ELSEVIER

Journal of Systems Architecture 46 (2000) 873–888

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

# Dynamic reconfiguration of node location in wormhole networks [☆]

José L. Sánchez [a,*], José M. García [b]

[a] Escuela Politécnica Superior, Departamento de Informática, Campus Universitario – Avda de Esp., Universidad de Castilla-La Mancha, 02071Albacete, Spain

[b] Facultad Informática, Universidad de Murcia, Murcia 30071, Spain

Received 30 June 1998; received in revised form 16 February 1999; accepted 22 September 1999

**Abstract**

Several techniques have been developed to increase the performance of parallel computers. Reconfigurable networks can be used as an alternative to increase the performance. Network reconfiguration can be carried out in different ways. Our research has focused on distributed memory systems with dynamic reconfiguration of node location. Briefly, this technique consists of positioning the processors in the network depending on the existing communication pattern among them, to suit the requirements of each computation.

In this article, we present a dynamic reconfiguration technique for wormhole networks. We have used both a crossbar and a multistage interconnection network to implement a reconfigurable logical two-dimensional (2-D) torus topology. The reconfiguration mechanism is based on a distributed reconfiguration algorithm. The algorithm is based on a cost function that requires only local information. We discuss reconfiguration features and adjust the different parameters of the reconfiguration algorithm. We have also studied the deadlock problem in reconfigurable wormhole networks, and give details of our solution. Finally, we have evaluated the performance of this technique under several workloads. © 2000 Elsevier Science B.V. All rights reserved.

*Keywords:* Interconnection networks; Dynamic reconfiguration; Wormhole switching; Channel pipelining

## 1. Introduction

In parallel machines, several computing nodes work together in order to solve large application problems. The nodes communicate data and coordinate their efforts by sending and receiving messages through an interconnection network. Consequently, the performance of such machines critically depends on the performance of their interconnection networks. As a result, it is important to improve network performance.

During the last decade, many multicomputer networks have used virtual cut-through or wormhole switching [10,12], which are techniques that reduce message latency by pipelining transmission through the channels along a message's route. In wormhole networks, messages are split into flow control digits, or flits. Message blocking is a major problem in wormhole networks. A blocking situation occurs when the header flit cannot find a free

[☆] This work was supported by the Spanish CICYT under Grant No. TIC97-0897-C04.

[*] Corresponding author.

*E-mail address:* jsanchez@info-ab.uclm.es (J.L. Sánchez).

channel in its way towards its destination. Consequently, the remaining flits in the message are blocked. Moreover, a message usually spans multiple channels in these networks. As a result, a blockage on one channel can have an immediate impact on the other channels. Tight coupling among channels can lead to one long message blocking the progress of many others. In wormhole networks, channel coupling effects lead to the performance becoming very sensitive to blockage problems. Clearly, the resulting congestion can significantly reduce network performance.

Several techniques have been proposed to reduce or avoid congestion, such as virtual channels, adaptive routing, random routing, or message combining. Other approaches are based on reconfigurable interconnection networks. These networks have topologies that can be configured during the execution of applications. The objective is to adapt the network topology to the communication requirements of each application. In this way, the cost of non-local communication is reduced. The reconfiguration is carried out by providing an underlying switching layer that enables the necessary changes to be made.

There are several approaches to perform network reconfiguration. For example, several research groups proposed the use of additional connections (which can be fixed broadcast buses [7,17,32]) among the different nodes of the network. Miller and other authors, extended this concept by introducing meshes with reconfigurable buses [14,28,29]. Ben-Asher [8] studied the computational aspects of this kind of network using several topological models.

Many of the latest studies on reconfigurable systems are based on the use of field programmable gate arrays (FPGAs). FPGAs consist of a matrix of fine grain computational elements, usually implemented using lookup tables, with a hierarchy of programmable interconnections. Although traditionally FPGAs have been used for logical design and hardware emulation, their suitability as computing engines for reconfigurable architectures has also been explored [1,19]. Research is also being carried out on the design of coarser grain architectures that incorporate reconfigurable features [20,23,27].

Another method of performing reconfiguration consists of modifying the structure of the network by altering the existing connections among the nodes. This does not necessarily mean that the topology is modified. For instance, we can exchange the position of pairs of nodes in the network. Our work addresses this issue and, as will be indicated in the following sections, the reconfiguration can be applied any time during the execution of any application.

Little work has been done on this approach, and most of it has been developed at European universities. The main reason for this was the development in the P1085 Esprit project [4] of transputer-based multiprocessor boards using the C104 switch, a link switching device with a structure that allows the reconfiguration of the interconnection network. Different proposals of reconfigurable systems can be found [2,4,18]. In all of these projects, reconfiguration of the interconnection network can be considered as *quasi-dynamic* because the changes are made at predetermined points during program execution. In other works [21,22], dynamic network reconfiguration has been studied. These approaches consist of the design and implementation of a reconfiguration algorithm. Moreover, they include the analysis of the conditions under which the algorithm can be developed in order to drive the modification of the network structure in an arbitrary way during the execution of a given application.

In this context, in our previous work we proposed a reconfiguration protocol [24,25,33], including the algorithm that performs the reconfiguration process. This work was developed in the context of multicomputers with store-and-forward switching. In this article, we present the foundations for dynamic reconfiguration in wormhole interconnection networks, and without the transputer-based dependence. Reconfiguration is based on a reconfiguration algorithm distributed at each node. The algorithm decides when and how the reconfiguration will take place. It also evaluates the communication contention and decides when the reconfiguration is more convenient. This algorithm is based on a cost function and requires only local information. Besides, we show the performance benefits of using dynamic network reconfiguration by means of several simulations.

The rest of this article is organized as follows: In Section 2, we introduce reconfigurable wormhole networks and present the algorithm and protocol for dynamic reconfiguration on these networks. In Section 3, we study the deadlock problem in reconfigurable wormhole networks, and in Section 4, we evaluate the performance benefits of using dynamic network reconfiguration. Finally, in Section 5, some conclusions are given.

## 2. Reconfigurable wormhole networks

Most message-passing systems are based on a fixed interconnection topology. In specialized architectures, the interconnection topology is selected so that it matches the communication requirements of a specific application. For more general purpose architectures, routing mechanisms must be implemented to allow a processor to communicate with non-neighbouring processors.

A reconfigurable network can be adopted in order to reduce the communication cost. Basically, the reconfiguration process consists of placing the different processors in those positions in the network that are better suited for the requirements of the computation. The positions depend on the existing communication pattern among the processors at each computational step.

A reconfigurable network has the following important advantages [2]:

- Programming a parallel application becomes less dependent on the target architecture as the architecture adapts to the application.
- It is easy to exploit the locality in communications.
- In wormhole networks, reconfigurable architectures alleviate the congestion due to the blocking problem. This problem is more important in networks with deterministic routing.
- Finally, there are applications in which the communication pattern varies over time. Reconfigurable architectures are very well suited for these applications.

There are two types of reconfiguration: static and dynamic. In this article, we focus on dynamic reconfiguration, that is, the topology can change almost arbitrarily at run-time. Fig. 1 shows the effects of this technique for a very elementary situation. In Fig. 1(a) and (b), messages sent by processors go through the same channels until they reach their destination, thus producing delays in communication due to channel contention. This problem disappears once the situation in Fig. 1(c) is reached. As a result, network performance is improved.

Reconfiguration works as follows: the processor that receives the messages manages to place itself in the best position in the network at every moment. This is achieved by changing the location of this processor in the network, and is carried out by exchanging its position with a neighbouring processor. Exchanging processor positions requires an underlying switching layer that will be studied later.

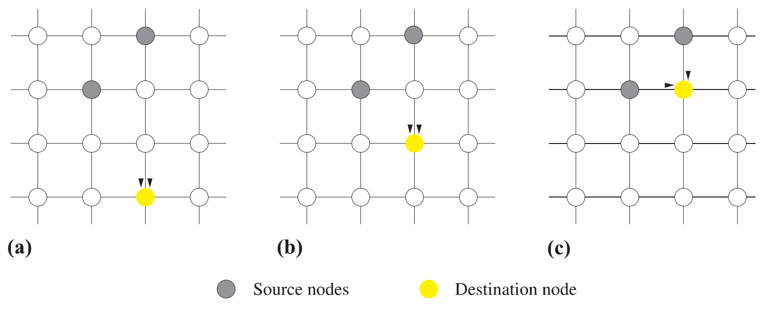


Fig. 1. An example of the effects of the reconfiguration algorithm.

## *2.1. The adopted reconfiguration technique*

There are several ways of reconfiguring an interconnection network. For example, the reconfiguration process can affect either the whole or part of the network. It does not necessarily alter the topology of the network, and can be controlled in a centralized or distributed way. Although reconfiguration techniques can be very different, we think that all of them should consider a series of aspects intrinsic to any reconfiguration process. The decision about each of the properties that characterize the reconfiguration technique will give rise to different forms of applying it. We now indicate the reconfiguration technique we have adopted, justifying the election for each one of the aspects that characterizes it.

*Reconfiguration scope*: Our design of the reconfigurable network allows both *local* and *global reconfigurations*. Local reconfiguration affects only the nodes that participate in a change, and possibly the nodes directly connected to them. Global reconfiguration can simultaneously modify the connections in multiple nodes located at different points in the network. As global reconfiguration affects a larger area of the network, it may be thought that it involves a higher cost. However, the final result of applying global reconfiguration depends on other factors, for example, the possible benefits of performing several changes simultaneously. Diverse tests have been carried out for determining the more appropriate option. The results will be presented in Section 4.5. Out of these results it can be seen that a better performance is obtained when local reconfiguration is adopted.

*Alteration of the topology*: We have maintained the topology at every moment, independently of the characteristics of the changes carried out. The main reason for this is to be able to use the same routing algorithm, besides obtaining a much simpler reconfiguration algorithm.

*Reconfiguration triggering*: A cost function is used to determine when the reconfiguration process is activated. A node determines whether it is convenient to exchange its position by taking into account the information that it receives about contention in the network. This information is obtained from the messages arriving at it. Each node records the time a message has been blocked in the network, and the number of channels that it has occupied. Both of these factors can be used to estimate the loss of network bandwidth due to congestion.

For a given node $D$ and a given channel $j$, the cost function takes the following expression:

$$\mathrm{CF}_j = \sum_{\text{messages}} \left( \sum_{b=S}^{D} t_b * c_b \right),$$

where $t_b$ is the time that a message arriving at node $D$ has been blocked at each node in its path $(S, \ldots, D)$, and $c_b$ is the number of channels occupied during $t_b$.

*Reconfiguration distance*: Basically, there are two strategies for carrying out the possible changes: *short distance* and *large distance* exchanges. In a short distance strategy, a node can only exchange its position with one of its neighbouring nodes. In the second case, a node can exchange its position with any other node in the network. As a result, fewer exchanges are needed to place a node at its optimal position, but it can produce abrupt and less uniform changes than those obtained from the short distance strategy.

In our case, the cost function provides information about the contention experienced by incoming messages, that is, about the direction in which the change must take place. As a result, the changes will take place among nodes that occupy neighbouring positions in the network, and therefore the short distance strategy will be used.

*Reconfiguration start*: Reconfiguration is started by some of the nodes in the network when the required conditions have been met. The conditions can be evaluated at fixed or variable time intervals. In the first case, the testing process takes place at regular intervals during the execution of the application. In the second case, the process is automatically activated at a node when a fixed number of messages arrives or leaves. We have selected the second option: a node begins the reconfiguration process after receiving a given number of messages.

*Thresholds in the reconfiguration*: A high number of reconfigurations can reduce system performance due to the reconfiguration overhead. In contrast, a small number of reconfigurations leads to slow systems, where the benefit obtained with the reconfiguration of the network will be diminished. With an adequate number of thresholds, the algorithm, and consequently the reconfiguration, will exhibit a better behaviour.

Our algorithm uses two thresholds: The first one indicates the conditions that must be satisfied so that the reconfiguration begins at a given node. The other one represents the minimum conditions to be met so that the processed information can be considered.

*Reconfiguration control*: There are two methods for controlling the dynamic reconfiguration of the network: *centralized control* (a particular node – system controller – is responsible for reconfiguring the network) or *distributed control* (each node controls its own reconfiguration). As a consequence of the system architecture (Section 2.4), the reconfiguration control is centralized.

## 2.2. The reconfiguration algorithm

The reconfiguration process is split into two phases: the study of the state of the network and the preparation–execution of the changes in the connections. To observe the state of the network, a set of actions in each of the nodes of the system is periodically executed. This is the *reconfiguration algorithm*. The second phase represents the *reconfiguration protocol*, and consists of the communication maintained by the nodes that are affected by the changes, and the actions that finally produce those changes.

In the description of the algorithm, we distinguish the part of it that is executed at each node in the system, and the part that is executed at the controller of the system. The first part consists of a group of actions that will be executed at every node, with the objective of analysing its particular state, and consequently, determine whether a change should be performed. The second part is executed by the node that acts as controller of the system. Each node considers only local information.

The reconfiguration algorithm for each *node* is as follows:

```
record_information(message);
message_counter++;
if (checkpoint(message_counter,threshold1));
{
  continue = process_information(threshold2);
  if (continue)
    {
      node_change = select( );
      change = check_change(node_change);
      if (change) reconfigure(node,node_change);
    }
}
```

where

*threshold1* indicates the number of messages that must be received at a node for initiating the reconfiguration process,
*checkpoint* establishes the starting points of the reconfiguration process study,
*threshold2* represents the contention level that triggers the reconfiguration process,
*process_information* evaluates the information recorded in the node in order to determine whether it is convenient to exchange its position in the network,
*select* determines a more appropriate position for its new location,
*check_change* examines the suitability of the change. It takes into account the *node_change* situation, and
*reconfigure* starts the reconfiguration protocol, which will exchange the positions of the affected nodes.

The following actions will be executed in the system controller:

```
receive(change_data);
change_connections;
broadcast(new_configuration);
```

where

*change_data* indicates the nodes that want to exchange their positions,
*change_connections* establishes the new connections in the network in agreement with the data for the requested change, and
*new_configuration* indicates the new node locations in the network.

The basic idea is the following: when messages arriving through a given channel at their destination nodes are significantly delayed, the reconfiguration process will try to move the destination node close to the site that is producing the delays, by exchanging its position with a neighbour closer to the congested zone. The algorithm works in the following way:

1. Each time a message arrives at its destination node, information about contention found along its path is recorded. It must be taken into account that there is a threshold for the contention value (*threshold2*). This threshold prevents one message, which may have experienced a very large contention, from starting the reconfiguration process.
2. The algorithm checks the state of the node each time the number of messages arriving at the node is a multiple of *threshold1* value.
3. If the contention in one channel is greater than the sum of the rest of the channels, the node sends a message to its neighbouring node through that congested channel to indicate the convenience of exchanging their positions.
4. If the change is beneficial, the network carries out the reconfiguration protocol.

The changes are not carried out in all the cases in which a reconfiguration process begins. When this occurs, the controller actions will consist of miscarrying the process and of communicating it to the affected nodes.

### 2.3. The reconfiguration protocol

The reconfiguration protocol describes the messages necessary to complete each reconfiguration process, and the required actions to produce the physical changes in the structure of the network. We have defined a reconfiguration protocol that adds a small amount of message traffic to the network. The main steps in this protocol are the following:

1. When a pair of nodes decide that it is necessary to reconfigure the network, both nodes inform all their neighbours that they are going to exchange their positions, and therefore, those nodes should stop sending messages to them.
2. The nodes that want to exchange their positions (one of them) send the reconfiguration data to the control node in order to carry out the reconfiguration.
3. When the affected area has reduced its activity until no messages circulate through it, the control node establishes the new connections in the network, adapting it to the traffic distribution.
4. Once the new configuration has been established, the control node sends information about the new node locations to the other nodes. Then, the pair of nodes that have exchanged their positions permit their neighbouring nodes to communicate with them again.

The changes may not to be carried out in all the cases in which a reconfiguration process begins. Some circumstances can arise forcing this process to be aborted, e.g., when excessive time is spent in the reconfiguration protocol. In this case, the reconfiguration protocol will communicate this fact.

In order to free the area of the network affected by a change from messages, two techniques can be adopted: storing the messages in the intermediate nodes, or allowing these messages to continue along their path until they abandon the changing area. As we are considering pipelined networks with small input and output buffers, the first option would consist of storing the message in the local memory of the nodes, significantly increasing message latency. The option of allowing the messages to continue towards their respective destinations can lead to difficulties if there is a lot of congestion in the network. These messages may take a lot of time to leave the area affected by the change. The situation may become even more complicated if for any reason the messages are wrapped in a deadlock. In this case, some strategy to avoid such situations should be used.

We have allowed the messages crossing the changing area to continue their path until they abandon it. This decision is based on the simulation results. These results will be presented in Section 4.5.

### 2.4. Implementation details

In our model, a dynamically reconfigurable system consists of several nodes, a link switch that allows communication among nodes, a system controller that supervises the switch configuration, and a bus used for reconfiguration control (Fig. 2).

The implementation of our reconfigurable architecture model must take into account several important details. In order to establish a connection in the switch, a configuration request message is sent by a node through the control bus to the switch controller. When possible, the required link connection is established in the switch. Then, according to the reconfiguration protocol described in the previous section, nodes requesting the connection change are acknowledged by appropriate messages.

We have studied two kinds of switches: a crossbar and an Omega multistage network [13]. The former is suitable for very small systems, up to 64 nodes. The latter is suitable for larger system sizes, but the reconfiguration capacity is much smaller than in the case above. The characteristics of these networks are such that the transmission time from an input port to an output port is significant. This does not enable, for example, a performance similar to the one obtained with direct networks. In order to improve the performance of the reconfigurable system, we have used *channel pipelining* [16]. In this way, the negative effect of the transmission time across the switch has been reduced [31].

In a pipelined channel network, data are clocked onto the wires at a rate determined solely by the switching speed [16], allowing multiple flits

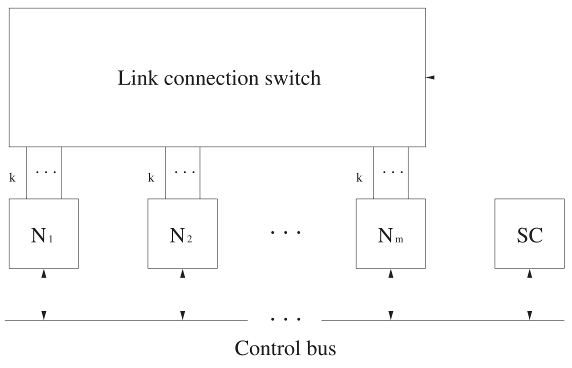


Fig. 2. General system structure for dynamic link connection reconfiguration.

to be simultaneously in flight on a single wire. In a multistage network, for example, several flits may be in flight simultaneously from an input port to an output port, along the path established through the different stages in the network.

Pipelined channels are very common in wide area networks and local area networks [5]. In this work, we have used a flow control protocol based on control flits. In particular, we have used the *Stop* and *Go* protocol, very much like in Myrinet networks [6]. The buffer size has been accurately calculated in order to avoid the loss of flits or the appearance of bubbles in the message pipeline.

## 3. Deadlocks in reconfigurable wormhole networks

Deadlocks can occur in many different situations. A deadlock occurs as a result of cyclic waits for resources by two or more messages. A cyclic wait can occur when a message is allowed to hold the resources allocated to it while waiting for other resources to become available. In store-and-forward and virtual cut-through networks, the resources are buffers. In wormhole networks, the resources are channels [10,11].

Deadlocks can also appear in reconfigurable wormhole networks, as shown in Fig. 3. In this example, we assume a deterministic routing strategy in which channels are allocated to messages in increasing order. The topology is $k$-ary $n$-cube. We used the following scenario: four messages ($M1$, $M2$, $M3$, and $M4$) and their destinations ($D1$, $D2$, $D3$, and $D4$), respectively.

When the four messages move towards their destinations, two of these destinations ($D1$ and $D3$) change their positions in the network. When the header flits off messages $M1$ and $M3$ arrive at nodes $S2$ and $S4$, respectively; they must change their normal trajectory to reach the new positions of their destinations. Finally, the situation as shown in Fig. 3(b) is reached. In this figure, it can be seen how each message holds a buffer while requesting the buffer held by another message.

It can be seen that a deadlock occurs when some messages are routed in such a way that the strategy used for channel allocation (that is, increasing dimension order) cannot be satisfied. This

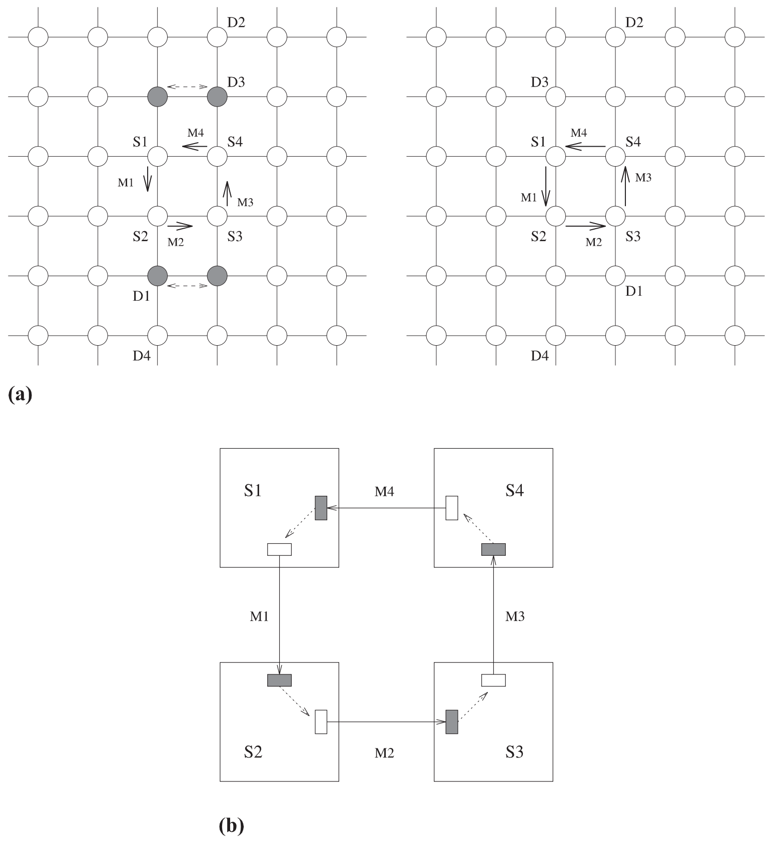


Fig. 3. An example of channel deadlock involving four messages.

situation occurs as a consequence of the changes of node location in the network. Therefore, we cannot guarantee the absence of these situations even if a deadlock-free routing algorithm is being used.

In order to recover from deadlock, a simple solution has been adopted. It consists of storing whole messages involved in a potential deadlock in the local memory of the node where their header flits are blocked. This frees the channels that those messages were using, so that other messages can advance. The messages that have been stored in the local memory will be re-injected when the channels they need become free. This solution introduces an important increment in latency. However, we have chosen this strategy, as our reconfiguration algorithm produces a small number of changes in the network, and all the changes do not necessarily produce deadlock. Through simulation, we have found that potential deadlocks occur in approximately 40% of the changes. As the number changes is small, the increment in latency is not significant.

## 4. Performance evaluation

In this section, we evaluate our network reconfiguration strategy. We study the behaviour of the reconfigurable network in two different situations. In the first, network traffic is produced by the simulated execution of a parallel algorithm. We have chosen a numerical algorithm. In the second, a non-uniform traffic model has been used. We analyse the effect of our reconfiguration strategy on the performance of the network with one or several hot-spots. The evaluation methodology we have used is based on the one proposed in [11] and it is summarized in the following sections.

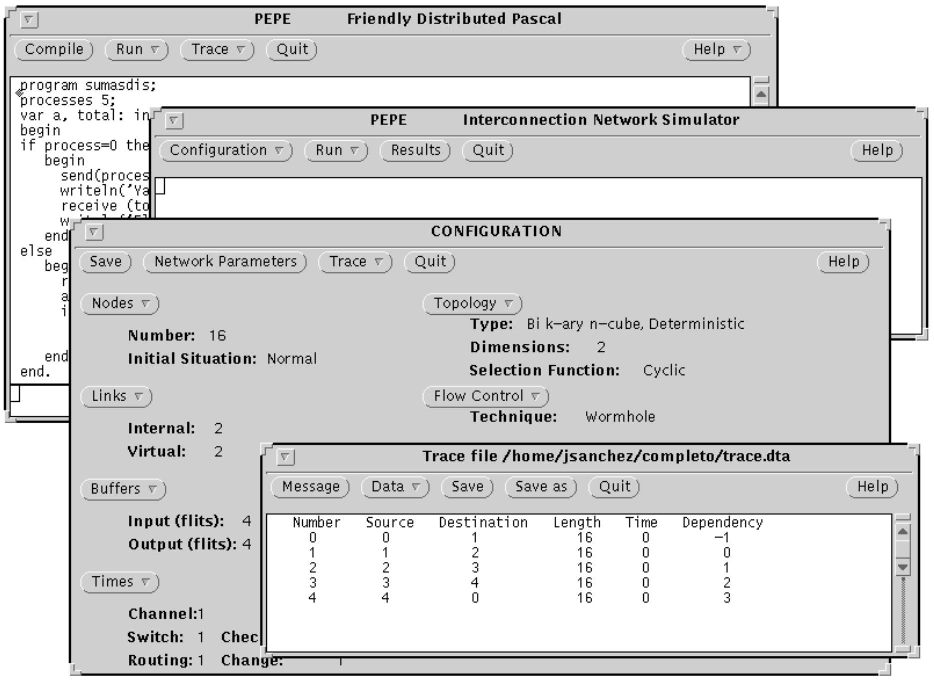
### *4.1. Programming and simulation environment*

The results have been obtained with the PEPE environment [26], a programming and evaluating tool for multicomputers. The key features of this environment are simplicity and completeness. Our main goal has been to obtain a flexible system that enables an easy and efficient way to evaluate the multicomputer reconfigurable architecture.

This environment has been developed on a workstation using C language. PEPE provides a user-friendly visual interface for all the phases of parallel program development and tuning. This graphical interface (Fig. 4) has been designed with the aim of making it comfortable for the user, following the styles currently adopted by most human-oriented interfaces. In our environment, the user gets tools for easy experimentation with both different parallelization possibilities and different network parameters. With this methodology, the programmer can analyse very quickly several parallelization strategies and evaluate these strategies with tools for performance analysis.

PEPE simulates the execution of a parallel algorithm at two levels: At the first level, the behaviour of the parallel algorithm can be studied. To observe this behaviour, its execution is simulated over a virtual architecture. At the second level, the behaviour of the reconfigurable network can be studied. To do this, we start with the communication pattern produced by the parallel algorithm and the performance of the network for this communication pattern is simulated. At this level, PEPE allows us to use different parameters for the reconfigurable network. These levels conform to the two phases of the simulator.

PEPE has two main phases and several modules within it: The first phase is more language-oriented, and allows us to code, simulate, and optimize a parallel program. In this phase, interactive tools for specification, coding, compiling, debugging, and testing have been developed. This phase is independent of the architecture. The second phase



```
program sumasdis;
processes 5;
var a, total: in
begin
if process=0 the
   begin
     send(proces
     writeln('Ya
     receive (to
     w
   end
else
   beg
     r
     a
     i


   end
end.
```

**Number:** 16
**Initial Situation:** Normal
**Internal:** 2
**Virtual:** 2
**Input (flits):** 4
**Output (flits):** 4
**Channel:**1
**Switch:** 1 **Chec**
**Routing:** 1 **Change:**

**Type:** Bi k–ary n–cube, Deterministic
**Dimensions:** 2
**Selection Function:** Cyclic
**Technique:** Wormhole

```
Number   Source   Destination   Length   Time   Dependency
  0        0          1           16       0       -1
  1        1          2           16       0        0
  2        2          3           16       0        1
  3        3          4           16       0        2
  4        4          0           16       0        3
```

Fig. 4. PEPE's graphical interface.

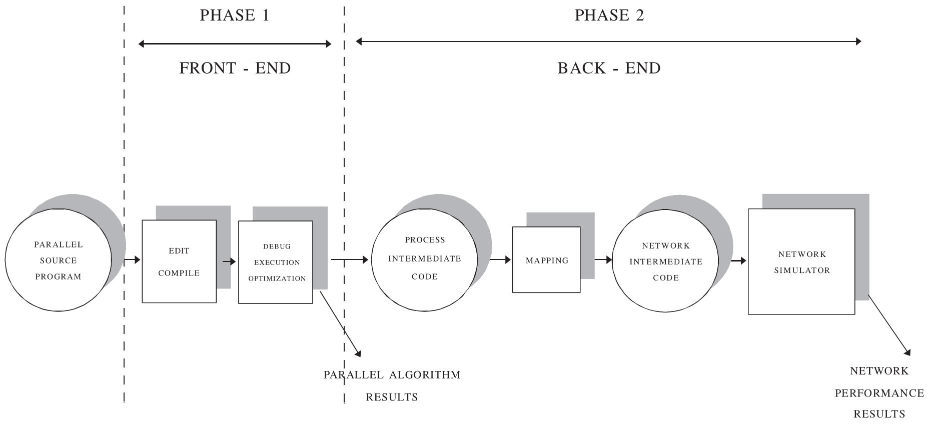

PHASE 1
FRONT - END
PHASE 2
BACK - END
PARALLEL SOURCE PROGRAM
EDIT COMPILE
DEBUG EXECUTION OPTIMIZATION
PROCESS INTERMEDIATE CODE
MAPPING
NETWORK INTERMEDIATE CODE
NETWORK SIMULATOR
PARALLEL ALGORITHM RESULTS
NETWORK PERFORMANCE RESULTS

Fig. 5. Modules of PEPE.

has several tools for mapping and evaluating the reconfigurable architecture. We could vary several network parameters, such as interconnection topology and routing algorithm. In Fig. 5, we show the modules of PEPE. The link between the phases is an intermediate code that is generated as an optional result of the first phase. This intermediate code is a trace of the communication pattern for the source program. This allows the user to manage the environment as a whole or each individual phase. For example, we could only execute the first phase for testing the parallel behaviour of an algorithm on an ideal multicomputer. We could also obtain an intermediate code from a key parallel algorithm. The second phase could then be executed several times from this trace with different network parameters to evaluate and tune the network for this key algorithm.

### 4.2. Message generation

We have considered two different traffic models: communication pattern produced by the simulated execution of a parallel numerical algorithm and hot-spots [15] workload. Both of them are described below.

#### 4.2.1. Triangularization of a sparse matrix

We have used a parallel application such as the triangularization of a sparse matrix using Fast Givens [3] rotations. The Fast Givens rotations algorithm produces communication among the different nodes of the multicomputer, which is mainly due to the movement of rows in the matrix from one process to another. It is therefore necessary to reduce the negative effect, due to this communication, on the total execution time of the algorithm.

This algorithm has been chosen because we cannot know a priori the communication pattern among nodes as it depends on the structure of the sparse matrix. Consequently, a suitable topology cannot be selected [30]. Moreover, the communication pattern will vary over time.

#### 4.2.2. Hot-spots

The situation that has been simulated is the following: let us consider a multicomputer with a uniform distribution of message destinations. At a given moment, and with the network in a steady state, the communication pattern changes and a small number of hot-spots appear in the network. This situation is repeated with variable frequency and the hot-spots are different every time.

Hot-spots produce congestion in the network, due to the great number of messages destined for the hot-spot nodes. This increases delays, thereby degrading the performance of the network. Our objective is to verify the behaviour of a reconfigurable network under this workload model.

### 4.3. Performance measures

The most important performance measures are delay and throughput. Delay is the additional latency required to transfer a message with respect to an idle network. It is measured in clock cycles. The message latency lasts from the moment the message is introduced in the network until the last flit is received at the destination node. An idle network means a network without message traffic. Throughput is usually defined as the maximum amount of information delivered per unit of time. It is measured in flits per clock cycle.

### 4.4. Simulation parameters

We have evaluated the performance of the reconfiguration algorithm on 8-ary 2-cube and 16-ary 2-cube networks. The deterministic algorithm proposed in [10] for the $k$-ary $n$-cube has been used. It has been modified so that it uses bidirectional channels with two virtual channels per physical channel. To compute the clock frequency for each node, we have used the delay model proposed in [9]. The router takes 4.7 ns to compute the output channel. The switch takes 4.8 ns to transfer a flit through the crossbar, and the time required to transfer a flit across a physical channel is 6.8 ns. We have also considered that the useful throughput of the bus is 1 MB ps, and that the token, moves quickly (50 ns per node). Finally, each switch reconfiguration takes 2 $\mu s$.

The proportion of messages at the hot-spot has been varied between 5% and 20%. For each simulation run, we have considered that the message generation rate is constant and the same for all the nodes. Each simulation was run until the network reached a steady state, that is, until a further increase in simulated network cycles did not change the measured results appreciably. Once the network has reached a steady state, the flit generation rate is equal to the flit reception rate (traffic). The number of hot-spots has been 1 and 2, chosen randomly. Finally, 16-flit and 128-flit messages have been considered.

We have considered a crossbar switch to connect the nodes in the system and, therefore, the time required to transfer a flit from an input port to an output port has been 13.2 ns. This time has been 17.3 ns when Omega multistage network has been considered. Input buffer size has been 21 flits. We have considered four flits as the output buffer size. For other values, different from the above parameters, the result tendency is maintained.

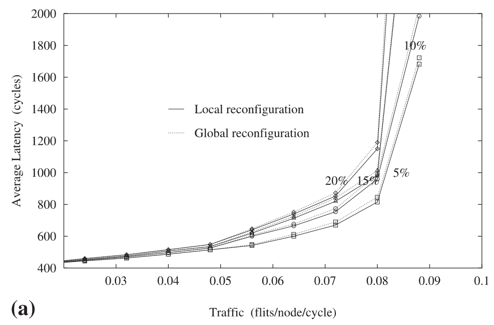
### 4.5. Simulations results

In this section, we present the main results obtained from the evaluation of the reconfigurable network. These results are presented in two separated groups. Firstly, we have included those results which allowed us to completely tune the proposed reconfigurable network. The second group of plots shows the performance evaluation results.

#### 4.5.1. Reconfiguration tuning

As indicated in a previous section, our design of the reconfigurable network allows both local and global reconfiguration. We have performed several tests to check how our system works in each case. In Figs. 6 and 7, we can see the results obtained in both cases. It may be thought that performing several reconfiguration processes at the same time could lead to better behaviour, as the nodes would move faster to the most appropriate locations. However, note that for a reconfiguration process to conclude successfully, the area affected by the change should be completely inactive. Therefore, no message should cross this zone. This causes significant congestion in the network for a given interval of time. Congestion will increase considerably when several areas, corresponding to several changes, are inactive. Therefore, local reconfiguration is preferable.

On the contrary, in order to free the area of the network affected by a change from messages, we have considered two alternatives, as mentioned in Section 2.3. In the first one, we have removed all the messages from the channels belonging to the affected area, storing them in the intermediate nodes where their header flits are. Once the change has finished, the messages will again be injected into the network and sent towards their destinations. This injection will have priority over the



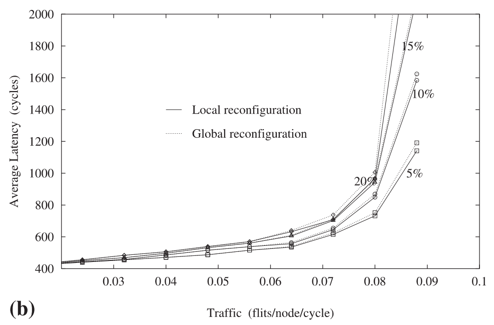
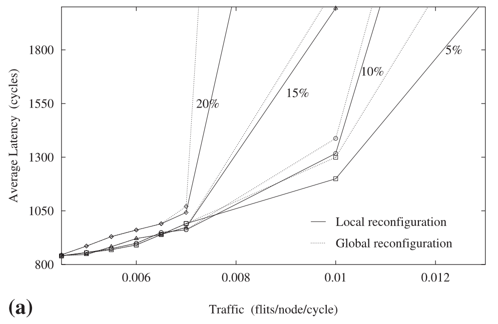


Fig. 6. Average message latency vs. traffic for 8 × 8 torus and Omega switch (a) One hot-spot, (b) Two hot-spots.



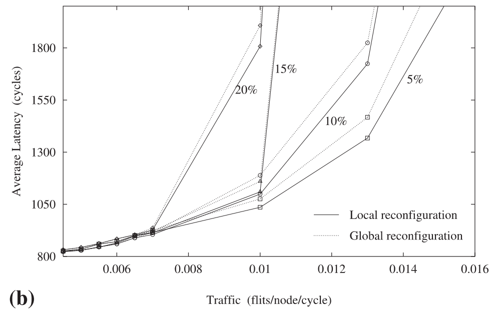


Fig. 7. Average message latency vs. traffic for 16 × 16 torus and Omega switch (a) One hot-spot, (b) Two hot-spots.

transmission of new messages. Another option consists of allowing the messages crossing the changing area to continue until they abandon it. In both cases, other messages will be prevented from entering the affected area, and will not be assigned the channels that they request.

Using the first option will enable the affected channels to be removed faster than by using the second option. However, the latency of the stored messages will increase. The result of applying both techniques on a reconfigurable system is shown in Fig. 8. From this figure, it can be seen that the second option offers better results. The drawback of the storage technique is the increase of the latency due to the temporary storage of messages in intermediate nodes.

### 4.5.2. Performance results

Once the reconfiguration technique has been tuned, we show the performance evaluation results. To obtain the plots shown below, several parameters were varied. In the case of trace workload, the parameters were load size (Figs. 9(a) and (b)) and the way of distributing processes among processors (Fig. 10(a)). For synthetic traffic load, the number of hot-spots and the traffic (Fig. 10(b)) were varied.

As mentioned earlier, we have used an indirect network to design the reconfigurable system architecture. In Fig. 9(a), the reconfiguration effect on the proposed system is shown (non-pipelined channel network with reconfiguration, (NPWR), versus non-pipelined channel network non-reconfiguration, (NPNR)). Note that the simulation time is reduced for all the load sizes. This reduction has reached a value of 15% in some cases. The changes in the network made it possible to locate the nodes of the system in more appropriate locations compared to their previous locations. Consequently, message delay is reduced. We

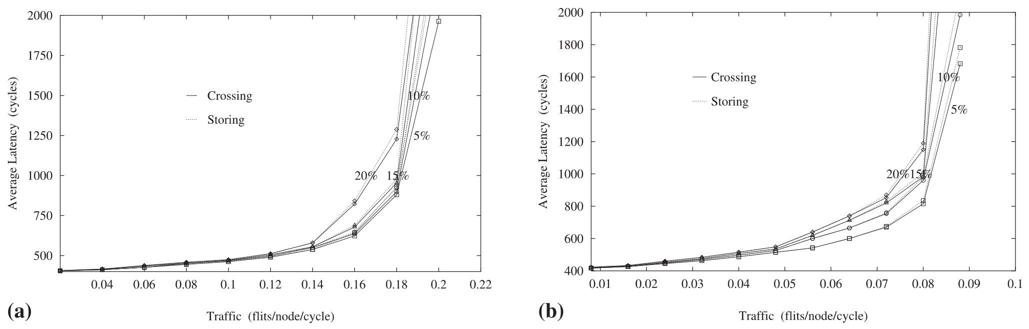


Fig. 8. Average message latency for 2-D torus, one hot-spot and crossbar switch (a) 16 nodes, (b) 64 nodes.

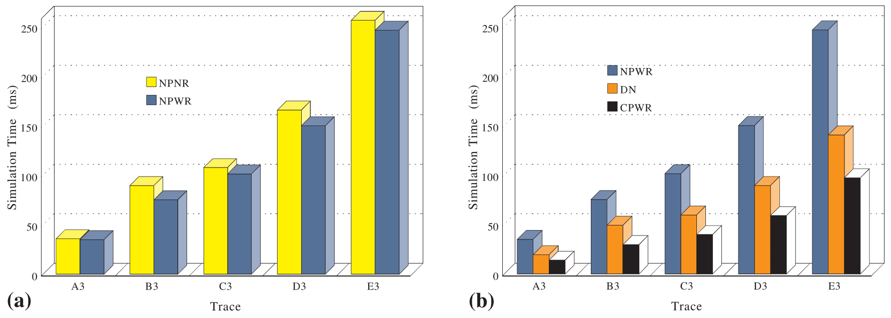


Fig. 9. (a) Reconfiguration and (b) Channel pipelining effects vs. load size for 2-D torus with 64 nodes.

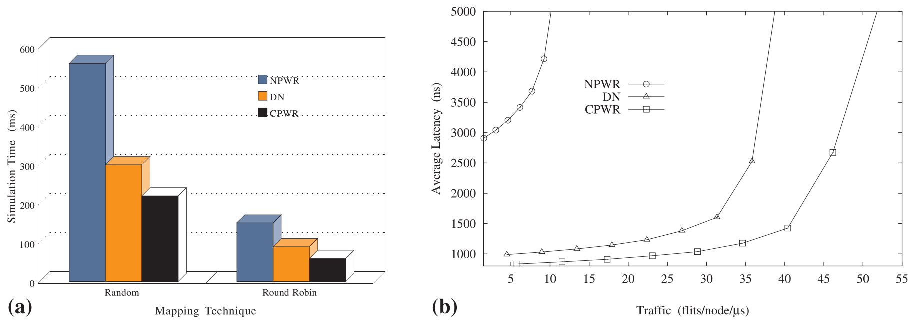


Fig. 10. Channel pipelining effects vs. mapping and traffic (a) $8 \times 8$ 2D torus and trace D3 (b) 2D torus with 64 nodes, one hot-spot and 20% non-uniform component.

would like to emphasize that this improvement is produced with a small number of changes in the network (approximately 30 changes).

Even though these results are significant, it is important to compare the behaviour of the system with a direct non-reconfigurable network (DN). The rest of the figures show some of these results in addition to those obtained by applying a channel pipelining technique with reconfiguration (CPWR) on the proposed system.

Two observations can be made from all the plots. First, an indirect reconfigurable network does not achieve the performance offered by a direct one. The reason is the overhead produced by the switching layer in order to allow reconfiguration. Second, the application of the channel pipelining technique to the proposed system significantly improves performance over direct networks. These improvements are maintained even when using different distributions of processes among processors, as can be seen in Fig. 10(a).

## 5. Conclusions

In this article, we have presented a dynamic reconfiguration technique for wormhole networks, and have proposed a general reconfiguration algorithm that decides when and how the reconfiguration takes place. We have presented a reconfigurable network model, and featured the reconfiguration technique supported by it. We have analysed the capabilities of several types of configurations, using a unique crossbar or by means of a multistage network. The performance of these models has been analysed by simulation.

As can be observed in the presented figures, the effect of the reconfiguration is positive and, for a trace workload, it is possible to reduce the simulation time to 15% in some cases, without needing a large number of changes. Moreover, it is an important fact that this reduction is independent of the way in which the processes are distributed among the processors.

We have compared these results with those obtained by considering a system with a direct interconnection network. In this sense, we have checked that the improvements pointed out previously are not enough to recommend the use of our approach initially. The characteristics of the crossbar or Omega multistage, that should establish the connections among the nodes in the system impose strong restrictions, reflected in high transmission times, superior to those that offer point to point connections. This is the main reason that prevents the proposed system from reaching levels of similar or superior performance to those that offer classic multicomputers based direct networks.

However, the characteristics of this kind of system enables the technique of channel pipelining to be applied. The results obtained are much better when this technique is applied. This technique enables to reach, using indirect networks, a throughput closer to direct networks. If these networks could be reconfigurated, the results in some situations could be improved.

Although the reconfiguration capacity of the Omega multistage is much smaller than the crossbar, the behaviour of the system is similar in both cases when the channel pipelining technique is applied. The necessity to use an Omega network as the interconnection device for systems of larger size does not produce excessive reduction of the level of benefits in comparison with the crossbar.

## References

[1] D.A. Buell, J.M. Arnold, W.J. Kleinfelder, Splash 2: FPGAs in a Custom Computing Machine, IEEE Computer Soc. Press, Silver Spring MD, 1996.

[2] Ch. Fraboul, J.Y. Rousselot, P. Siron, Software tools for developing programs on a reconfigurable parallel architecture, in: D. Grassilloud, J.C. Grossetie (Eds.), Computing with Parallel Architectures: T. Node, Kluwer Academic Publishers, Dorderecht, 1991, pp. 101–110.

[3] G.H. Golub, C.F. Van Loan, Matrix computations, North Oxford Academic, 1983.

[4] D.A. Nicole, Reconfigurable transputer processor architectures, in: T.J. Fountain, M.J. Shute (Eds.), Multiprocessor Computer Architectures, North-Holland, Amsterdam, 1990.

[5] A.S. Tanenbaum, Computer Networks, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1988.

[6] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J. Seizovic, W. Su, Myrinet – A Gigabit per Second Local Area Network, IEEE Micro, 1995, pp. 29–36.

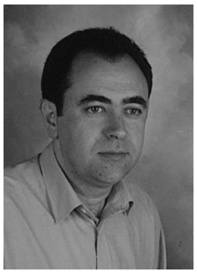
[7] S.H. Bokhari, Finding maximum on an array processor with a global bus, IEEE Trans. Comp. C-33 (2) (1983) 133–139.

[8] Ben-Asher, D. Peleg, R. Ramaswami, A. Schuster, The power of reconfiguration, J. Parallel and Distributed Comput. 13 (1991) 139–153.

[9] A.A. Chien, A cost and speed model for *k*-ary *n*-cube wormhole routers, IEEE Trans. Parallel and Distributed Sys. 9 (2) (1998).

[10] W.J. Dally, C.L. Seitz, Deadlock-free message routing in multiprocessor interconnection networks, IEEE Trans. Computers C-36 (5) (1987) 547–553.

[11] J. Duato, A new theory of deadlock-free adaptive routing in wormhole networks, IEEE Trans. Parallel and Distributed Systems 4 (12) (1993) 1320–1331.
[12] P. Kermani, L. Kleinrock, Virtual cut-through: a new computer communication switching technique, Computer Networks 3 (1979) 267–286.
[13] D.H. Lawrie, Access and alignment of data in an array processor, IEEE Transactions on Comput. C-24 (12) (1975) 1145–1155.
[14] R. Miller, V.K. Prasanna Kumar, D. Reisis, Q.F. Stout, Parallel computations on reconfigurable meshes, IEEE Trans. Comp. 42 (6) (1993) 678–692.
[15] G.F. Pfister, V.A. Norton, Hot spot contention and combining in multistage interconnect networks, IEEE Trans. Comp. C-34 (10) (1985) 943–948.
[16] S.L. Scott, J.R. Goodman, The impact of pipelined channels on *k*-ary *n*-cube networks, IEEE Trans. Parallel and Distributed Sys. 5 (1) (1994) 2–16.
[17] Q.F. Stout, Mesh connected computers with broadcasting, IEEE Trans. Comp. C-32 (1983) 826–830.
[18] J. Adamo, C. Bonello, Tenor++: a dynamic configurer for supernode machines, in: Lecture Notes in Computer Science, vol. 457, Springer, Berlin, 1990, pp. 640–651.
[19] R. Amerson, R.J. Carter, W.B Culbertson, P. Kuekes, G. Snider, Teramac-configurable custom computing, in: IEEE Symposium on FPGAs for Custom Computing Machines, 1995, pp. 32–38.
[20] R. Bittner, P. Athanas, Wormhole run-time reconfiguration, in: Proceedings of the ACM International Symposium of FPGAs, 1997, pp. 79–85.
[21] A. Bauch, R. Braam, E. Maehle, DAMP: a dynamic reconfigurate multiprocessor system with a distributed switching network, in: The Second European Distributed Memory Computing Conference, Munich, April 1991, pp. 495–504.
[22] F. Desprez, B. Tourancheau, Evaluation des performances de la machine Tnode, La lettre du Transputer, LIB Besacon, No. 7, 1990.
[23] C. Ebeling, D.C. Cronquist, P. Franklin, RaPiD-reconfigurable Pipelined Datapath, in: Proceedings of the Sixth International Workshop on Field-Programmable Logic and Applications, 1996.
[24] J.M. García, J. Duato, An algorithm for dynamic reconfiguration of a multicomputer network, in: Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, 1991, pp. 845–855.
[25] J.M. García, J. Duato, Dynamic reconfiguration of multicomputer networks: limitations and tradeoffs, in: P. Milligan, A. Nuñez (Eds.), Euromicro Workshop on Parallel and Distributed Process, IEEE Computer Soc. Press, Silver Spring, MD, 1993, pp. 317–323.
[26] J.M. García, J.L. Sánchez, P. González, Pepe: a trace-driven simulator to evaluate reconfigurable multicomputer architectures, in: Lecture Notes in Computer Science, vol. 1184, Springer, Berlin, 1996, pp. 302–311.
[27] E. Mirsky, A. Dehon, MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources, in: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, April 1996.
[28] R. Miller, V.K. Prasanna Kumar, D. Reisis, Q.F. Stout, Meshes with reconfigurable buses, in: Proceedings of the MIT Conference on Advanced Research, VLSI, January 1988, pp. 163–178.
[29] D. Reisis, V.K. Prasanna-Kumar, VLSI arrays with reconfigurable buses, in: Proceedings of the International Conference on Supercomputing, 1987.
[30] J.L. Sánchez, J.M. García, J. Fernández, Improving the performance of parallel triangularization of a sparse matrix using a reconfigurable multicomputer, in: Lecture Notes in Computer Science, vol. 1041, Springer, Berlin, 1996, pp. 493–502.
[31] J.L. Sánchez, J.M. García, J. Duato, Using channel pipelining in reconfigurable interconnection networks, in: The Sixth Euromicro Workshop on Parallel and Distributed Processing, IEEE Computer Soc. Press, Silver Spring MD, 1998, pp. 120–126..
[32] Q.F. Stout, Meshes with multiple bus, in: Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science, 1986, pp. 264–273.
[33] J.M. García, Desarrollo de herramientas para una programación eficiente de las redes de transputers: estudio de la reconfiguración dinámica de la red de interconexión, Ph.D thesis, Universidad Politécnica de Valencia, 1991.



**Jose L. Sanchez** received the Ph.D. degree from the Universidad Politecnica de Valencia, Valencia, Spain, in 1998. In 1986 he joined the Departamento de Informatica at the Universidad de Castilla-La Mancha. He is currently an Assistant Professor of Computer Architecture and Technology. He teaches several courses on architecture and organization. His research interests include multicomputer systems, high-performance local networks, interconnection networks, parallel algorithms and simulation.

**Jose M. Garcia** was born in Valencia, Spain on 9 January 1962. He received the Ingeniero Industrial (Electrical Engineering) and the Ph.D. degrees from the Universidad Politecnica de Valencia, Valencia, Spain, in 1987 and 1991, respectively. In 1987, he joined the Departamento de Informatica at the Universidad de Castilla-La Mancha at the Campus of Albacete. From 1987 to 1993, he was an Assistant Professor of Computer Architecture and Technology. In 1994 he became an Associate Professor at the Universidad de Murcia (Spain). From 1995 to 1997 he served as Vice-Dean of the Facultad de Informatica (School of Computer Science). At present, he is the Director of the Departamento de Ingenieria y Tecnologia de Computadores (Engineering and Computer Technology Department), and also the Head of the Parallel Computing and Architecture Research Group. He has developed several courses on computer structure, peripheral devices, computer architecture and multicomputer design. His current research interests include cluster computing, meta-computing, parallel benchmarking, interconnection networks, parallel algorithms and simulation. He has published over 30 refereed papers in these fields. Dr. Garcia is a member of the IEEE Computer Society, the ACM, the Euromicro European Computer Society and also of the ATI Spanish Computer Society.