic Reconfiguration of Multicomputer Networks: Limitations and Tradeoffs

J. M. García
Departamento de Informática
Universidad de Castilla-La Mancha
Albacete, SPAIN 02071
utjmgarcia@02ccv1.ucma.es

J. Duato
Facultad de Informática
Universidad Politécnica de Valencia
Valencia, SPAIN 46071
jduato@aii.upv.es

Abstract

The dynamic reconfiguration of the interconnection network is an advanced feature of some multicomputers to reduce the communication overhead. Up to now, the work carried out in this field has focused on static switching, i.e., the network changes its topology before starting the execution of a phase of an application program and then it remains constant throughout the phase execution. However, our work focuses on true *dynamic reconfiguration, i.e., the network topology can change almost arbitrarily at runtime. In a previous paper [6], we presented an algorithm to handle the dynamic reconfiguration and some simulation results, showing the benefits achieved by this reconfiguration algorithm. In this paper, we expound in depth the reconfiguration algorithm and the different concepts related to it. The previous work is analyzed and compared with our algorithm, showing the improvements we have achieved.*

1. Introduction.

Multicomputers are among the most interesting architectures, meeting the high performance computing requirements in application areas such as computational fluid dynamics, image processing and circuit simulation. Multicomputers rely on an interconnection network between nodes (processors) to support the message-passing mechanism.

The interconnection network plays a major role in determining the overall performance of a multicomputer. If the network cannot provide adequate performance, nodes will frequently be forced to wait for data to arrive. Moreover, as technological improvements increase the computing power of nodes, additional pressure is placed on the communication network to provide a comparable improvement in performance to avoid becoming a bottleneck. In addition to performance considerations, the interconnection network should be tolerant to failures.

As each processor has a limited number of links, it can directly communicate with a few neighbours. Thus, due to classical integration constraints (pin limitations, wiring problems, communications capabilities, etc), most multicomputers are based on a fixed interconnection topology. We face two possibilities:

- For specialized architectures, the interconnection topology is selected in such a way that it matches the communication requirements of a specific (class of) application(s).
- For more general purpose architectures, the processing elements are connected according to a regular pattern. But routing mechanisms must be implemented to allow a processor to communicate with a non-neighbour processor. Such routing mechanisms may introduce unacceptable delays.

Classical solutions to this problem are selecting the optimal topology, or improving the routing mechanisms (for example, wormhole routing [4]) or using a dynamic remapping for processes. A novel solution is to make the interconnection topology reconfigurable, i.e., the network topology can change at runtime [1,3]. There are two approaches to handle the dynamic reconfiguration of a multicomputer network. The first approach is based on a fixed topology (static switching). A program is divided into several phases, where each phase requires a different topology. Before a new phase starts its execution, a new topology is selected by means of a software reconfigu-ration point. This approach is quite simple, but the flexibility of reconfiguration is limited.

The second approach consists of changing the topology arbitrarily at runtime. In this approach, any arbitrary topology is allowed, so that the interconnection network can easily match the communication requirements of a given program. The main problem of this approach is the software complexity and overhead, because the dynamic reconfiguration implies a cost.

Our research has been motivated by the need to improve the latter approach. We have tried to handle the dynamic reconfiguration of the network in a more efficient way. A network with this feature has the following properties:

- It can easily match the network topology to the communication requirements of a given program, properly exploiting the locality in communications. It provides the flexibility required for an efficient execution of various applications. Moreover, an optimal topology can be chosen for each application or each application phase.
- Programming a parallel application becomes more independent of the target architecture because the architecture adapts to the application.
- Classical mapping problems become easier to manage in such a reconfiguration context.
- Faulty nodes or links can be bypassed and spare nodes can be switched on.

The two main objectives of our project were the design of an algorithm to handle the dynamic reconfiguration and the development of software tools needed for coding applications that efficiently use the reconfiguration capabilities of the architecture.

In this paper, we present in depth the main concepts and options about dynamic reconfiguration, its limitations and tradeoffs. We also present our algorithm for the dynamic reconfiguration of the network. Reconfiguration is limited, preserving the original topology. Long distance message passing is minimized by positioning communication partners close to each other. The algorithm decides when a change must be carried out by means of a cost function. This algorithm is transparent to the programmer and is not restricted to a particular class of application, being very well suited for parallel applications whose communication pattern varies over time.

The remainder of this paper is organized as follows. In the next section, we outline some concepts and consider some tradeoffs for the design of the reconfiguration algorithm. Section 3 is devoted to describe the reconfiguration algorithm we have developed. In section 4 we explain the problems we have found while testing the algorithm. Section 5 is devoted to show the related work in this field. Finally, in section 6, we point out some conclusions.

2. The dynamic reconfiguration: definitions and trade offs.

The goal of the dynamic reconfiguration of the network is to increase the multicomputer performance by minimizing the traffic of messages in the network. Therefore, when the traffic between a pair of nodes is intense, the dynamic reconfiguration algorithm will try to put the source node close to the destination node.

We want to reduce two parameters: the total message traffic in the network and the maximum node traffic. And we also want to have a small number of changes for

keeping the reconfiguration cost low. In order to explain our algorithm, it is necessary to introduce several concepts and tradeoffs regarding the dynamic reconfiguration.

1. Local versus global reconfiguration. A local reconfiguration only affects the requesting node and any other node. A global reconfiguration, on the other hand, can bring about a modification in the links of multiple nodes at the same time. In a centralized control scheme, global reconfiguration could produce a great cost because the master node must collect information from all the nodes in the network, but its advantage is that several changes can be carried out in one step. In a distributed control scheme, global reconfiguration could produce deadlock problems.

2. Preserving a given topology. Reconfiguration can be made in such a way that the new network has the same topology (only some nodes have exchanged their places) or allowing any arbitrary topology. Preserving a given topology is equivalent, although much faster, to renumbering some nodes, also exchanging all the processes they are executing. Also, it is not necessary to modify the routing algorithm. Alternatively, any arbitrary topology may be allowed even irregular topologies. In the last case, we must take into account deadlock problems.

3. Reconfiguration triggering. Somehow the reconfiguration algorithm has to decide when the reconfiguration of the network is suitable. The algorithm may use a cost function based on different values measured from the network. This cost function does not take into account internal communications (between processes executed in the same node), because this communications do not produce network traffic.

Up to now, we have taken into account two important aspects: the *weight* of a communication and the *distance* between nodes. By *weight* of a communication we mean the number of messages which have been sent in a particular direction. We have measured the distance between nodes using the *nominal distance*, i.e., it measures the number of intermediate nodes that a message has to cross to go from node A to node B; if A and B are physically connected, the distance is null.

4. Type of change. When a node decides that it is convenient to make a reconfiguration, it computes the effect of possible changes to see if the traffic conditions will improve. There are two strategies to choose possible changes:

a) *Small alteration in the network*. A node can only exchange its position with one of its neighbour nodes. For example, in a ring topology a node only has two possibilities of exchange, with the node on the right, or the node on the left.

b) *Large alteration in the network*. A node can change to any other place in the network. To keep the same topology easily, it is convenient for the changed node to take the position left by the node which requested the change.

In a large alteration strategy, less exchanges are needed to approach the source and destination nodes, i.e., we need only one exchange to put the source node close to the destination node. However, in a small alteration strategy, we need several exchanges to reach the same final situation. Therefore, it seems better to use a large alteration. However, when all the nodes are communicating with each other this may not be true, because this strategy produces brusquer and less uniform changes than the other one.

5. Thresholds in the reconfiguration. Two thresholds have been detected for the correct execution of the algorithm. The first threshold checks the value of traffic in the network for reconfiguration triggering and the other threshold determines how often the cost function mentioned in point 3 should be evaluated.

These two thresholds try to minimize the cost implied by a network reconfiguration, choosing the best moment to carry out a change. On the one hand they try to avoid a large number of reconfigurations (a great cost), and on the other hand, a small number (no improvement in the performance of the multicomputer) of reconfigurations.

6. Reconfiguration control. There are two ways for controlling the dynamic

reconfiguration of the network: a centralized control (a node -the system controller- is responsible to reconfigurate the network) or a distributed control (each node controls its own reconfiguration).

In a centralized control, the system controller is connected to all the nodes of the system through a control bus. This type of control could be a bottleneck for the system, since all the nodes must send reconfiguration requests to the system controller. This situation may be avoided by means of the reconfiguration algorithm for moderate systems, i.e. up to 64 nodes.

An alternative is to use a distributed control. In this case, several network reconfigurations can be carried out simultaneously, without bottleneck problems. Furthermore, distributed control is well suited for fault tolerant operation as it does not require any central facility and can easily bypass faulty nodes or links. However, the deadlock problem could appear. Another problem is that every node must communicate with each other to make a change, increasing the message traffic in the network.

In both schemes there exists a considerable overhead to carry out a reconfiguration [3]. Therefore, it is too costly to change the topology very frequently (e.g. for each message). So, it is more adequate to make a change when an intense traffic between two non-neighbour nodes is observed (a large amount of data is being transferred between these two nodes).

3. Our reconfiguration algorithm.

As stated above, the reconfiguration algorithms have different possibilities. In our case, the reconfiguration algorithm was developed while trying to increase the communication performance of a transputer-based machine, the PARSYS SN 1000. Therefore, the main features of this algorithm have been designed for that target machine.

We have developed a reconfiguration algorithm with the following features: it uses local reconfiguration, it preserves the topology, it is based on a cost function, it produces a small alteration in the network, it uses two thresholds for network reconfiguration and it has a centralized control.

The implementation of this algorithm is distributed, each node taking into account the information recorded locally.

To handle the reconfiguration of the network, we have developed a network reconfiguration protocol among the nodes and the system controller. The main steps are the following:

1) When a node decides that it is necessary to reconfigurate the network, it sends a signal to the system controller through the control bus. This decision is made evaluating the cost function: when its value is greater than threshold_1, a reconfiguration will be convenient. The reconfiguration algorithm only evaluates the cost function each time it receives a number of messages threshold_2. When a node sends a message, this information is only recorded without doing anything else.
2) Then, the system controller informs all the nodes that it is going to make a reconfiguration, and therefore they should stop sending messages to each other. To minimize the reconfiguration time and relieve the cost that it implies, messages in transit are only allowed to reach the next intermediate node. After stopping messages in transit, each node sends an acknowledgement to the system controller to ensure that message traffic in the network has stopped.
3) The node which made the request sends the reconfiguration data to the system controller to carry out the reconfiguration.
4) The system controller modifies the interconnection network topology, adapting it to the new circumstances.
5) Once the new configuration has been established, the system controller broadcasts this configuration to all the nodes and permits node communication again. That information is used by the routing algorithm.

This protocol is easily implementable using the control bus available in the

Supernode architecture, which does not add message traffic to the network. Moreover, the use of a bus allows the system controller to perform efficiently the broadcasting operations required in steps 2 and 5.

We can see from the reconfiguration protocol that our algorithm has two parts: one for the system controller and another one for the remaining nodes. These algorithms are included in the run-time kernel, whether of the system controller or the nodes.

As the machine is not ideal, whenever a change is made in the topology, some time is wasted. The reconfiguration time has been included in the cost function of the reconfiguration algorithm.

3.1 The reconfiguration cost.

Now, we are going to describe briefly how to evaluate the cost involved when a change in the network topology is performed. By analysing the network reconfiguration protocol, the following times can be obtained:

a) T_a : Time spent in each node to evaluate the cost function and to decide whether a reconfiguration is needed. It measures part of the step 1 of the previous protocol.

b) T_b : Time spent to transfer the reconfiguration information from the node which requests the change to the system controller. It includes the time required to request the change as well as broadcasting the message to stop the communication between nodes, receiving the acknowledgement and sending information about the new topology to the system controller. Therefore, this time includes part of the step 1 and steps 2 and 3 of the previous protocol.

c) T_{rec} : Time needed to reconfigurate the interconnection network topology. It corresponds to step 4 of the previous protocol.

d) T_c : Time needed to broadcast the new configuration to all the nodes, allowing again communications between processors. It corresponds to the last step of the previous protocol.

Then, the cost for making a change in the network topology is given by the following expression:

$$T_{change} = T_a + T_b + T_{rec} + T_c$$

We have evaluated an upper bound for this expression. We have found a T_{change} of 1,5 ms for short messages and a T_{change} of 2 ms for long messages. These values have been included in the cost function of the reconfiguration algorithm for the real case.

4. Problems with the algorithm and solutions.

Recently, we have evaluated the reconfiguration algorithm for several numerical problems. The results are very promising, with a total message traffic reduction of 35% or more. These results and a wider study of several test cases can be found in [6]. Our evaluations have been obtained with the FDP environment we have developed [7], both with and without including the reconfiguration cost.

Up to now, we have found two main problems in the dynamic reconfiguration: the endless cyclic changes and the existence of multiple optimal choices.

The **endless cyclic changes** problem appears when the reconfiguration algorithm only takes into account the messages received in each node, without

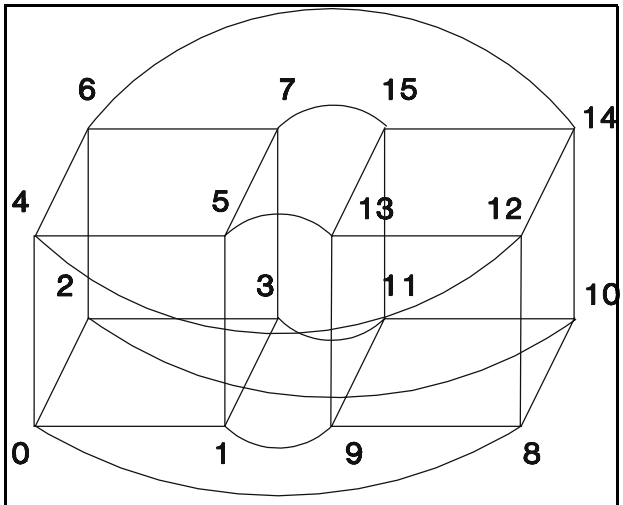

Fig. 1 Hypercube of 16 nodes.

recording the messages it sends. Then, it is

possible that two or more nodes try to reach each other following different paths falling in a situation of cyclic changes. To explain it, let us imagine the following situation. We have a hypercube topology with a deterministic routing algorithm. It forwards messages crossing the hypercube dimensions in decreasing order. Let us consider that there is some message traffic among nodes *0*, *4*, and *5* in the positions 0, 4 and 5 respectively in this way: node *0* sends messages to node *4*, node *4* sends messages to node *5*, and node *5* sends messages to node *0*. Then, this message traffic in the network produces that the reconfiguration algorithm in each node tries to reduce the traffic. So, node *0* changes its position to position 1 to approach node *5*. Then, node *4* changes its position to position 0 to approach node *0*, and then, node *5* changes its position to position 4 to approach node *4*. Again, node *0* tries to approach node *5* and so on. This situation leads to a high number of changes and a small reduction in message traffic.

To be able to avoid it, the cost function must take into account the total number of communications between each pair of nodes (input and output messages). This improvement produces a great stability in the process of reconfigurating the interconnection network.

The second problem is the existence of **multiple optimal changes**. When a node detects that the topology must be modified, it tries to carry out a change which minimizes the cost function. In some cases, there are several possible changes that minimize the cost function. If the reconfiguration algorithm always chooses the same strategy, it could lead to endless changes, because two nodes could try to reach the same position in the network. Let us imagine the following situation. We have a hypercube topology with a deterministic routing algorithm. Again, it forwards messages crossing the hypercube dimensions in decreasing order. Let us consider that there is some message traffic in the following way: node *5* in position 5 sends messages to node *0* (in position 0) and to node *3* (in position 3). Then, node *0* tries to place close to node placed in position 5. It has two possibilities that minimizes the cost function: position 1 or position 4. On the other hand, node *3* also tries to reach a position close to position 5. We suposse it has two possibilities too: position 1 or position 7 (see fig. 1). If we have a fixed strategy, for example, to choose the positions with lower numbers. Then, node *0* exchanges its position to position 1. But node *3* also wants to go to position 1, so it carries out an exchange with node *0*. Now, node *3* is in position 1 and node *0* is in position 3. But in this position, node *0* tries to reach position 1 again and so on. Because the strategy of choosing the change is fixed, node *0* and node *3* always try to reach position 1, and never consider the possibility of going to position 7, although this position also minimizes the cost function. This situation could produce a practically infinite number of changes without reduction in message traffic.

This situation is avoided if a round-robin strategy among the changes which minimize the cost function is adopted in the algorithm. In this case, initially node *0* and node *3* choose position 1 to carry out the first change. But the second time, node *0* chooses position 7, finishing the changes in the network.

Our algorithm has been corrected to avoid these problems. In this way, the results we have obtained with the current algorithm are better than those from the preliminary versions.

5. Related work.

When we speak about dynamic reconfiguration in a multicomputer network, we usually think of a limited dynamic reconfiguration. It is due to the fact that most of the work carried out in this field has focused on this approach [1,8,9]. In this way, the programmer has to decompose his application into algorithmic phases. After completion of a given phase, the interconnection topology can be modified before beginning the execution of the following phase. Thus an optimal topology can be automatically associated to each algorithmic phase. For example, we

can choose a tree topology in the phase of initial data broadcasting and a hypercube topology in the computation phase. Therefore, a reconfigurable application can be described as a set of algorithmic phases separated by predetermined reconfiguration points. When a reconfiguration point is reached, a processor synchronization procedure and a switch setting sequence will be initiated. In order to benefit from locality of distributed memory, the programmer must be able to express that data can be left in processor's memory by a process of a given phase for a new process of the following phase. This approach is good for some applications, but it has a drawback: the way to find out which is the best topology for each phase.

Some advances have been carried out about the second approach (true dynamic reconfiguration). In [3], when a pair of processors need to exchange a message, the topology is changed, connecting those processors directly. Paths can be established and destroyed at any time in an unpredictable way. Of course, there are some restrictions in these dynamic reconfiguration capabilities. First, only application graphs of degree 4 can be formed due to the limitation to 4 transputer links. Second, it can happen that an application graph cannot be configured because of blocking, i.e. there are no more free links in the static network. Moreover, since the network reconfiguration implies a cost, a reconfiguration is only made when a large amount of data has to be transferred between a pair of processors. The problem lies in the way to know a priori when a pair of processors are going to exchange several messages. In [10], a dynamic connection is established between processors on the request of the message sender. This request must be added by the programmer to the application code, so the parallel program must be modified to allow the dynamic reconfiguration.

Our work is focused in this last approach. We minimize long distance message passing by positioning communication partners close to each other. However, this task is not carried out by the programmer; it lies on the run-time kernel. We have developed an algorithm to handle the dynamic reconfiguration. The algorithm decides when a change is more adequate, and it is transparent to the programmer. Therefore, the parallel program is the same whether there is dynamic reconfiguration or not.

6. Conclusions.

This paper deals about the dynamic reconfiguration of the interconnection network of a multicomputer. This feature is a valid alternative to solve the main problem associated to multicomputers: the communication bottleneck. By means of using a dynamic topology, the principle of locality in communications is exploited, leading to a substantial improvement in network latency [2].

We explain a way to reconfigurate the network dynamically. We have analyzed several important tradeoffs and then we have presented an algorithm to reconfigurate dynamically the interconnection network for multicomputers with store-and-forward routing. We have detailed the reconfiguration protocol and we have described the main problems found in the algorithm. This algorithm is very well suited for parallel applications whose communication pattern varies over time. Recently, we have evaluated our algorithm for several test cases and the results are very promising [6].

Finally, we have described the other main models developed up to date, detailing the main features of each one and comparing them with the features of our dynamic reconfiguration model.

References

[1] Adamo, J. and Bonello, C. "Tenor++: A dynamic configurer for Supernode machines". *Lecture Notes in Computer Science*, No. 457, pp. 640-651, Springer-Verlag, 1990.

[2] Agarwal, A. "Limits on interconnection network performance". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 2, No.

4, pp. 392-412, October 1991.
[3] Bauch, A.; Braam, R. and Maehle, E. "DAMP: A dynamic reconfigurate multiprocessor system with a distributed switching network". *2nd European Distributed Memory Computing Conference*, Munich, April 1991.
[4] Dally, W.J. and Seitz, C.L. "Deadlock-free message routing in multiprocessor interconnections networks". *IEEE Trans. on Computers*, Vol. C-36, No. 5, pp. 547-553, May 1987.
[5] Fraboul, Ch.; Rousselot, J.Y. and Siron, P. "Software tools for developing programs on a reconfigurable parallel architecture", in D. Gassilloud & J.C. Grossetie (Eds.), *Computing with Parallel Architectures: T.Node*, pp. 101-110, Kluwer Academic Publishers, 1991.
[6] García, J.M. and Duato, J. "An algorithm for dynamic reconfiguration of a multicomputer network". *Third IEEE Symposium on Parallel and Distributed Processing*, Dallas (Texas), December, 1991.
[7] García, J.M. and Duato, J. "An advanced environment for programming transputer networks with dynamic reconfiguration". *Int. Conf. on Parallel Computing and Transputers Applications '92*, Barcelona, September 1992.
[8] Jin, L. et al. "Dynamically reconfigurable architecture of a transputer-based multicomputers system". *20 Int. Conf. on Parallel Processing*, August, 1991.
[9] Nicol, D.A. "Reconfigurate transputer processor architectures", in T.J. Fountain and M.J. Shute (Eds.), *Multiprocessor Computer Architectures*, North-Holland, 1990.
[10] Vincent, P. and Wei, Z. "Dynamic link interconnections for the transputer", in M. Valero, E. Oñate, M. Jane, J.L. Larriba and B. Suárez (Eds.), *Parallel Computing and Transputer Applications*, IOS Press/CIMNE, 1992.