Dear Prof. Succi

We acknowledge receipt of your interest in the International Conference on Parallel Computing and Transputers Applications (PACTA’92) which will be held in Barcelona (Spain) on September 21 - 25, 1992.

We are looking forward to receiving your abstract before January 10th 1992.

We will keep you informed on the organization of the Congress.

Thanking you for your interest and collaboration, we remain

Yours sincerely,

EUGENIO ONATE
Organizing Committee

Prof. Giancarlo Succi
DIST.
Università di Genova
Via Opera Pia 11A
1-16145 Genova
ITALIA

Barcelona, June 18th 1991

A SYSTEM-INDEPENDENT TOOL FOR DEVELOPING APPLICATIONS ON SIMD ARCHITECTURES

G. Marino, G. Succi, G. Levo, R. Pavone, T. Vernazza
DIST. - Università di Genova via Opera Pia 11A 1-16145 Genova

Abstract

This paper describes the features owned by an utility program designed to permit configuration of SIMD architectures in a graphical way, also providing a mechanism of runtime evaluation of the computation and communication loads over the various nodes and connections building up the network. This utility is called NAUTA and it has been implemented by now on a Transputer architecture, namely an In-Sun Computing Surface produced by Meiko ltd.

The graphical interface for configuring the architecture has been designed with the aim of keeping it really immediate and comfortable to the user, following the styles adopted nowadays by most of the human-oriented interfaces [1].

With regards to its structure, NAUTA is an XWindow application built upon the X/Intrinsic [2] and the Athena Widget Set [3].

Probably the most interesting feature of NAUTA is its portability to other kinds of SIMD architectures than those based on Transputers, especially as far as the mechanism for monitoring the runtime activity is concerned. In fact, while the interface for network configuration must in some way resemble the structure of the architecture upon which it operates, the approach adopted for load measurement is quite independent from the underlying hardware. Therefore, it appears to be a general method that can be applied to several different environments with good results.

1. INTRODUCTION

The ability of SIMD machines to achieve parallelism relies upon the concept that a complete work can often be split into several more elementary activities that can be executed in the same time. These activities can be assigned to different processors, and the relationships between the various modules in form of data interchanging are granted by a built-in communication mechanism.

In this way it could be possible, in theory, to complete the global work in a much shorter time than it would take on a normal sequential computer. Anyway, this is hindered by a lot of factors, among which there may be an unsuitable subdivision of computational loads over the various processors of the architecture, as well as a scarce fluency of the communication traffic between the processors.

When dealing with a reconfigurable SIMD machine like the Transputer, these two aspects become very important, as the user can partly operate to reduce their negative influence upon the system's performances in executing
the application program. In fact, it can be said that for every application it exists one optimal network configuration over which it can be mapped and executed i.e., the available processors and the connections between them can be arranged freely by the user in search for the structure best fit to the problem at hand.

In substance, the user of a reconfigurable MIMD architecture should take great care during the phase of constructing a suitable network environment to run his application programs.

2. TRANSPUTERS, CSTOPOLS AND PARFILES

For the purpose of configuring parallel architectures to execute user programs, several interfaces have been developed: a very popular one among them is the drafting of `.par` files, a mechanism used inside the CSTOPOL environment [4], developed by Meiko Ltd. for its Transputer-based MIMD Computing Surface [5].

Writing a parfile, according to the syntax in force, the user can assign the parallel program modules building up the whole application to the proper processors of the architecture, and define the structure of the connections between the processors themselves, thus giving form to a network of computing nodes and communication arcs.

![Figure: An example of a NAUTA-generated layout](image)

In figure 1 is shown a network fragment corresponding to a parfile like the following:

```
par
    processor 3 A & E
    processor 4 C & D
    processor 5 A & D

    network: userdefined
    close to 3 4
    close to 4 5
    close to 3 5
endpar
```

This simple example should be clear also to those who are not familiar with parfiles. It can be noted, anyway, that a human sees his way much better ahead when looking at the figure than when reading the corresponding parfile, even though such a file seldom turns out to be obscure. It is likely too, that the user who writes a parfile first draws the corresponding layout on a paper sheet and then translates it into words.

Therefore, it seems a good idea that of sparing the user this translation step, and let him configure the hardware simply by means of the bare layout sketch, automatically interpreted by the computer. This idea led to the development of a graphical interface for configuration of MIMD architectures.

3. NAUTA AS A GRAPHICAL USER INTERFACE

NAUTA is a utility program, currently implemented on a Meiko In-Sur Computing Surface, that provides a graphic window upon which the user can comfortably define a diagram layout of the desired network. Then he can specify which processes are to be executed on each of the network's processors. This information is later automatically converted into a corresponding parfile that can be submitted to the MIMD machine for execution.

It is evident that NAUTA's graphical interface, for the sake of representation of the processors and because of the way in which the same are connected with each others and loaded with the parallel processes, refers strongly to the hardware i.e., a Transputer architecture [6].

It will now be described the steps into which the architecture configuration phase can be divided, but it must be underlined first that there is not a really fixed sequence to comply to: NAUTA leaves the user the greatest freedom to arrange the work in the most natural way, and every choice can be taken back at any moment.

(a) Placing the Processes

When the user decides to add a new processor to the network currently being edited, he pops it up from a hypothetical stack and it appears on the graphic window, across which it can be moved to reach the desired position on the layout. A processor can be moved also after it has been connected to other ones. In this case the connections are preserved and the lines representing the connections are dynamically redrawn during motion.

(b) Connecting Processors among them

The connections between processors, necessary to ensure message-passing among the parallel processes, make use of specially dedicated ports on the processors. In the case at hand, dealing with Transputers, each of them provides four ports or links.

To establish a connection, the user must simply select one link on the first processor and one link on the second one: a line will be drawn automatically, and the two links will change their colours, to prove that the connection has been carried on. This will not happen in case the connection is not allowed. In fact, it is forbidden to connect a processor with itself or to connect two processors more than once.
(c) Assigning Processes to Computing Nodes
As already said, the global application program is composed of a set of simpler processes that can be assigned to different processors in order to be executed in parallel.

The user can place the processes on the various computing units displayed on the layout window by popping up for each processor a text editor window, and type in the names of the program modules to be executed there. Multiple process names on the same node must be separated by ' & ' (like in '.par' files).

Figure 2: Placing processes through a text editor

In figure 2 is shown the outlook of the text editor.
A shortcut is provided for special cases, when the same process or group of processes must be loaded on several processors. Then the user must write their names only once in an extra text editor called ' Export '. The content of this list can be imported by any processor and appended to its own local list of processes.

(d) Selecting load monitoring points on the Network
As it will be explained in the next section, NAUTA provides a system for runtime monitoring of computation and communication loads over the nodes and arcs of the network. So, during the phase of network configuration, prior to execution of the application, the user can specify, one by one or collectively, which processors and communication connections must be surveyed and the respective loads be reported.

(e) Saving and loading network configurations
Once the user has completed the phase of architecture initialization, all the information he specified can be saved into a file for later use. Number and positions of the processors, connections between them, processes loaded on each of the processors, load measurements selected: all this is recorded on an extended parfile, that the user can subsequently retrieve from inside NAUTA.

(f) Running the Application
When all the necessary and accessory information has been specified, possibly also saved on a parfile, it represents an application program. This program can now be passed to the CSTOOLS batch loader and executed. The execution can also be launched from inside NAUTA: if this does not happen, then the specifications relative to load monitoring on the network are ignored, otherwise NAUTA creates output windows and there it displays the values obtained by such measurements.

4. BENCHMARKING A MIMD MACHINE
As it was already pointed out, on a reconfigurable MIMD architecture it takes great importance the way in which the available resources (processors and communication links) are managed to accommodate a parallel application. This is the user's duty, and he must mostly rely on his common sense, since no tool exists that analyzes the problem and automatically suggests the best configuration.

It is very likely that the search for the optimal structure must proceed by a sequence of attempts when the complexity of the problem at hand is high.
A little help in this direction would be given by a system of runtime monitoring over the loads held up by the resources constituting the network. Checking which processors are busier and which links are choked during execution of an application would allow the application designer to perform more targeted modifications to the distribution of duties and to the network topology in order to achieve better performances.

5. A STRATEGY FOR MEASURING LOADS
NAUTA provides a mechanism of load measurements over the nodes and arcs of a multiprocessor network. It has been implemented for a Transputer architecture, but the underlying method is quite general and could be ported onto other MIMD machines.

The basic idea is that if a process has exclusive access to a resource, be it a processor or a communication link, it can complete its tasks in a certain minimum time. On the other hand, if the resource is condivided among several processes, all of them are delayed. Considering that the greater the number of processes requesting service from a resource, the slower each of them can bring on its duties, it is possible to evaluate the time elapsed by a process in executing a sample computation or communication task. This duration, compared with the minimum time for that sample task, gives a measure of the number of requests pending on the resource, that is after all the resource's load.

6. THE SHADOW PROCESSES
NAUTA adopts the mechanism described above to evaluate computation and communication loads over the network, and it accomplishes it by means of a set of special 'Shadow Processes' which are in charge of the time measurements and report to the user. These processes are loaded on the parts of the network whose loads the user wants to monitor.
The shadow processes are of three kinds. The first kind is used to evaluate the computational load on each single processor; thus, on every node whose computational load needs to be measured, such a process must be loaded. The measure is performed in the following way: the process reads the
current time from the local clock, then invokes execution of an elementary calculation task. When this is terminated, it reads the time again and evaluates the time interval elapsed during execution of the instruction. The value thus obtained is sent to a global sorting process residing on the host machine, that takes care of issuing the output on the screen. After executing this sequence of operations, the shadow process suspends itself, thus letting the processor free for normal use by the normal processes belonging to the application. In this way, the shadow process avoids being too heavy an obstacle to normal application's progress.

The shadow process remains sleeping for a fixed time, after which it performs a new cycle. It can be seen that in this way the load evaluation is carried on in a 'sampling process' fashion, in which every sampling event corresponds to execution of the calculation task and evaluation of the time taken for this, while the sampling interval can be thought as the period that the shadow process remains sleeping. This concept is visualized in figure 3.

![Figure 3: Load measurement is similar to a sampling process](image)

To perform evaluation of communication loads over the arcs connecting processors with each other, the same strategy is adopted. In this case, however, two kinds of shadow processes are needed: from now on they will be referred to as 'sender' and 'receiver'.

The sender is placed on one of the two processors linked by a communication connection, and it carries on most of the work. Like the shadow process for computational load measurements, it begins a cycle reading the current time, then it attempts access to the communication link concurrently with the processes of the application that use the same resource, by invoking execution of an elementary message-passing instruction. After this, a second glance at the clock, evaluation of the time difference, and finally forwarding of the obtained value to the global sorting process. Then, the sender becomes inactive and is suspended in order to let the resources free.

The receiver process is placed on the processor being the other end of the communication connection, and all what it does is waiting for the incoming messages from its respective sender and replying with them an acknowledge signal, so that the sender is sure that its communication task has been carried on.

It must be pointed out which are the features owned by the shadow processes that prevent them from corrupting the normal use of the resources done by the application program running on the architecture. First of all, as already said, the shadow processes spend most of the time inactive, and when they do something, it is a very short sequence of instructions. This also implies that the amount of code forming their bodies is quite small, and therefore they also do not waste a lot of storage. Moreover, they are designed so that at most a single sender and a single receiver must be placed on every processor, the former sending messages to all the links departing from the node where it resides, the latter receiving messages from all the incoming links.

Thus, for the sake of load measuring, the shadow processes that must be put onto each node of the network are at most three: one that evaluates computational load on the processor plus a sender and a receiver to measure communication traffic through the link's. Besides them, as mentioned above, a global sorting process must be loaded on the computer serving as a Host to the MIMD architecture, in charge of collecting all the messages sent by the shadow processes, sorting them and sending them to the several separate output windows on the screen.

The output values are shown to the user in form of Athena 'Stripchart' Widgets, plots of the same kind of the well known 'XLoad', contained by an outer window. There is a plot for visualising the computational load on every selected processor, and one for each communication connection, considered unique even if bi-directional. In fact, although on both the end processors NAUTA places a sender and a receiver, the load evaluations made in the two directions are averaged and issued as a single one.

The scheme is shown in figure 4.

7. CONCLUSIONS

The system for load monitoring on MIMD architectures provided by NAUTA only assumes that the MIMD machine has, for every processor of its, a local timer and a fair scheduling mechanism to access to the resources by the processes. Thus, the whole system results quite hardware-independent and therefore it counterbalances a possible lack of precision in the reported values with a good degree of portability on other kinds of MIMD architectures.

Anyway, the issued report seems to be reliable enough to consider the strategy successful. In fact, tests made with processes simulating a real application program, and whose behaviour was of course known a priori, have pointed out that the stripchart plots linearly increase with an increasing number of concurrent processes, and that they are able to recognize different phases of activity during a process' lifetime (computation, communication, suspension).

It must not be expected, however, that the information over the behaviour of processes collected in such a way can give an immediate solution to the problem of load-balancing. In practice, it can result quite hard to distinguish which are the individual contributions of the several processes to the global load reported by each plot, especially when the complexity of the application is high.

Moreover, it must not be thought that NAUTA is an intelligent tool to analyze a parallel program and automatically map it onto a best-fit architecture: it is still up to the user to decide how to configure the network in the
proper way, even if NAUTA helps him to check the progresses made with each attempt.

On the other hand, the graphical interface provided by NAUTA to define the architecture organization is expected to be an useful tool in the user’s hands, resulting easy to understand and providing a very natural approach to the tasks that have to be carried on.

Probably, the aspect that mostly could induce discomforts to the user, is the necessity of showing on the layout window a great number of processors and relative connections, when the network being created is really complex. In addition, in such cases the same problem comes out also in the later phase of load monitoring because, when a lot of nodes and arcs are selected for observation, there is need to display on the screen a great number of stripcharts. Therefore, the user could be a little puzzled, having to keep an eye on the many dynamic entities evolving on the display.

Unfortunately, NAUTA has not been really tested on the field yet, used to develop concrete applications, because it is still in a refining phase. It is hopeful that it will be used soon by several parfile writers, enabling to point out which features could be improved.

References


