AN ADVANCED ENVIRONMENT FOR PROGRAMMING TRANSPUTER NETWORKS WITH DYNAMIC RECONFIGURATION

García, J. M.
Dpto. de Informática
Universidad de Castilla-La Mancha
02071 Albacete (Spain)

Duato, J.
Facultad de Informática
Universidad Politécnica
46071 Valencia (Spain)

SUMMARY

In this paper we present a programming environment for multicomputers. Among other features, it allows us to evaluate the performance of parallel algorithms running on a multicomputer with both, static and dynamically reconfigurable topologies. The results of this evaluation are obtained by simulating a machine model based on a transputer network.

Our environment -called FDP- permits the simulation of the behaviour of a multicomputer. Several machine parameters can be adjusted. For example, we can vary the network topology, the number of nodes, the routing algorithm in the network, etc. Choosing different options is easy, because FDP has a friendly user interface.

In our environment, a parallel algorithm is programmed in the Distributed Pascal language. This new parallel language, which we have developed, is based on standard Pascal. Some extensions allow an easy and elegant programming of parallel algorithms, consisting of processes which communicate by means of message-passing.

1. INTRODUCTION.

The increasing demand of greater processing power capacity in computers has led to the development of several parallel architectures, usually of the SIMD or MIMD type according to the classification proposed by Flynn [1]. MIMD computers can be classified as machines with shared memory (multiprocessors) or distributed memory (multicomputers). In recent years more attention has been given to message-passing multicomputers because they offer a good cost/performance ratio and they are easily expandable up to a large number (hundreds or thousands) of nodes [2].

The basic idea behind parallel programming is straightforward: to split a computation among several processes cooperating in the solution of the global problem. The first difficulties arise when this idea is put into practice, as then one discovers that the sharing and cooperation among processes is a complex task and that this task is dependent upon the type of architecture the machine has. In multicomputers, cooperation and synchronization among processes is

usually performed by exchanging messages.

As for Von Newmann architectures, a set of software tools are needed for the development and execution of programs in these parallel architectures. The development of languages, compilers, operative systems, debuggers, performance monitors, etc., is at present one of the main research areas in order to obtain the required performance from the new architectures.

Usually, this research focuses on two ways of programming these systems:

a) Automatic parallelization: In this technique, the compiler assumes the responsibility of detecting which loops (or generally speaking, a series of sentences in the language) can be executed in parallel.

b) Definition of new languages. We face two different possibilities. We can add extensions to sequential languages, allowing the communication between processes. Alternatively, a new language can be defined; an example of this is the Occam language [3].

In this paper we present an environment for multicomputer programming. This environment allows us to edit a program written in Distributed Pascal [4], compile, debug and simulate its execution on a multicomputer, thus permitting the use of different parameters.

Moreover, our environment also provides the opportunity to study the main problem we face at present when dealing with multicomputers: the interconnection network.

For many applications, the interconnection network is the main bottleneck for this architecture. There are several ways to alleviate this problem. One of the most important improvements consists of pipelining message transfers. The best known technique of this kind is wormhole routing [5]. Another important improvement consists of allowing the network topology to be reconfigured dynamically, in such a way that the nodes which communicate more frequently can be directly connected. Although this technique can be applied to both, store-and-forward and wormhole networks, the only commercially available machines with that feature are transputer-based machines, like the Parsys SN 1000. Up to now, these machines only support store-and-forward routing. A multicomputer with a dynamically reconfigurable network can change its own topology arbitrarily at runtime [6,7]. With this feature, we can attempt matching the network topology with the communication requirements of the application program, increasing the performance of these machines.

A remarkable feature of our environment is the possibility to evaluate the improvement obtained in the multicomputer performance when using a dynamically reconfigurable interconnection network. This improvement is more noticeable for those applications with a time varying communication pattern.

The rest of this paper is structured in the following way. Section 2 is dedicated to the presentation of our environment, focusing on its

basic features. The advanced features of our environment are presented in section 3. Section 4 is devoted to the presentation of the related work in this field. Finally, in section 5, the main conclusions are presented as well as alternative ways for future work.

2. FDP: AN ENVIRONMENT FOR PARALLEL PROGRAMMING.

FPD (Friendly Distributed Pascal) is a friendly environment for program development with the Distributed Pascal language [4].

This programming environment has been developed for an IBM PC or compatible, although it can be easily ported to other computers, e.g. workstations. It allows us to edit a Distributed Pascal program, compile it and detect possible errors as well as executing it on a simulated multicomputer. The environment has a graphical interface.

Next, we are going to present two fundamental aspects of FDP: the machine model in which this environment is based and the basic features it has.

2.1. Computer model.

FPD is an advanced environment developed for multicomputers. Therefore, the only way to communicate and synchronize the different processes is by means of message-passing.

In order to make the environment independent of the computer, the number of processes in a program is not limited to the number of processors. With regard to the programming of an algorithm the concept of virtual processor is used, i.e., we assume that there are as many processors as there are declared processes. It is a function of the run-time kernel to determine which processes are loaded in each processor (mapping). This mapping is static, i.e. once it has been decided in which processor each process is executed there is no change throughout the execution of the program.

The communication model is based on message-passing. For the programming the concept of virtual network is employed, supplying a totally connected topology to the user. This concept permits a direct communication between processes without taking into account the interconnection network found in a particular multicomputer. The user must take this into account, since the correct execution of a program must not depend upon the topology of the interconnection network. Again, it is a function of the kernel to determine if the source and destination processes of a message are in the same processor or not. In the first case, the communication is internal. Otherwise, the router is executed in each processor through which a message passes. The physical channels of a processor are multiplexed to provide service for all the processes which are being executed on it. This can be done because the mapping is static and known throughout the entire system.

The programming style we have adopted is the SPMD one [8], i.e., all the processes execute the same code. In the case several processes

share a processor, there is only one code copy in memory, each process having its own variables. This approach reduces memory occupation significantly. However, it does not imply that all the processes execute the same instructions. In order to be able to execute some code blocks only by some processes, the Distributed Pascal language defines the variable *process*, which identifies each process. So, for the process 0 the variable *process* will be equal to 0, for the process 1 it will be 1, etc...

2.2. Basic features.

FDP includes a set of basic features shared by most programming environments. For a detailed description, the interested reader can see [9]. Figure 1 shows the aspect of the FDP environment during the edition of a program with several displayed windows.

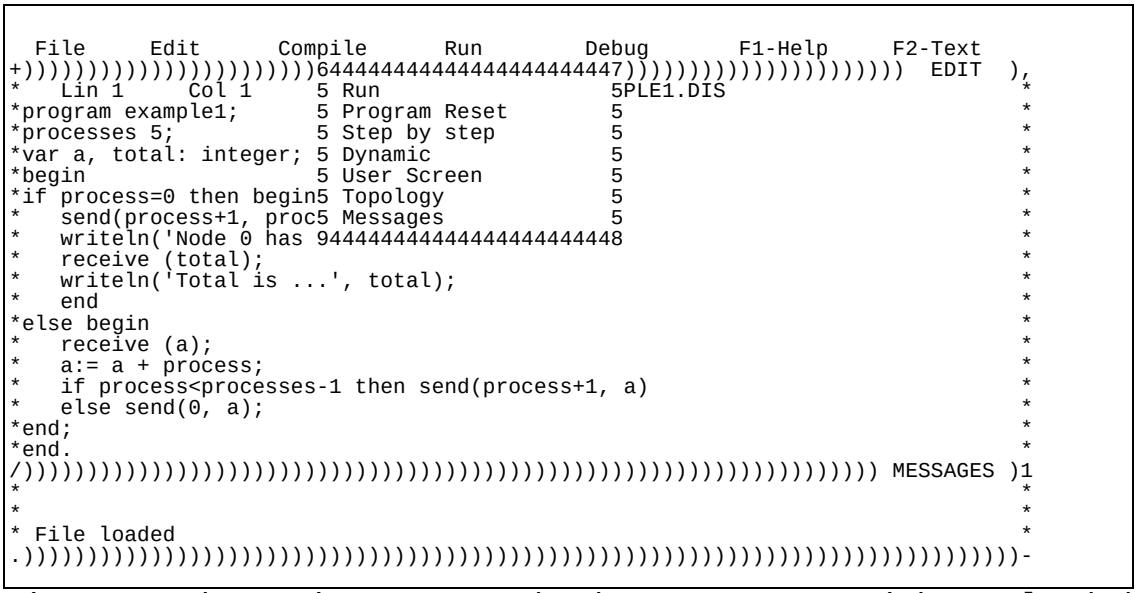

Fig. 1. The main menu and the run menu with a loaded program.

The main menu consists of the following items:

a) FILE: Several functions for file handling, such as creating a file, loading or saving a file, etc.
b) EDIT: To create new files or modify the ones that already exist.
c) COMPILE: It compiles a file written in Distributed Pascal to an intermediate language. In case of error, it informs with a suitable message about its type and location.
d) RUN: It runs a previously compiled program. This option allows the configuration of the multicomputer architecture.
e) DEBUG: This menu option gives access to a symbolic debugger with three functions and allows a more efficient error detection and correction.

The environment offers a context-sensitive help to the user, displaying the adequate information. It also presents an information window, displaying messages when functions are executed: file saving, program compilation, etc.

During the execution, the intermediate code generated by the compiler is interpreted and the behaviour of the real computer is simulated with the supplied parameters. One of the advantages of this approach is the possibility to detect execution errors due to parallel processing. For instance, in the case of deadlock, information is given about where each process was blocked (at the source code level). In this way we can comfortably learn how to move form sequential to parallel programming, detecting the most frequent errors produced when programming parallel systems. In fact, this environment has proved to be extremely helpful while teaching parallel programming.

3. ADVANCED FEATURES OF FDP.

FDP includes two advanced features: symbolic debugging of a parallel program and selection of a specific architecture to simulate the execution of a program.

3.1. Symbolic debugging of a parallel program.

Most environments developed to simulate multicomputers include debugging functions. Through these functions the programmer intends to become familiar with the complex parallel world, while simultaneously permitting a more efficient error detection and correction.

FDP includes a symbolic debugger with three functions:

1) Step-by-step execution: FDP allows a step-by-step execution of a program by concurrently executing a statement from each active process; generally, this statement is different for each process, since the execution is asynchronous. With this option, a procedure or function is also executed step-by-step. In the bottom right margin of the environment the process being executed step-by-step at that moment is shown.

2) Quick step-by-step execution: During the step-by-step execution of a program, sometimes it is desired that procedures and functions are executed in a single step. To achieve this, the debugger offers this alternative function which allows a more effective and rapid step-by-step execution of a program.

3) Variable value display: The third function of the debugger allows to observe the contents of the variables declared by each process, whether they are local or global variables for a procedure (whenever such procedure is activated).

3.2. Multicomputer architecture.

As it has already been mentioned, programming is simplified by implementing a generic multicomputer with an ideal architecture. This architecture supports the concepts of virtual processor and completely connected topology. Regarding the execution of the program, FDP allows the specification of a group of features related to the multicomputer architecture. The mentioned features are the following:

a) Network topology. FDP admits the specification of three different topologies for the interconnection network: 2-D mesh, hypercube and the ring. The routing algorithm is static and uses a store-and-forward flow control mechanism. Network traffic can be reduced by dynamically reconfiguring the network topology. Reconfiguration is carried out in such a way that the topology is not altered. Simply, some nodes interchange their positions in the network [10].

b) Number of nodes. The concept of virtual processor allows the programmer to define any number of processes without taking into account the number of physical processors available in the multicomputer. This number of physical processors can be changed using one of the options of the environment. Most of the simulations have been carried out with a number of nodes equal to 16, because this is the number of nodes of the Supernode SN 1000 available in our laboratory.

c) Mapping of processes to processors. The mapping of processes to processors is performed in a simple way, leaving aside requirements such as load balancing for each processor or a low communication cost. Using this mapping, a satisfactory load balance is obtained for most numeric algorithms, for which this environment has been designed. The reduction in the communication cost is expected to be attained from the dynamic reconfiguration of the network. A process *i* is assigned to a node *j* according to the following function:

$$j = i \ \mathbf{mod} \ node\_count$$

where *node_count* indicates the total amount of physical nodes for that particular simulation. By applying this function, a cyclic distribution of processes to processors is obtained.

The number of links each processor has to communicate with other processors is restricted to four, using the transputer [11] as a model. It implies that the maximum number of nodes for hypercube topology is also four.

d) Dynamic reconfiguration. The most remarkable feature of FDP is the simulation of a multicomputer with a dynamically reconfigurable interconnection network. Some of the parameters for the reconfiguration can be changed, for example, if the dynamic reconfiguration is active or not, the minimum thresholds to reconfigure and the kind of algorithm used in the reconfiguration. An explanation of the main features of these

algorithms and some simulations results can be found in [10,12]. These algorithms are transparent to the programmer.

e) Simulation results. The main menu offers the possibility to create a file where the results of the simulation are collected. Two levels of detail are available. The information given in the file covers the following aspects:

- Changes performed: In the case the option of dynamic reconfiguration of the network is selected, it informs about the number of reconfigurations performed.
- Message traffic: Whenever a message has to cross an intermediate node to arrive at the destination node, the message traffic in such intermediate node is supplied. The sum of the traffic in every node is the total message traffic in the interconnection network. This is a very important parameter because it is used to measure the quality of the algorithms for dynamic reconfiguration; one of the main goals of the dynamic reconfiguration of the network is to decrease the message traffic through the interconnection network. From this point of view, for the same number of changes in the topology, the best reconfiguration algorithm will achieve the lowest message traffic.
- Reconfiguration threshold: Two thresholds are necessary to decide when the reconfiguration is carried out [10]. Therefore, some information is given about the value established by the user for such thresholds.
- Processor traffic: Another important objective of the reconfiguration is to reduce the traffic through intermediate nodes. In order to do so, it informs about the five processors with the lowest traffic and the five processors with the highest traffic (specifying the value for each case).
- Total number of messages: It also gives information about the value of the total number of messages which have been sent through the network. This is useful to obtain a percentage of the amount of messages which have caused message traffic in the intermediate nodes.
- Local information for each node: When the higher level of detail is selected, it also adds the number of messages each node has sent, the amount it has received and the amount of messages which have circulated through it (local message traffic in that node).

Figure 2 shows a report for the lower detail level.

```
Threshold_1: 55     Threshold_2: 35
Message traffic: 1787
Changes: 34
Processors without traffic:
Processors with higher traffic: 15 (283), 3 (195), 11 (107),
Messages sent: 833
```

Fig. 2. The report for the lower detail level.

4. RELATED WORK.

Some projects have also been developed in order to model multicomputers, mainly aimed at developing programming environments and tools for programmers, rather than tools designed to work with the architecture of the system. This is the case of PIE [13], a programming and instrumentation environment for parallel processing. Although it has originally been designed for shared-memory systems, it is expected to be translatable to other systems. PIE offers a graphic representation of the processes and their interrelations.

Another environment is Poker [14], which has been designed to emulate a specific architecture: the Configurable Highly Parallel Computer. Poker presents an integrated environment and allows the user to control several levels of the architecture of the system. Although Poker offers an excellent modelling of the multicomputer, it is difficult to add new architectures; it has been recently extended to support the Cosmic Cube.

Two other recent developments are PARET and Hypertool. PARET [15] attempts to simulate the behaviour of a specific program in a specific architecture rather than emulating in detail the execution of the program itself. The objective of PARET is the design and study of multicomputers as systems, rather than considering them as separate components. Algorithms and architectures have been graphically represented, and their state is controlled interactively during the simulation.

Even newer than PARET is the Hypertool [16] environment. Hypertool has been designed to increase the programmer's productivity. The development of a parallel program using Hypertool is as follows: the programmer designs an algorithm, he carries out a partition in processes writing a sequential program as a set of procedures (the different processes). Hypertool automatically converts the program to a parallel program for a machine with message-passing, assigns each process to a processor, simulating its execution and properly informing about the results obtained regarding power, execution time, communication time, inactivity time for each processor and delay

introduced in the network in each communication channel. This report also presents the dependency of data among processors, as well as the extent of parallelism and load distribution for each processor. The transmission time of a message is evaluated by means of a classic linear model as a function of the length of the message and the startup time; the model assumes that the traffic within the network is not too heavy, and therefore it ignores the delay caused by message contention in a node.

Finally, we are going to describe Tenor++ [17]. This environment is a high level interface which allows the development of applications with variable topologies on the Supernode reconfigurable multiprocessor. The functionality of the interface is available by means of two algorithmic languages -the SDR and the SDT- which allow reconfiguring strategies to be specified and topologies to be symbolically described. Tenor++ aims at taking advantage of the dynamic reconfiguration facilities of the Supernode. These facilities are available via two programs: a first program for defining the application as a set of phases separated by reconfiguration points, and a second program where the topology is defined as a graph construction.

This environment, the most similar to FDP, is distinguished from FDP by two fundamental aspects. In the first place, in FDP an application program is not divided into phases for the reconfiguration of the network. We have developed an algorithm which controls the reconfiguration of the network. Depending on the message traffic, the algorithm modifies the network topology in order to decrease such traffic, therefore increasing the multicomputer performance. In Tenor++, the main problem is that we must know which topology adjusts better to each phase before the execution of the application program.

Secondly, at the end of the execution, FDP presents the results concerning message traffic and saturation in intermediate nodes. This report allows us to compare different topologies both, with and without dynamic reconfiguration.

5 CONCLUSIONS AND FUTURE WORK.

In this paper, we have presented a new programming environment, the FDP, suitable for the programming of distributed memory MIMD computers, such as transputer networks. The programming language for this environment is the Distributed Pascal, which allows us a simple, elegant, flexible and powerful programming of these computers.

The most important feature of FDP is that it allows the study of the interconnection network. This study can be considered from two points of view: comparison of several static topologies and study of the dynamic reconfiguration of the network.

Up to now, FDP is the only programming environment which allows the study and evaluation of the dynamic reconfiguration of the interconnection network in multicomputers. An algorithm has been

proposed which allows to reconfigure the network [12].

FDP has made possible the extensive evaluation of the dynamic reconfiguration of the network, both for simple problems and for matrix computations [10,12]. We have obtained excellent results, both with a null cost for each reconfiguration and considering the cost for the reconfiguration.

We are planning to generate code for multicomputers based on transputer, given the spread of these computers in the EEC. We also plan to develop various algorithms to handle the run-time problems: mapping, load balancing, etc. We are specially interested in those solutions that take advantage of a unique hardware feature found in these computers: the dynamic reconfiguration of the interconnection network.

REFERENCES

1. FLYNN, M.J.- Some Computer Organizations and their Efectiveness,
   IEEE Trans. on Computers, 21, 948-960, (1972).
2. ATHAS, W. and SEITZ, C.- Multicomputers: Message-passing Concurrent Computers,
   IEEE Computer, 21, Aug. 9-24, (1988).
3. INMOS LIMITED.- *"Occam-2 Reference Manual"*. Prentice-Hall, England, 1988
4. GARCIA, J.M.- A New Language for Multicomputer Programming, *Sigplan Noticies*, To appear.
5. DALLY, W.J. and SEITZ, C.L.- Deadlock-free Message Routing in Multiprocessor Interconnection Networks,
   IEEE Trans. on Computers, 36, 547-553, (1987).
6. BAUCH, A.; BRAAM, R. and MAEHLE, E.- DAMP: A Dynamic Reconfigurate Multiprocessor System with a Distributed Switching Network,
   2nd European Distributed Memory Computing Conference, Munich, April 1991.
7. NICOL, D.A.- "Reconfigurate Transputer Processor Architectures", Multiprocessor Computer Architectures,
   Ed. T.J. Fountain and M.J. Shute, North-Holland, 1990.
8. KARP, A.- Programming for Parallelism,
   IEEE Computer, 20, May 43-57, (1987).
9. GARCIA, J.M.- "*Manual de Usuario del FDP*". Universidad de Castilla-La Mancha,
   Albacete, 1992.
10. GARCIA, J. M.- "*Desarrollo de Herramientas para una Programación Eficiente de las Redes de Transputers: Estudio de la Reconfiguración Dinámica de la Red de Interconexión*". PhD thesis. Universidad Politécnica de Valencia, December, 1991.
11. INMOS CORPORATION.- *"The Transputer Databook"*. Inmos Ltd., England 1989.

12. GARCIA, J.M. and DUATO, J.- An Algorithm for Dynamic Reconfiguration of a Multicomputer Network,
Proc. 3rd IEEE Int. Symp. on Parallel and Distributed Processing, Dallas, December, 1991.
13. SEGALL, Z. and RUDOLPH, L.- PIE: A Programming and Instrumentation Environment for Parallel Processing,
IEEE Software, Nov. 22-37, (1985).
14. SNYDER, L.- Parallel Programming and the POKER Programming Environment, *IEEE Computer*, 17, July 27-36, (1984).
15. NICHOLS, K.M. and EDMARK, J.T.- Modeling Multicomputer Systems with PARET, *IEEE Computer*, 21, May 39-48, (1988).
16. WU, Min-You and GAJSKI, D.D.- Hypertool: A Programming Aid for Message-passing Systems, *IEEE Trans. on Parallel and Distributed Systems*, 1, July 330-343, (1990).
17. ADAMO, J. and BONELLO, C.- Tenor++: A Dynamic Configurer for Supernode Machines, Lecture Notes in Computer Science 457, 640-651, Springer-Verlag, 1990.