Process Communication in NÓ//

Antônio Augusto Frölich, Cesar Albenes Zeferino & Valeria Alves da Silva

Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Ciência da Computação
88.049.970 - Florianópolis - SC - BRASIL
Tel.: +55 48 231-9543 Fax: +55 48 231-9770
E-mail: {guto,zeferino,valeria}@inf.ufsc.br

ABSTRACT

This paper presents a communication strategy for the NÓ// project of a parallel programming environment supported by a multicomputer with dynamic interconnection network. This project is being developed by Federal University of Santa Catarina in cooperation with Federal University of Rio Grande do Sul. The project gathers research groups in areas such as operating systems, computer architectures, programming languages and simulation.

The proposed communication strategy is organized in three levels: physical, link and mailbox. The three levels interact to supply processes with a flexible and location independent mechanism for communicating.

Key-Words: High Performance Computing, Multicomputers, Operating Systems, Parallel Architectures.

1. INTRODUCTION

NÓ// project aims to develop a comprehensive parallel programming environment, including the conception of a multicomputer that will serve as basis for the whole project. NÓ//'s basic motivation is the possibility to adopt such a kind of architecture as a natural environment to implement parallel programs by means of communicating sequential processes networks [2].

At present, the project gathers research groups in areas like computer architectures, operating systems, programming languages and simulation from two federal universities: Santa Catarina (UFSC) and Rio Grande do Sul (UFRGS).

This paper focuses on the process communication strategy for NÓ//, which is based on the mailbox concept. The proposed scheme is implemented in three levels: physical, link and mailbox. Together, these three levels provide a flexible and location transparent mechanism for process communication in the multicomputer.

After this introduction, the text continues with a brief description of NÓ//'s architecture, including the prototype implementation. After, the communication strategy is depicted through its three levels. At last, authors' personal conclusions are presented.

2. THE ARCHITECTURE

The multicomputer architecture proposed for the NÓ// programming environment is constituted by a set of processing nodes interconnected by a crossbar network, which can be dynamically configured to satisfy the load imposed by the running programs. Besides the work nodes, the architecture presents a specialized control node that commands the crossbar to establish and to release connections among any two work nodes. Work nodes are the ones that actually execute the parallel program [7].

An extra control mechanism helps on the interconnection network management, providing the control node a manner to receive work nodes' requests and to acknowledge the required services completion. Each of multicomputer node is comprised of a processor with private memory and a communication adapter that is responsible for the interface to the crossbar and to the control mechanism.

The Prototype

In order to validate the project, a low-cost prototype has been designed (figure 1). This first prototype has eight work nodes, each one comprised of an i486 microprocessor, 8 Mbytes of memory and a communication adapter card specially developed for NÓ// [7].

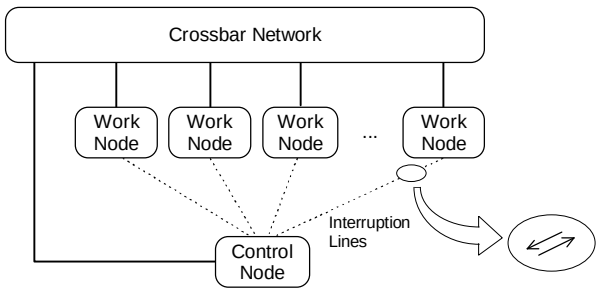

Figure 1: NÓ// prototype architecture.

In the prototype, the interconnection network is based on Transputer-Links [5][6] and is implemented by an IMS C004 crossbar and by IMS C011 link adapters in the communication cards. Besides the Transputer-Link adapter, each communication card has a two-way interruption line that connects the work node to the control node. One line allow the work node to interrupt the control node to request a service, while the other one is used by the control node to confirm the service. By using these interruption lines in combination with temporary connections in the crossbar, the link level can define an efficient control protocol to manage connections.

3. COMMUNICATION: THE PHYSICAL LEVEL

In the physical level, the communication between any two nodes is serial, asynchronous, full-duplex and byte confirmed, as defined by the Transputer-Link protocol [5]. This protocol is implemented at the communication cards by C011 link adapters at 10 Mbps. Each data byte is transmitted as a package that is confirmed by the receiver with an acknowledge package (figure 2).

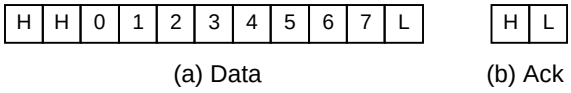

Figure 2: Data and acknowledge packages.

This acknowledgment mechanism is also used for flow control. A new data package is only sent after the previous package acknowledgment.

In a full-duplex, 10 Mbps transmission flow of 11 bits packages, the physical level supports a 7.2 Mbps bandwidth to the link level.

4. COMMUNICATION: THE LINK LEVEL

The link level makes use of physical level mechanisms to provide an efficient connection-oriented service to the mailbox level. This level defines a control protocol that allows any two nodes in the multicomputer to establish a data connection. The link level, as well as the mailbox level, is implemented by a micro-kernel that executes in all of multicomputer nodes, even in the control node.

The Control Protocol

In order to establish a connection between two specific nodes, the link level defines the follow protocol:

1 - When a work node wants to send or receive a message, it first interrupts the control node through the proper line;

2 - The control node recognizes the interruption, identifies the source and establishes a temporary connection through the crossbar to the source node;

3 - The control node then acknowledge the interruption and waits for a service request;

4 - The work node issue a service request, e. g., “want to send a message to mailbox X” or “want to receive a message from mailbox X” and then blocks;

5 - The control node releases the connection and passes the service request to the mailbox level;

6 - When the mailbox level detects to processes trying to exchange a message, i. e., a send and a receive to the same mailbox, it requests the link level to establish a direct connection between the two host nodes;

7 - The control node configures the crossbar and signalizes the proper interruption likes.

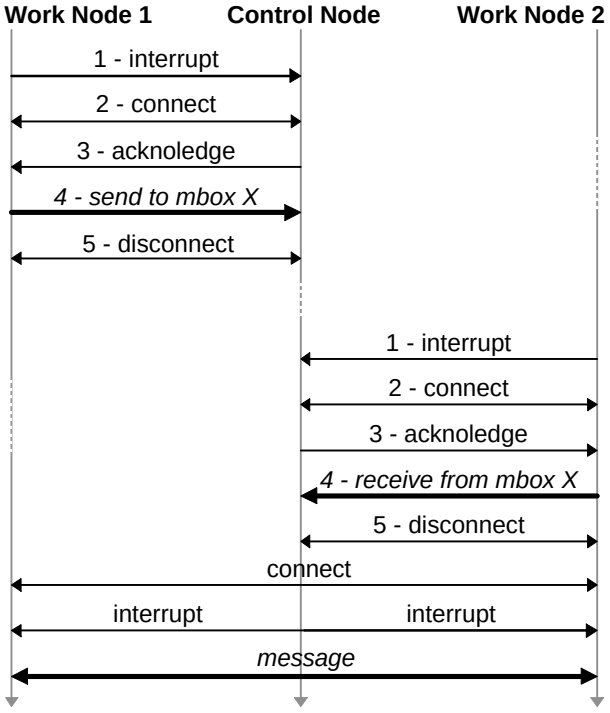

Figure 3: Link level control protocol.

5. COMMUNICATION: THE MAILBOX LEVEL

Communication mechanisms from physical and link level are made available to applications through primitives implemented by the micro-kernel. Such primitives exploit the connection oriented protocol from link level to provide a datagram service to applications.

The decision of not supporting connection oriented communication to application comes from observations on the client/server model [4]. In such model, most communication occurs in the simply way of requests and

replies. Once a server can receive service requests from different clients with variable frequency, it is easy to conclude that keeping control information on connections would be a big overhead.

A reliable, location transparent and high performance datagram service is thus provided to applications through the mailbox concept. Every time a process wish to communicate, it can create one or more mailboxes to receive and send messages. In this fashion, the object identified in communication is the mailbox, not its associated processes.

Mailboxes

The words mailbox and port are some times used in the same sense, what is certainly a mistake. A mailbox is a two-way communication channel that allows simultaneous receives, even from different processes, while just a single process can receive messages from a port [1]. According to this concept, NÓ// supports a communication mechanism based on mailboxes.

Each mailbox is uniquely and globally identified to the whole multicomputer [3]. This is easy to achieve because the existence of a central control node that maintains information about all active mailboxes. As the return to a mailbox creation request, a process receives a capability, which identifies and protects the mailbox. By holding a mailbox's capability, a process can send and receive messages or pass it to other processes. Micro-kernel does not control which or how many processes hold valid mailboxes capabilities.

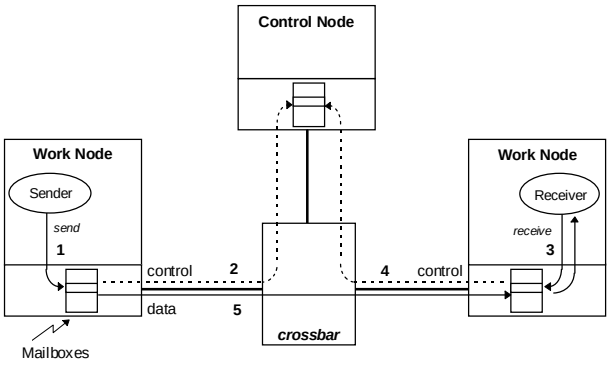

Figure 4: Mailbox communication in NÓ//.

Figure 4 depicts message exchange between two processes using mailboxes. To send a message, a process issue a "send" identifying source and target mailboxes (1). When the micro-kernel notice the send, it establishes a connection to the control node and identify both mailboxes: source and target (2). The sender process then blocks until another process request a receive from the common mailbox. In this way, process communication is synchronous and does not imply in buffer management at the control node.

Similarly, a process wishing to receive a message, requests the micro-kernel (3), which then informs the control node (4). If the specified mailbox is empty, i. e., there are no blocked processes trying to send a message to it, then the process trying to receive blocks. When the control node detects a process trying to send and another trying to receive from the same mailbox, it establishes a data connection between the nodes hosting the processes, so the message can be directly transmitted, without passing through the control node (5). It is also possible to set a time-out for the send and receive operations, which, when reached, generates an appropriate error.

Using this mailbox scheme, a process does not need to know the identification, nether the location, of other processes to send them messages. Actually, the receiver process, or its location, can dynamically change during a message exchange.

Automatic Server Replication

The micro-kernel executing at the control node presents some peculiarities if compared to the ones executing in work nodes. Once all mailbox creation requests, as well as sends and receives, are centralized at the control node, it can maintain valuable statistical information, that can be used by external process in order to optimize the multicomputer load.

Based on this information, it is possible to wander several optimizing servers, one of them, already developed, yields to automatic idempotent server replication. An idempotent server returns always the same result to a certain request, independently of how many times it is issued. That is, an idempotent server does not base its actions on local control information. All the information necessary to execute a service is inserted into the request. Such a kind of server can be replicated without modifying its implementation. An idempotent distributed file server, named PYXIS [4], has been developed and has validated this ideas.

Knowing each server nature (idempotent or not), a special server can constantly monitors the communication system load. It is easy to conclude that a server attending to a mailbox with a long list of pending messages for a long time is an overloaded server and that replicating it can only be a good action.

Based on this facts, a server has been conceived to automatically replicate other idempotent servers every time its associated mailboxes present a message queue grater than a specified limit. In the same way, every time a replicated server presents an empty mailbox for a certain time, it is automatically destroyed.

CONCLUSION

This paper has presented a communication strategy for the dynamic interconnection network multicomputer of NÓ// project. The strategy is conceived by three levels: physical, link and mailbox. The physical level supports Trasputer-Links among any two nodes in the multicomputer; the link level provides a point-to-point connection-oriented service; and the mailbox level supports an abstraction by which processes can exchange datagrams, regardless other processes location. Besides, the mailbox level uses the flexible mechanism from the link level to avoid unnecessary data transfers, as well as to abolish buffer management from the system.

The link and mailbox levels have been implemented over a physical level simulation and are now being debugged. The multicomputer prototype that implements the described physical level is already design, simulated and validated. Its implementation, nevertheless, is restrained by monetary resources. We believe a operational version will be available latter this ear.

REFERENCES

[1] ANDREW, G., Concurrent Programming: Principles and Practice, Redwood City, Benjamin/Cummings, 1991.

[2] CORSO, T., *Ambiente para Programação Paralela em* Multicomputador, Florianópolis, UFSC/INE, 1993 (thechincal report).

[3] FRÖLICH, A. A. & CORSO, T., *SDFS: Simple Distributed File System*. In: Proceeding of the 20th Latinamerican Conference on Computer Science, Mexico, Mexico City, September 1994.

[4] FRÖLICH, A., *PYXIS: A Distributed File System*, Brazil, Florianópolis: UFSC/CPGCC, September 1994 (M.Sc. Thesis).

[5] INMOS, *IMS C011 Link Adapter*, Inmos Limited, 1988.

[6] INMOS, *IMS C004 Programmable Link Switch*, Inmos Limited, 1989.

[7] ZEFERINO, C. A., LÜCKE, H. A., & SILVA, V., *Um Multicomputador com Sistema Experimental de Comunicação*. In: 7th Brazilian Symposium on Computer Architecture and High Performance Processing, Brazil, Canela, July 1995.