AN APPLICATION OF DISTRIBUTED ENVIRONMENT IN FLIGHT SIMULATION

D. Canetta, S. Ceriani, D. Eufri
Agusta S.p.A.
Via Isonzo, 33
21049 Tradate (VA) Italy

Abstract

Today's flight simulators are rather expensive, mainly because only by high-cost supermini computers the required computational power can be obtained. However a different approach can be adopted, using a distributed environment of microprocessors. In our paper we describe this approach to flight simulator realization, taking into account new concepts of parallel computing and the use of transputers. To evaluate the possibility of using these processors in flight simulators, applications to the flight mathematical model and visual system will be described.

1 Introduction

In these last years the computational power of computers has become very high, increasing the number of operations sequentially run in the time unit. This process, carried out maintaining the Von Neumann logic for the computer design, is now approaching to the physical limits of technology. This is the reason that drove to the design of multi-processor supercomputers, that allow for distributing processing.
In this scenario, at the beginning of the 80's, the transputer appears (Rif. 1). This processor was designed taking into account the parallel architecture concepts, however preserving some characteristics of the traditional microprocessors.
The transputer is becoming of great importance in the simulation world, where high computational power is requested and parallel architectures are useful for reducing costs.
In this paper the application of transputers to two critical parts of helicopter simulators are described, that is the mathematical model of the helicopter rotor and the visual system.

2 The Transputer

The transputer is a programmable chip, communicating with other chips of the same kind in parallel networks. Since it was design from the beginning to operate on parallel networks, the transputer offers high performances if well utilized, showing a low overhead (overhead = lost of effectiveness due to the presence of several processors) and low communication time. The difference with respect to the other processors is the absence of the traditional bus, that is usually the communication channel for all the resources of the computer. The bus is the

origin of bottlenecks during program run.
Moreover, difficulties arising in synchronizing and management of data are easily overridden with transputer, because the serial links for communication and relevant instructions for synchronization are implemented on the processor instruction set.
Together with transputer, the Occam language was developed. This language describes any system as a group of processes that operate concurrently and communicate through channels. Occam was developed with transputers, but it is important to note that it can be used on other system, in the same way as transputer can be used without Occam. It is the relationship between Occam and transputers that facilitates network design: a program run on a transputer is formally equivalent to an Occam process, in a manner that transputer network can be described as an Occam program.
As far as the software is concerned, a program is seen as an interconnected group of processes, considered independent units communicating between them through point-to-point channels. By the connection between processes it is possible to build complex systems.
A group of processes is a process as well: process can have internal concurrency. In this way a hierarchy of processes is created inside any process.
In the same manner a group of transputers that operate concurrently can be seen as a single transputer and then from the physic point of view a hierarchy is represented.
From the hardware point of view the communication between transputer is realized through links, that are the physic implementation of the Occam channels.
The traditional kind of transputer is the model IMS T800, that has the architecture shown in Fig. 1.
On the single chip there is:

- CPU 32 bit
- Floating Point Unit 64 bit
- 4 communication link
- 4 kB RAM
- external memory interface
- device interface
- hardware scheduler for concurrent programs granting a switch lower than 1 micro-second

The processor can have 10 MIPS or 1.5 MFlops at 20 MHz.
Other kind of transputers can have different computational power and can be integrated on boards with possibility of static and dynamic memory expansion.
To be all components implemented on chip, the processor must be very small: it is similar to a RISC (Reduced Instruction Set Computer). The processor accesses directly the on-chip memory. When more memory is required, the access is to the external memory through the appropriate interface.
But what really characterizes the transputer is the communication. Each transputer presents 4 links that substitute the traditional bus. A communication link is constituted by two serial lines (one for each direction) and implements a simple communication protocol. The connection between the two transputer is made connecting the link interfaces of the two chips using two serial lines that carry both data and

identifiers.

3 Transputer Network

The network configuration consists of associating the processes that make a program to the processors (Rif. 2). All that is made possible by special instructions of the Occam language.
A particular instruction allows to specify the transputer where to allocate part of the program. The same procedure can also be allocated on different processors.
Another instruction allows to associate the Occam channel to the physical links. Each channel must obviously be positioned in input on a transputer and on output on another one.
It is important to observe that the configuration procedures do not affect the logic behaviour of a program written for a single transputer. The implementation on more transputers can then represent the final phase of a of the software development.
The transputer networks can be built without limit of shape or dimension, with the only limit of the availability of 4 links. There is not an absolute correct network for each application. Each possibility can correspond to a compromize between easy of programming and effectiveness, taking into account the reliability and the final hardware cost. It is clear, in any way, that the choice of a topology must have the target to have max cooperation between transputers, to reach the best performances in speed and precision.

4 The Rotor Model

To really evaluate the performances of transputer, it was employed in two parts that are critical for helicopter flight simulators: the mathematical model and the image generator.
For the rotor model a Blade Element Theory (BET) model was assumed. The faithful reproduction of the rotor dynamics was achieved by dividing the blade in 10 elements. For each element the dynamic laws of motion were written, taking into account the aerodynamic and inertial forces and moments. Forces and moments are then summed together to give the total forces and moments supplied by the rotor.
The code, called BETTI, was at the beginning written in FORTRAN, then translated in Occam. The previous version in FORTRAN was used to check results and to compare the computational time.
Looking at Fig. 2 it is easily understood how an integration step is performed. In input the initial conditions are supplied: data is required to define the geometrical and physical conditions of the aircraft, for example dimensions, areas, aerodynamic coefficients, etc. In particular the aerodynamic coefficients of the wing sections are represented by two profiles, one for the first 7 elements and the others for the remaining 3 elements. Data is supplied for 99 angles of attack for the two profiles. Moreover, a dependence from 5 Mach numbers (first profile) and 7 Mach numbers (second profile) is taken into account. Then kinematic transformations are carried out: for each element velocity and acceleration in the local

reference frame are requested.
At this point an important part of the code begins: the induced velocity calculation. For this velocity two expressions have been adopted: momentum theory and Young method. For the calculation an iterative method is used: iterations are stopped when the error is less than 0.01 or the iteration number is more than 20. Experimental results have shown that 3 or 4 iterations are usual for convergence.
After the induced velocity calculation, the motion equations are solved. The blade is considered to be stiff, therefore flapping and lagging are only allowed. Being two degrees of freedom permitted, two differential equations of second order must be written and solved for each blade. To solve the equations Euler method is used.
Two versions of the BETTI code were written, BETTISER (serial version) and BETTIPAR (parallel version). For parallelization two ways could be employed: a mathematical parallelization, that is to group together parts of the code that for their implementation are easily parallelized, and physical parallelization, that is a parallelization that follows the physical model. The second way was adopted and, where possible, calculation for the four blades was executed in parallel.
Fig. 3 shows the network used for the rotor model. The host computer was a personal computer interfacing the root transputer, The transputer called ´shaft´ deals with the management of the other four transputers, called ´blade 0´, ´blade 1´, ´blade 2´ and ´blade 3´. Each ´blade´ transputer simulates the dynamics of the single blade: the program is the same for the four transputers, except the ´blade 0´ that also deals with input/output.
To obtain a good model of the rotor dynamics the experience has shown that an integration rate of 180 Hz is useful for rotor simulation. This value is also what usually asked by customer requirements. The 180 Hz rate means an integration step of 5.6 ms.
BETTISER reached a performance of 10 times slower than the real time. BETTIPAR, the parallel version, some modification was needed. In fact the memory capacity of each T800 used for blade simulation was too small if the aim is to implement the aerodynamic coefficient tables in the on-chip memory. Then one table only for each blade was kept and the dependence from Mach number was not taken into account. Data was reduced to 1/12 and, in this case, the results were different of about 5% after 1000 steps.
The parallel version reached a very interesting result: an integration step was performed in 8.8 ms (average). This result showed that a good use of the low part of the memory allows for a very short computing time.
To reach the real time, some other modifications were carried out. The number of elements was reduced to 5, but other tests will have to be performed, in order to be able of having a larger number of results to be compared with flight tests and to increase the simulation faithfulness.

5 The Image Generator

Before describing the implementation of the image

generator, a short description of the graphic pipeline utilized for the system will be geven (Rif. 3).
The image to be shown is memorized in a data base in terms of 3D coordinates of the polygon vertex which compose the scene. This scene model is retrieved at the beginning of the graphic pipepline and transformed step by step. The modules are:

- View Transform. It makes the transformation from the data base reference frame (world) to the reference frame with the origin in the view point of the pilot (eye). The objects composing the scene are expressed in a clock-wise reference frame, with the Z-axis up and the eye reference system is counter clock-wise with Z-axis toward the line of sight, X-axis on the right of the observer and Y-axis up. This choice allows the X- and Y-axis to be the axis horizontal and vertical of the screen.

- Clipping. The observer can see only part of the objects present in the scene. Clipping eliminates all polygons outside the vision pyramid of the observer.

- Perspective transform. It makes the transformation from 3D to 2D, that is from the 3D eye reference frame to the 2D screen reference frame.

- Scan conversion. It spots pixels that are within each face starting from vertex that determine the contour. After spotting, to each pixel a value of brightness is associated.

- Shading. It determines the shading of each face, depending on the light source direction.

- Z-buffer. It allows the elimination of the hidden surfaces.

The first simple implementation of the graphic pipeline can was carried out utilizing one transputer for each pipeline step. This configuration utilizes 7 transputers but experimental results showed that it is too slow for real time. A second implementation was then performed utilizing two processors for each of the last three steps of the pipeline. In this case the production of images was 1 per second, far from real time but showing to be in the right way for real time. At this point, however, it was noticed that the bottleneck was the graphic board. For this reason new configurations were not tried, waiting for the new graphic board INMOS G300 Graphics TRAM B419. With this new graphic board, it is expected real time to be met.

6 Concluding Remarks

Transputers have been proved to be effective microprocessors to be employed in simulation applications. The features that make transputers so attractive for these applications are low costs and modularity, that are important features in simulation were costs are usually high and computational power always unsufficient.
However, the transputer is recently appeared on the market and suffer from the poor software tools availability and the need of new boards, for example the graphic board for image generators. We are expecting in the future the elimination of these issues.

References

(1) "The Transputer Databook", INMOS, November 1988.

(2) Laurie Pegrum, "Configuring Occam Programs", INMOS, Technical Note 31, January 1988.

(3) Newmann, Sproull, "Principles of interactive computer graphics", Mc Graw Hill, 1979.

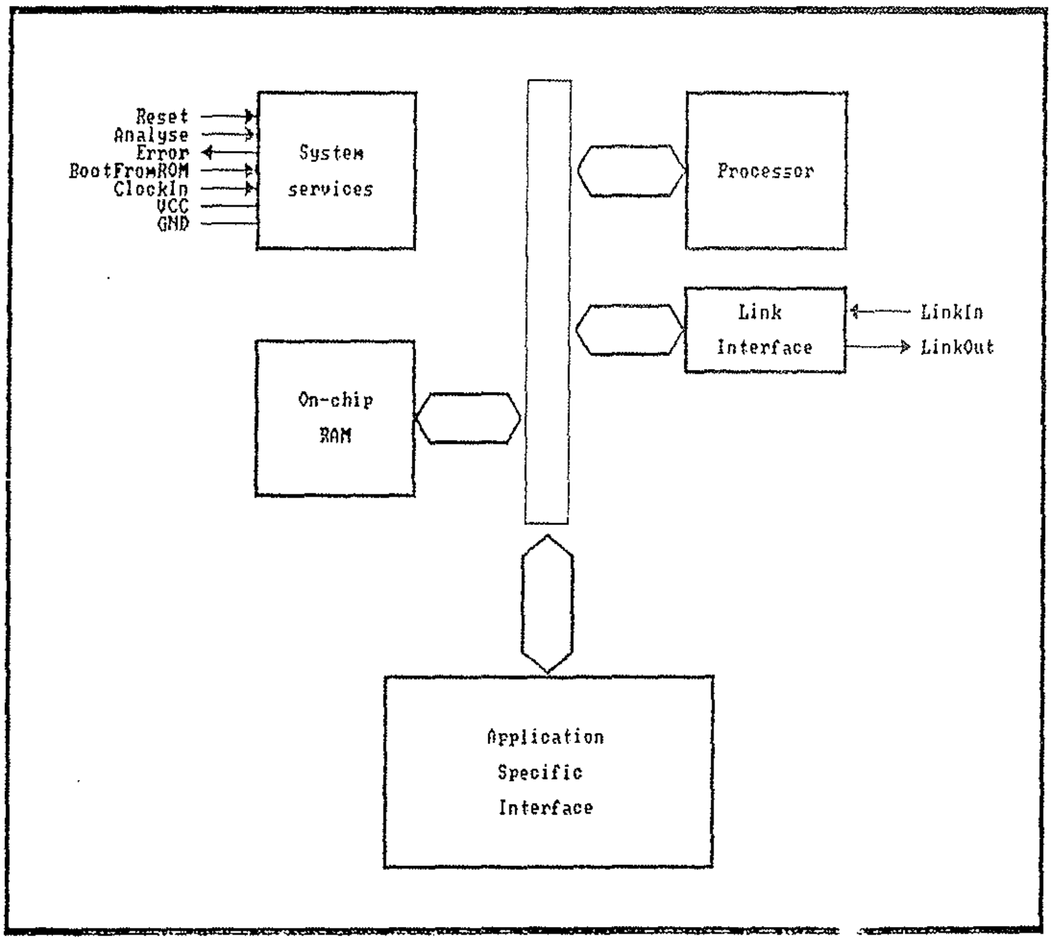

FIG. 1 - Architecture of a Transputer

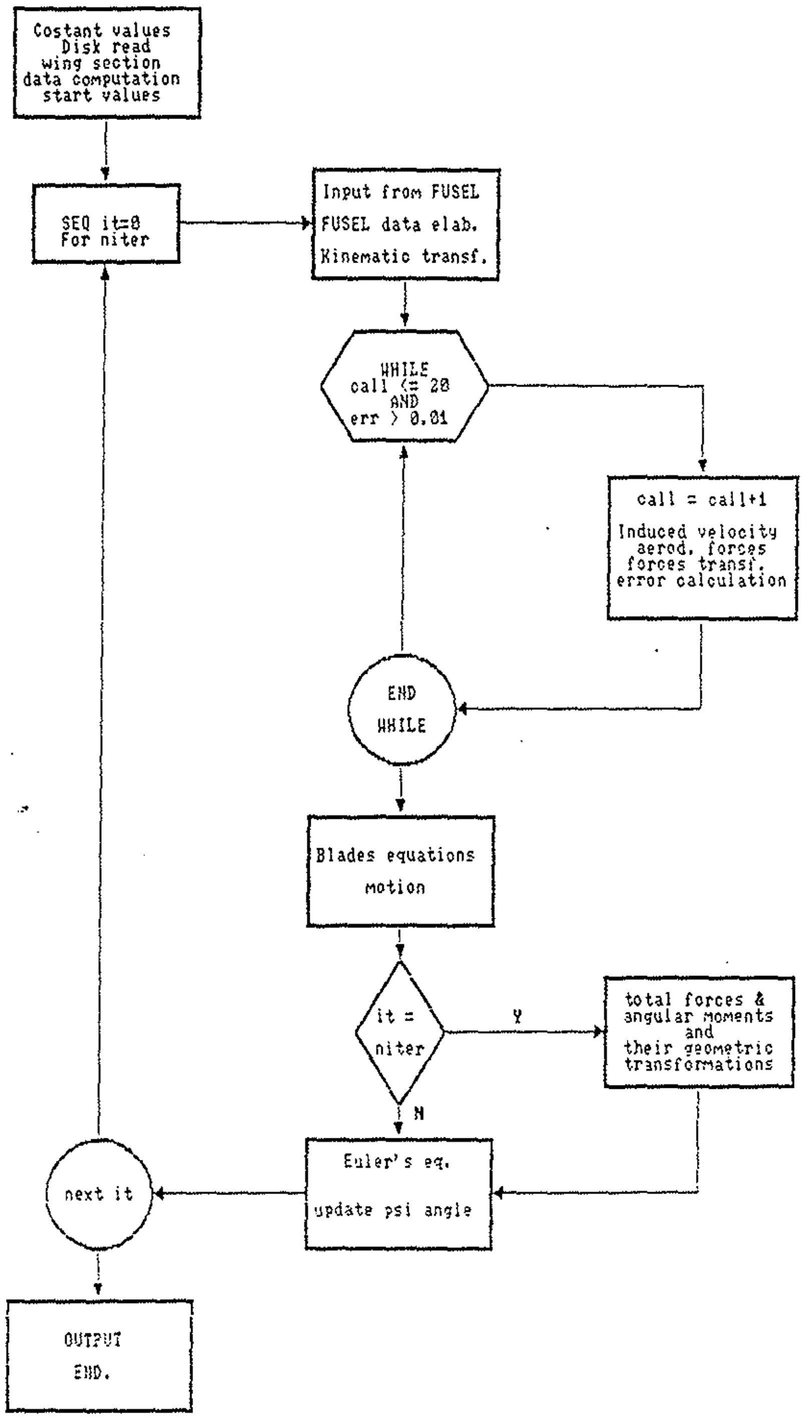

FIG. 2 - BETTI code scheme

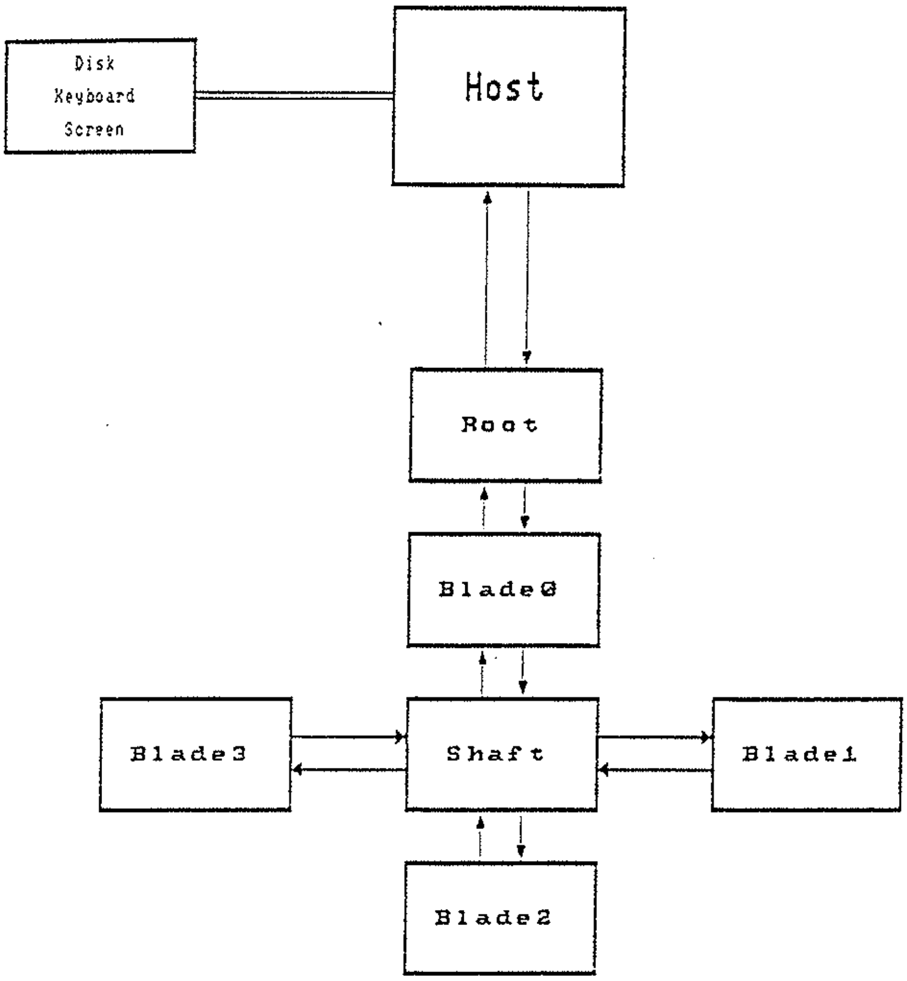

FIG. 3 - Transputer Network for Rotor Model