Using Channel Pipelining in Reconfigurable Interconnection Networks *

José L. Sánchez
Dpto. Informática
Escuela Politécnica
Univ. Castilla-La Mancha
Albacete, SPAIN 02071

José Duato
Dpto. Ing. Sist. Comp. Aut.
Facultad de Informática
Univ. Politécnica Valencia
Valencia, SPAIN 46071

José M. García
Dpto. Informática y Sistemas
Facultad Informática
Univ. Murcia
Murcia, SPAIN 30071

Abstract

The major problem in wormhole routing networks is related with the contention due to message blocking. Reconfigurable networks are an alternative to reduce the negative effect that congestion produces on the performance of the network. Our work is focused on dynamic reconfiguration. This technique consists basically of placing the different processors in the network in those positions which, at each computational moment and according to the existing communication pattern among them, are more adequate for the development of such computation. In a reconfigurable architecture, the clock period is determined by the transmission time across the switch. To increase this frequency the channel pipelined technique is used.

In this paper we present the foundations of reconfigurable network architecture. We show the general structure of the reconfigurable systems and we indicate the characteristics of the channel pipelining technique. Finally, we evaluate the performance of a reconfigurable system.

1. Introduction

In highly parallel machines, a collection of computing nodes works in concert to solve large application problems. The nodes communicate data and coordinate their efforts by sending and receiving messages through a routing network. Consequently, the achieved performance of such machines depends critically on the performance of their routing networks. Network performance depends on a variety of factors, not only of network architecture, but also the features of communication that is actually carried on the network such as message sizes and traffic patterns.

So, it is important to increase the performance of the network. To meet this objective, the designer manipulates three independent variables [3]: topology, routing and switching. The network topology defines how the nodes are interconnected by channels. The routing algorithm decides the path selected by a message to reach its destination node. Switching mechanism refers to the strategy used to regulate traffic in the network, that is, it must allocate the available resources to messages, and on the other hand, resolve the conflicts that may emerge as a consequence from this.

Many recent multicomputer networks use cut-through or wormhole routing [4, 9], a technique which reduces message latency by pipelining transmission over the channels along a message's route.

A major problem found in wormhole networks is blocking situations. In large parallel processing systems there are applications in which blocking situations can occur more easily. A blocking situation appears when the header flit cannot find a free link to move towards the destination and then it is blocked in an intermediate node. Consequently, the intermediate links that carry the remaining flits are blocked. This in turn may block other messages. Moreover, in these networks a message spans multiple channels which couples the channels tightly together, therefore blockage on one channel can have immediate impact on others. Tight coupling between channels means that one long message can block the progress of many other messages. In these networks, channel coupling effects make the performance quite sensitive to blockage problem. So, this congestion can reduce the network performance in a significant rate.

Several techniques have been proposed to reduce or avoid congestion, such as virtual channels, random routing or message combining. In this paper we propose a network reconfiguration mechanism in order to try to improve the performance of these systems. The goal of a reconfigurable network is to increase the performance by minimizing the congestion of messages in the network. This objective is achieved by means of changing the position of the nodes in the network that cause a large congestion due to the deterministic routing.

As it will be seen in section 3, the reconfigurable sys-

*This work was supported in part by Spanish CICYT under Grant TIC97-0897-C04-02

tem architecture is based on the use of indirect networks. In these networks, the time required to transmit a flit from an input port to an output port is very significant. So, the cycle time of the reconfigurable system is determined by the slowest stage, this is to say, the transmission time across the switch. The clock frequency must be reduced in order to transmit information properly through all the components. This decreases throughput and increases message latency. To reduce the negative effects of the slowest stages, the channel pipelining [12] can be used. In a pipelined channel network, several flits may be simultaneously in flight on a single wire. This technique is very common in computer networks and it makes clock frequency completely independent of wire length.

In this paper we present the foundations of reconfigurable network architecture. The reconfiguration capacity is based on a reconfiguration algorithm distributed in each node. The algorithm decides when and how the reconfiguration will take place. The algorithm evaluates the communication contention and decides when the reconfiguration is more favourable. This algorithm is based on a cost function and requires only local information.

The rest of the paper is organized as follows. In section 2 we introduce the reconfigurable network architectures and we present the algorithm for dynamic reconfiguration. In section 3 we show different implementation aspects. In section 4 we present the pipelined channel technique. In section 5 we show and analyze the evaluation results, and finally, in section 6 some conclusions are given.

2. Reconfigurable network architecture

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 a non-neighbour processor.

A reconfigurable network is adopted in order to reduce the cost of the communication. Basically, it consists of placing the different processors in the network in those positions which, at each computational moment and according to the existing communication pattern among them, are more adequate for the development of such computation.

A reconfigurable network has the important advantages [6]. Programming a parallel application becomes more independent of the target architecture because the architecture adapts to the application. This feature provides the flexibility required for an efficient execution of various applications. Moreover, in this way it is easy to exploit the locality in communications. In wormhole networks, reconfigurable architectures alleviate the congestion in due to the blocking problem. This problem is more important in networks with deterministic routing. Finally, there are applications which communications pattern varies over time. For these, reconfigurable architectures can be very well suited.

There are two types of reconfiguration: static or dynamic. In this paper, we focus on dynamic reconfiguration, that is, the topology can change almost arbitrarily at runtime.

Figure 1 shows the effects of this technique for a very elemental situation. We suppose a wormhole network. In (a) and (b) messages sent by processors must in some situations go through the same channels until they reach their destinations, originating delays in communication due to the blockage of the channels. Once situation (c) is reached, this problem disappears, thus accelerating the emission and the reception of these messages and improving the network performance.

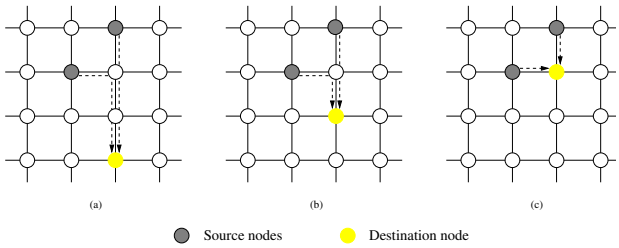

Figure 1. An example of the reconfiguration algorithm effects

The processor which receives the messages handles itself to place in its best position in the network and in every moment. This is achieved by changing the location of this processor by means of small alterations in the network, that is, exchanging its position with a neighbour processor.

2.1. The reconfiguration algorithm

A reconfigurable network is controlled by the reconfiguration algorithm. We show an updated version of the algorithm presented in [7]. This old version was designed to work on networks using the store-and-forward routing technique. Now, most multicomputers use wormhole routing. Therefore, we have developed a new version accounting all features of this routing algorithm.

The basic idea is the following: when messages arriving by a given channel to their destination nodes have supported an important delay, the algorithm will try to put the destination node close to the site that is producing those delays, by exchanging its position with its neighbour more close to the conflict zone. Then, 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 a limit exists for the contention value (*threshold2*). This limit prevents that only one message having a very large contention can perform the reconfiguration process. Moreover, it is required a minimum number of messages for the reconfiguration to begin (*threshold3*). Then, we can select an average contention and a number of messages received per channel in our algorithm.

2. The algorithm checks the state of the node each time that 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 channels, the node sends a message to its neighbour node through that congested channel to indicate the convenience of exchanging their positions.

4. If the change is adequate, the network carries out the reconfiguration protocol. Although the reconfiguration algorithm is distributed, the reconfiguration protocol needs a control node to perform the change.

Thus, the algorithm for the dynamic network reconfiguration has the following properties: *it preserves the topology* (after reconfiguration, the network has the same topology), *it is based on network contention* (a node can reconfigurate the network taking into account information about the contention in the network), *it produces a small alteration* (a node can only make an exchange with one of its neighbour nodes), *it uses three thresholds* for network reconfiguration.

A node determines the convenience of changing itself taking into account the information that it receives about contention in the network. This information is obtained from the messages arriving to it. Each node records the time through a message has been blocked in the network, and the number of channels that it has occupied. With both factors, blocked time and number of channels, we can estimate the loss of network bandwidth due to a congestion problem.

Then, for a given node D and a given channel j, the cost function takes the following expression:

$$FC_j = \sum_{messages} \left(\sum_{k=S}^{D} t_{b_k} * c_{u_k} \right)$$

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

2.2. The reconfiguration control

As it will be seen in the next section, the dynamically reconfigurable system consists of several nodes, a link connection switch which allows communication among nodes and a system controller which supervises the switch configuration.

We have chosen a centralized control for the dynamic network reconfiguration. A reconfiguration protocol among the nodes and the control node has been developed for handling the reconfiguration of the network. The main steps are the following:

1. When a pair of nodes decide that it is necessary to reconfigurate the network, both nodes inform all their neighbouring nodes that they are going to interchange their positions and therefore those nodes should stop sending messages to them.

2. The nodes which want to interchange their positions (one of them) send the reconfiguration data to the control node to carry out the reconfiguration.

3. The control node modifies the interconnection network topology, adapting it to the new circumstances.

4. Once the new configuration has been established, the control node sends information about the new situation to the other nodes. The pair of nodes that have interchanged their positions, permit their neighbouring nodes to communicate with them again.

This protocol adds little message traffic to the network. Next, we describe a feasible implementation of a reconfigurable network.

3. Implementation of reconfigurable networks

The dynamically reconfigurable system consists of several nodes, a link connection switch which allows communication among nodes, a system controller which supervises the switch configuration and a bus used for reconfiguration control (figure 2).

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 it is possible, the required link connection is established in the switch. Then, according to the reconfiguration protocol described in the prior section, nodes requesting the connection are acknowledged by means of the adequate messages.

The control bus is organized in a very simple way. A controller has been introduced, which acts as an interface between the nodes and the bus. This model permits, in parallel, receiving and servicing connection requests and acknowledges, due to autonomous functioning of node link interface. On the other hand, the node can continue with its normal process once it has sent the configuration request even though it has not the control of the bus.

The bus is composed of a data line and an acknowledge line. These lines are used to transmit the different messages

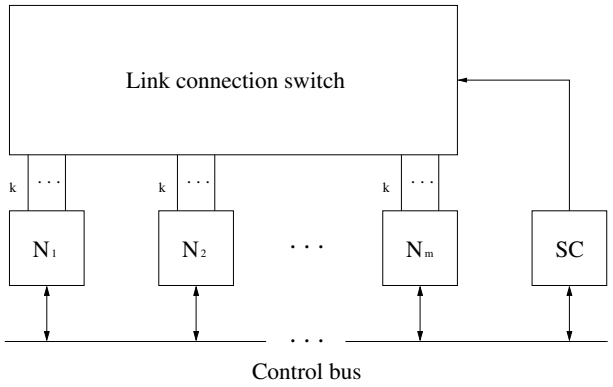

Figure 2. General system structure for dynamic link connection reconfiguration

involved in the reconfiguration process. Additional lines are provided by the controller for permitting the token-based access to the bus.

We have chosen this type of bus due to several motives: Firstly, it is very simple and it guarantees a absence of conflicts. Secondly, the maximum delay for each message transfer is predictable. Thirdly, the length of the wires is not excessive. And finally, the traffic through the bus is very small since this happens only in the control messages. This is the reason why the bus will not become the bottleneck of the system.

3.1. The switch

In our research we have studied two kinds of switches: A crossbar and a Omega multistage [10]. The former allows us to obtain very tiny size systems, up to 64 nodes. By means of the latter we will get larger size systems, being therefore the reconfiguration capacity much smaller than in the above case.

As it is shown in section 5, we are going to evaluate the behaviour of our reconfigurable system comparing its results with those obtained for a static network system. Having such a network a 2D torus topology and being both (static and reconfigurable networks) under the same workloads. Therefore, we must establish the adequate connections in the crossbar (o multistage) to obtain the 2D torus topology.

In general, to get the k-ary n-cube topology we need a crossbar with $2nk^n$ input and output ports, or an Omega multistage with $\log_r 2nk^n$ stages using $r \times r$ switches. It is clear that these values are very high for n and k values not too high, making it even more unviable the crossbar and not very advisable a multistage. In order to reduce this problem, and considering the k-ary n-cube features, we have split the global network in so many sub-networks as output channels there are in a node. This is to say, $2n$ for a k-ary n-cube with bidirectional channels, or n for unidirectional channels.

The advantages of this kind of split are clear: A crossbar with a smaller number of input and output ports is needed, as well as multistages formed of many less stages and switches. This means a shorter transmission time. Besides, the sub-networks split does not introduce any overhead in the reconfiguration process since this can is done simultaneously in all of them.

Even though, the characteristics of these networks are such that the transmission time from an input port to an output port is very significant. This fact doesn't permit to obtain, for example, performance similar to the one obtained with direct networks. In order to improve the performance of the reconfigurable system, we have applied channel pipelining and in this way it is reduced the negative effect of that important transmission time across the switch.

4. Channel pipelining

In a non-pipelined channel network, the cycle time includes the transmission time across the longest wire. Thus, clock frequency must be reduced in order to transmit information in an adequate way through all the links. The bandwidth is reduced and the message latency increased. This is more significant in the case of high-dimensional networks.

For reconfigurable systems, the reduction of the clock period is even larger. As it has been shown in the previous section, the architecture is based on indirect networks, which introduce a greater transmission time. However, the characteristics of this kind of systems permit to apply the technique of channel pipelining. In a pipelined channel network, data is clocked onto the wires at a rate determined solely by the switching speed [12], allowing multiple flits to be simultaneously in flight on a single wire. In a multistage, for example, several flits may be simultaneously in flight 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. There is an abundance of research regarding the different issues related to this technique [13]. In this work, we have used a protocol based on using control flits instead of dedicated ports to flow control task. In order to reduce the effect of these flits over the channel bandwidth the *Stop* & *Go* protocol has been used. This protocol, - implemented in Myrinet [1], for example -, is based on using input buffers quite larger than in a non-pipelined channel network. These buffers are divided into three parts separated by two watermarks: *Stop* and *Go*. When the buffer fills over the *Stop* mark, a *Stop* control flit is sent to the previous node. When this node receives that control flit, it stops sending more flits. As the input buffer empties, if the number of stored flits is below the *Go* mark,

the node sends a *Go* control flit to the previous node so that it restarts sending flits again.

The buffer size must be accurately calculated in order to avoid the loss of flits or the appearance of bubbles in the message pipeline. The part above the *Stop* mark must be able to store incoming flits that are in flight since the receiving node sends the *Stop* control flit until the previous node stops sending more flits. This period is twice the propagation time through the switch plus the time required to process the control flit. On the other hand, the part below the *Go* mark must be as large as the part above the *Stop* mark, in order to avoid that the input buffer empties before the flit flow is resumed after sending the *Go* control flit. Taking into account this, we have considered that input buffers have capacity for all the flits arriving during a period three times the maximum round trip delay. In this way, the central part is large enough to avoid sending control flits continuously.

5. Performance Evaluation

In this section we are going to evaluate our reconfigurable network model. The evaluation methodology used is based on the one proposed in [5]. The most important performance measures are latency and throughput. Latency is the required time to deliver a message. It is measured in nanoseconds. Throughput is usually defined as the maximum traffic accepted by the network, where traffic is the flit reception rate. It is measured in flits per node per microsecond. We have also taken into account the simulation time, this is to say, the time required for making a determined number of messages arrive to their destinations.

The results have been obtained with Pepe environment [8], a programming and evaluating tool for multicomputers and multiprocessors. Pepe has two main phases: the first phase is more language-oriented, and it allows us to code, simulate and optimize a parallel program. The second phase has several tools for mapping and evaluating the architecture. The main feature included in Pepe is that it permits the network to be dynamically reconfigured.

With the network simulator, we can evaluate the performance of the interconnection network for parallel applications and synthetic workloads. The simulator allows us to vary the parameters of the network and study how to improve its behaviour in real cases and predetermined situations. The simulator can model at the flit level different topologies and network sizes.

5.1. Message generation

We have considered hot-spot [11] and trace traffic models. In the first case, the situation that is going to be simulated is the following: Let us consider a network with a uniform distribution of message destinations. In this message pattern, message destinations are randomly chosen among all the nodes with the same probability. At a given moment, and with the network in steady state, the communication pattern changes, and a small number of hot-spots appear in the network. This situation repeats with a variable frequency and, in general, the hot-spots are different.

As it is easy to imagine, enormous congestion is produced in the network, due to the great number of messages that want to reach the hot-spot node. This causes the appearance of contentions in the network and the consequent delays, therefore degrading the performance of the network.

On the other hand, the traces have been obtained by means of the simulated execution of a parallel algorithm. This algorithm is used for triangularizating a sparse matrix and it is based on the fast Givens rotations. The parallel implementation of the algorithm requires as many processes as columns the sparse matrix has. If we define the type of a row as the column position occupied by its leftmost non-zero element, then it is well known that only rows of the same type can be rotated together. Then we distribute the rows among processes in such a way that each process stores all the rows of the same type. After a pair of rows has been rotated, one of them increases its type, being sent to the corresponding process to be rotated again.

5.2. Parameters of simulations

We have evaluated the performance of the reconfiguration algorithm on 2D torus with 16, 64, and 256 nodes. The deterministic algorithm proposed in [4] 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 of each node, we have used the delay model proposed in [2]. The router takes $4.7\ ns$ to compute the output channel; the switch takes one $4.8\ ns$ to transfer a flit through the crossbar and the time required to transfer a flit across a physical channel is $6.8\ ns$. On the other hand, we have considered that the useful throughput of the bus is 16 Mbps and token moves quickly ($100\ ns$ per node). Finally, each switch reconfiguration spends $1\ \mu s$.

The testing matrices have been generated at random by making a homogeneous distribution of their non-zero elements. Large size matrices have been selected to produce a large message traffic to better appreciate in this way the advantages of the reconfiguration algorithm. For a specific number of columns in the matrix, tests with different rectangularity factors have been carried out for a minimum value of two, since smaller factors can hardly produce traffic in intermediate nodes of the network. An average number of two non-zero elements per row has been taken in order to ensure we are dealing with sparse matrices. Results included in this work refer to matrices shown in the table 1.

The proportion of messages at the hot-spot has been varied between 5% and 20%. For each simulation run, we have considered that message generation rate is constant and the same for all the nodes. Each simulation was run until the network reached 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 taken has been 1 and 2, and they have been obtained randomly. Finally, 128-flit messages have been considered.

	Matrix		Trace	
	Rows	Columns	Messages	Avg. Length
A3	2400	600	300068	946
B3	3200	800	527165	1254
C3	3600	900	646648	1388
D3	4000	1000	823832	1583
E3	4800	1200	1170154	1882

Table 1. Matrices and corresponding traces

5.3. Simulations results

In this section, we present the results obtained from the evaluation of the reconfigurable network model. We have born in mind 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 is 13.2 ns. According to section 4, input buffer size is 21 flits. We have considered 4 flits as the output buffer size. For other values, different from the above parameters, the result tendency is maintained.

To obtain the graphics below shown several parameters have been varied. In the case of trace workload: Load size (figures 3 and 4) and the distribution way of processes among processors (figure 5); and for the other type of load, the number of hot-spots and the traffic (figure 6).

As it has been shown in section 3, we have used an indirect network to design the reconfigurable system architecture. In figure 3, the reconfiguration effect on the proposed system is shown (Non-pipelined channel network with reconfiguration, NPWR, versus Non-pipelined channel network non-reconfiguration, NPNR). It can be appreciated how the simulation time is reduce for any load size. This reduction has achieved, in some cases, a value of until 15 %. The changes on the network have made possible to situate in much more adequate positions than the ones they had before, the nodes of the system. As a consequence of this fact the messages spend much less time in reaching their destinations. We would like to emphasize this improvement is produced with a small number of changes in the network.

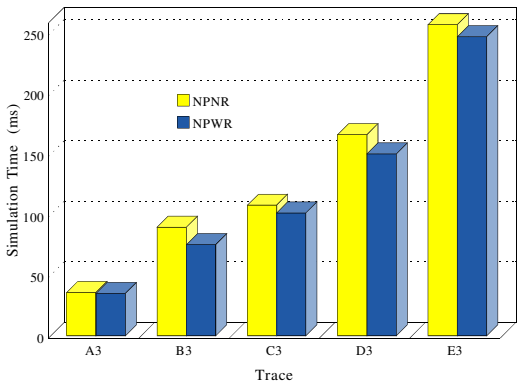

Figure 3. Reconfiguration effect according to load size for 2D torus with 64 nodes

Even though these results are significative, it is important to know the behaviour of the system by making use of a direct non-reconfigurable network (DN) and to compare results. Rest of figures show some of these results and also those which were obtained applying channel pipelined technique and reconfiguration (CPWR) on the proposed system.

Two clear facts can be observed in all the graphics. On the one hand, an indirect reconfigurable network does not achieve the performance offered by a direct one. And on the other hand, the application to the proposed system of the channel pipelined technique improves in a significative way the performance of direct networks. These improvements are kept even when realizing a good distribution of processes among processors as can be seen in figure 5.

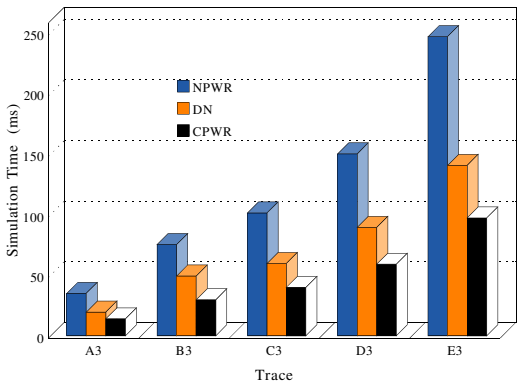

Figure 4. Channel pipelined effect according to load size for 2D torus with 64 nodes

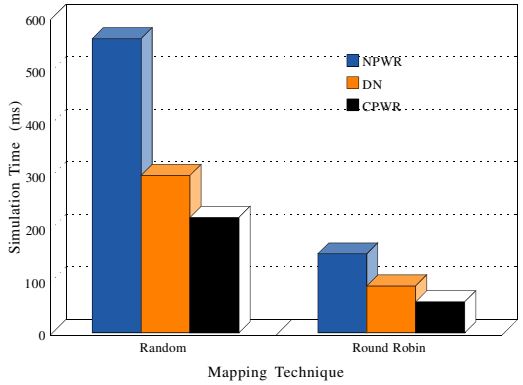

Figure 5. Channel pipelined effect according to mapping for 2D torus 8×8 and trace D3

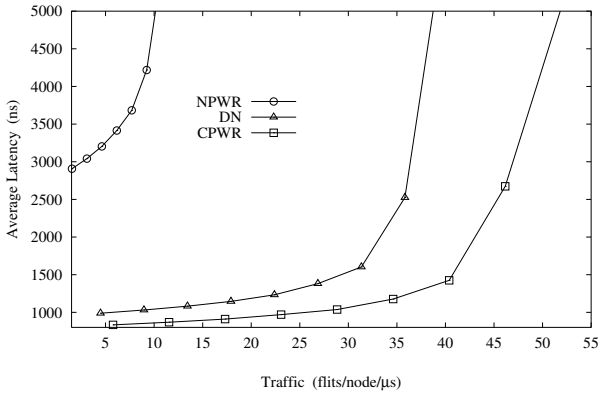

Figure 6. Channel pipelined effect according to traffic for 2D torus with 64 nodes, one hot-spot and 20 % non-uniform component

6. Conclusions and future work

In this paper, we have presented a reconfigurable network model. We have featured the reconfiguration technique supported by this model. We have analyzed the capabilities of several types of configuration, using a unique crossbar or by means of a multistage network. The performance of these models has been analyzed by simulation. We have presented here the obtained results for an average size network (64 nodes) under trace and hot-spot workload.

As it can be observed in the presented figures, the reconfigurable network obtains good results, without needing a high number of changes to reach these. The results obtained are much better when the technique of channel pipelining has been applied. This technique permits to reach throughput close to direct networks using indirect networks. If these networks can be reconfigurated, the results could be improved in some situations.

For future work, we would like to extend our study over larger sized networks, such as 512 or 1024 nodes. To achieve this, we would use a multistage instead of a crossbar. Besides, we would like to study the behaviour of the employed technique over other topologies such as mesh and 3D torus, and under other models of workload. We are interested in getting to know the real capabilities of the reconfigurable interconnection networks in any kind of context.

References

[1] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet - a gigabit per second local area network. *IEEE Micro*, pages 29–36, Feb. 1995.

[2] A. Chien. A cost and speed model for k-ary n-cube wormhole routers. In *Proc. of Hot Interconnects'93*, Aug. 1993.

[3] W. Dally. *Network and processor architecture for message-driven computers*, chapter 3, pages 140–222. VLSI and Parallel Computation. Morgan Kaufmann Publishers, 1990. R. Suaya and G. Birtwistle.

[4] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, C-36(5):547–553, May 1987.

[5] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, Dec. 1993.

[6] C. Fraboul, J. Rousselot, and P. Siron. Software tools for developing programs on a reconfigurable parallel architecture. In D. Grassilloud and J. Grossetie, editors, *Computing with Parallel Architectures: T. Node*, pages 101–110. Kluwer Academic Publishers, 1991.

[7] J. García and J. Duato. Dynamic reconfiguration of multicomputer networks: Limitations and tradeoffs. In P. Milligan and A. Nuñez, editors, *Euromicro Workshop on Parallel and Distributed Proces.*, pages 317–323. IEEE Computer Society Press, 1993.

[8] J. García, J. Sánchez, and P. González. Pepe: A trace-driven simulator to evaluate reconfigurable multicomputer architectures. In *Lecture Notes in Computer Science*, volume 1184, pages 302–311. Springer Verlag, 1996.

[9] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[10] D. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, C-24(12):1145–1155, Dec. 1975.

[11] G. Pfister and A. Norton. Hot spot contention and combining in multistage interconnect networks. *IEEE Trans. Comput.*, C-34:943–948, Oct. 1985.

[12] S. Scott and J. Goodman. The impact of pipelined channels on k-ary n-cube networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):2–16, Jan. 1994.

[13] A. Tanenbaum. *Computer Networks, (2nd ed.)*. Prentice-Hall, 1988.