UDK 681.3.06

Branko Mihovilovič, Peter Kolbezen, Jurij Šilc
Jozef Stefan Institute, Ljubljana

Abstract - The implementation of a raytracing algorithm on an array of transputers is presented. Such system has some important properties. Processing speed is directly proportional to the number of transputers used. The system is remarkably robust namely, individual transputers can be removed from the system while the program is running, and the system will continue to function althought with reduced performance and some loss of data.

KEY WORDS - Transputer, occam, raytracing, performances.

1. INTRODUCTION

This paper describes the implementation of a computer graphics program on an array of transputers. This program was written to provide a demonstration of the performance obtainable by using large numbers of transputers. We used a technique known as raytracing wich can generate very realistic images but requires massive amounts of computer power. This is an ideal application for transputers as the calculation for each picture element on the screen are independent of one another an so can be done in parallel on separate transputers. The entire program is written in occam, however, the main part of the program could have been written in any suitable language. Only those parts of the program which deal exsplicitly with concurrency and the distribution of work are easier to write in occam.

2. BACKGROUND

2.1. Transputer

Concurrent systems can be constructed from a collection of microcomputers which operate concurrently and communicate through point to point serial communication links. The INMOS transputer /1-4/ is the ideal building block of high performance multiprocessor systems which provide maximum speed with minimum hardware.

A transputer contains memory, a processor and a number of standard point-to-point communication links which allow direct connection to other transputers. The processing capability may be general purpose or may be optimized to a specific purpose. The on-chip memory may be extended of chip by suitable interface. A transputer may also have special purpose interfaces for connection to specific types of hardware.

The first transputer available was the IMS T414, a 32 bit RISC microprocessor with a throughput of 10 mips. It has 2 Kbytes of fast SRAM (50 ns) and four serial links. The 32 bit multiplexed bus allows up to 4 Gbytes of external memory to be accessed. Compatible with the T414 is IMS T800 wich includes 4 Kbytes SRAM and floating point arithmetic.

2.2. Occam

Occam /5,6/ is a small and elegant language. It combines the best of contemporary thought about control structures and variable scoping with some radical new structures to handle concurrency. It is based on an model of computation that is different from conventional languages in that it includes the notion of communication, parallel execution, and synchronization in its very structure.

The basic unit of occam programming is a process that performs a sequence of actions and then terminates. There may be more than one proccess executed at any given moment. Occam programs are constructed from three primitive processes: assignment, input and output. The assignment

```
v := e
```

sets the variable v to the value of the expression e. The output

```
c ! e
```

outputs the value of expression e to the channel c. Similarly, the input

```
c ? v
```

sets the variable v to a value input from the channel c. Constructors are used to combine processes to form larger processes. The sequential constructor, SEQ, causes its components to executed one after another. The parallel constructor, PAR, causes its components to be

executed concurrently. Finally the alternative constructor, ALT, chooses one component process for execution which is earliest ready. It is cleare that IF and WHILE constructors are also provided in occam program.

2.3. Raytracing

Raytracing is a now well introduced technique for realistic image synthesis from three dimensional geometric scenes. The basic raytracing algorithm is described in /7/ and is briefly given here. Simplifying somewhat, for each picture element of the rastered image, a ray is trased from the viewpoint into the three dimensional scene to calculate the first intersection with an object. If the object is reflecting or refracting, an appropriate ray is determined by the law of reflection and refraction. These new rays are traced analogously. To calculate shadows, the ray-object intersection points are connected by line segments to the point light sources illuminating the scene. If there is an object intersecting the line segment, the intersection point lies in the shadow of this light source, and its intensity is not taken into account for intensity calculations.

3. LOGICAL ARCHITECTURE

The key problem with raytracing is the relatively high unavoidable basic amount of computation. In the past, two main strategies were followed to process sets of elementary primitives: image decomposition and scene decomposition. In a former case a subset of picture elements of the image are assigned to each of several processors. Every processor has access to the relevant primitives of the scene. In the later case a subset of primitives are assigned to each processor. A processor has access to the relevant rays of the scene.

The calculations performed for each picture element on the screen are completely independent so they can be performed in any ordered and on any number of processors. The example of distributing the work to a number of processors is given in Fig. 1.

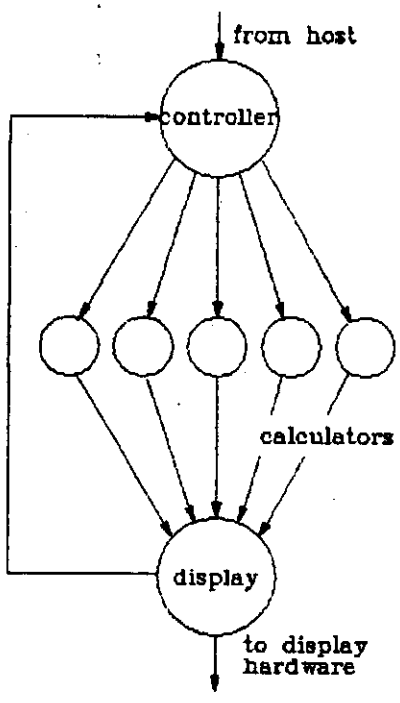

Fig.1: Logical architecture

This solution requires three different processes running concurrently on three, or more, processors:

- control process (controller),
- intersect and shading calculation process (calculator),
- display process (display).

A controller interfaces with the user or host computer to provide a description of the scene beeing viewed and allocate work to processors. A calculator can be replicated any number of times, to render the picture elements. A display collect the results from each calculator and drives the graphic display.

Every calculator is first given the description of the scene and then processing work can be allocated by the controller giving each calculator picture element to evaluate. When the calculations have been completed the results are passed out to the display. The display then informs the controller that there is now a free processor and another picture element is sent out for evaluation.

The amount of computation required varies from picture element to element and this method automatically balances the load amongst the processors and ensures they are all kept busy. An interesting idea here is that the picture elements do not need to be generated in sequence and, if they are generated in some pseudo-random order, a good impression oof the final picture can be obtained well before every picture element has been evaluated. This could be particularly useful in a CAD system where the user wishes to change his view of an object very rapidly.

Note, that this structure is not related to the raytracing algorithem and is suitable for any problem which can be broken into independent parts.

4. PHYSICAL ARCHITECTURE

4.1. Basic design

It appears, a first sight, that the above architecture cannot be maped directly into a network of transputers because of the fixed number of communication links available.

There is a partitioning which can aid the understanding and implementation of the structure of parallelism. The data processed on calculators consist a sequence of values (picture element), then all of the processes can be executed concurrently, even those which process the data in sequence. Alternatively the constructed process can be replicated over a number of calculating transputers each of which will execute the construct on a subset of the data structure as illustrated in Fig. 2. This hardware realisation named proces replication is mapped onto a network of transputers, active data structure is mapped onto the reconfigurable processor array (RPA). Both are ideal for occam process virtualisation. The occam model adopted in the RPA system, uses point to point communications to synchronise processes. A processprocessor mapping is implemented by providing a physical network of transputers, which is isomorfic to the process structure, but only at the chosen point in the hierarchy of the occam program.

It is very simple to arrange for the controller to communicate with any transputer in a network by passing messages through the intervening

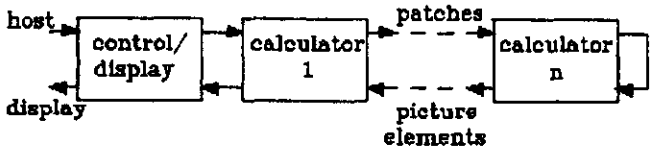

Fig. 2: Physical architecture

transputers. For simplicity, the raytracing algorithm was mapped on to a linear array of transputers /8/.

In Fig. 2 where the basic architecture is shown we see that each transputer link implements two occam channels, one in each direction. Therefore, this mapping uses only two of the four links available on a transputer. Control and display processes are executed in parallel on one transputer (control/display transputer) while the rest of the transputers do the intersection and shading calculation processes (calculating transputers). The control/display transputer also does these calculations and the same, parameterised, program is loaded into every calculating transputer. Such method of mapping processes requires that each calculating transputer also execute routing processes, i.e. commands and data are passed from the controller along the array and results are passed back for display. This linear connection of transputers implies some sort of command protocol for identifying the nature and destination of data, consequently the routing process on each transputer only needs to decide whether a message is to be accepted locally or passed on to be dealt with elsewhere.

4.2. The control/display transputer

There are two processes executed by the control/display transputer (Fig. 3):

- sendPatches,
- loadBalance.

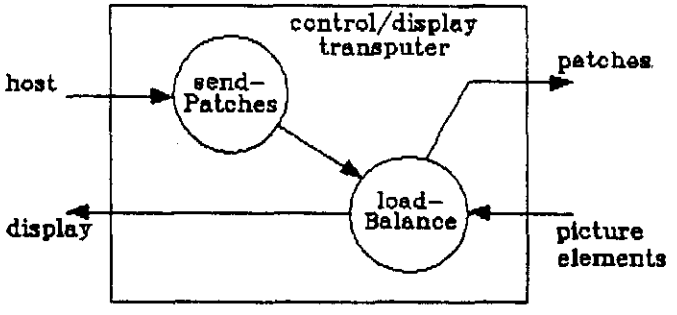

Fig. 3: Processes on control/display transputer

SendPatches interfaces to the host computer to receive the description of the scene being modelled and other commands. It passes the world model out to all the other transputers and then sends out requests for picture elements to be evaluated. Square areas of the screen, called patches, rather than individual picture element, are given to each transputer for two resons:

a) to give better ratio of calculation to communication;
b) to enable segments of data to be transmitted.

A segment communication transmits an array of words as a single operation. This has two big advantages over the transmission of individual words. Firstly, there is the same processor overhead for setting up the links to transmit a single word as for a million words. This better exploits the autonomous transputer links and allows the processor to continue calculating at very nearly full speed. Secondly, it gives a better ration of communication to computation.

LoadBalance coordinates the sending of data to the other transputers and the display of the generated picture elements. The first thing it does is to determine the number of transputers in the system. This is done by sending a count, incremented by each transputer, around the loop. If there are n transputers then loadBalance passes on primitives containing 2n picture elements from the process sendPatches and then waits until a result is returned before passing out another request.

4.3. The calculating transputer

The work of each of calculating transputers is organised as three processes (Fig. 4):

- througput,
- render,
- feedback.

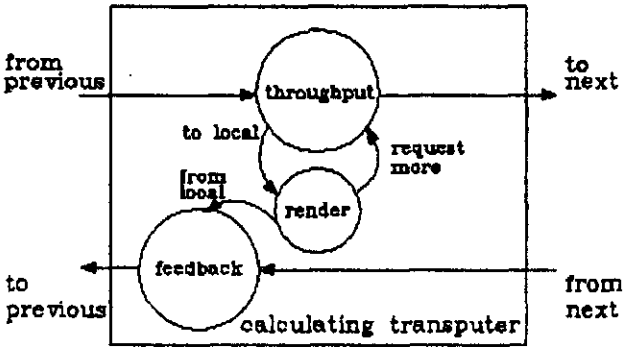

Fig. 4: Processes on the calculating transputer

The throughput process receives patch requests from the previous transputer and either pass them on to the next or keep them for processing locally. It is also able to hold one request in an internal buffer so, initially, it accepts two requestes. The first is passed immediately to render process for evaluation and the second is held until needed. Any further patches received are passed on to be evaluated elsewhere. As soon as render has finished the computation of its patch, throughput passes it the buffered patch to work on and is then ready to accept another. Since the time taken to compute a patch is less than the time before throughput receives the next patch request, the render process is kept busy.

The render process is given patches to evaluate, i.e. it does all the calculations to find intersections, build the bundle of rays and then traverse this bundle to get the final picture element value. When the picture ele-

ments in the patch are evaluated then another patch is requested from throughput and the picture elements are passed out to the feedback process. This is a completely sequential part of program which can be written in any standard programming language.

The feedback process multiplexes the local result and those received from other transputers and passes them back towards the display transputer via the shortest route.

4.4. Occam implementation

As we have said the transputer architecture simplyfies system design by the use of processes as standard software and hardware building blocks. The ability to specify a hardwired function as an occam process provides the architectural framework for transputers with specialized capabilities (e.g. graphics). The required function (e.g. graphics drawing and display engine) is definied as an occam process, and implemented in hardware with a standard occam channel interface.

The occam program of the proposed architecture (see Fig. 2) contain:

- the description of the entire transputer system,
- the control/display transputer program,
- the calculating transputer program.

For simplicity, only the essential outline of the mentioned above occam programs is given here.

The system description is as follows:

```
VAL number.transputers IS 24:
VAL last IS number.transputers - 1:

CHAN host, display, loopback,
     forward[number.transputers],
     return[number.transputers]:

PLACED PAR

-- transputer 0 = control/display transputer

 PROCESSOR 0 T4
       -- data from host
  PLACE host       AT link0in :
       -- to display
  PLACE display  AT link0out:
       -- patches out
  PLACE forward[0] AT link1out:
       -- picture element value back
  PLACE return[0]  AT link1in :

  control ( host, display,
            forward[0], return[0] )

-- the main body of the pipeline of calculators

 PLACED PAR i = 1 FOR number.transputers - 2
  PROCESSOR i T4
       -- patches in
   PLACE forward[i]  AT link0in :
       -- picture elements out
   PLACE return[i]   AT link1out:
       -- patches out
   PLACE forward[i+1] AT link1out:
       -- picture elements in
   PLACE return[i+1]  AT link0in :

   calculate ( forward[i], forward[i+1],
               return[i+1], return[i] )

-- the last transputer is a special case as it
-- has no one else to talk to. The fact that
-- the channel 'loopback' is not placed means
-- that an internal ("soft") channel will be
-- created. In fact this channel is never used
-- but is required as a parameter.

PROCESSOR last T4

 PLACE forward[last] AT link0in :
 PLACE return[last]  AT link0out:

 calculate ( forward[last], loopback,
             loopback, return[last] )
```

The program running on the control/display transputer is:

```
PROC control ( CHAN fromHost, toDisplay,
                    toCalculators, pixelsIn )

 ... definition of sendPatches procedure
 ... definitions of loadBalance procedure

 CHAN data:
 PAR
  sendPatches ( fromHost, data )
  loadBalance ( data, toCalculators,
                pixelIn, toDisplay )
```

Finally, each of the calculating transputers runs the following program:

```
PROC calculate ( CHAN fromPrev, toNext,
                      fromNext, toPrev )

 ... definition of the throughput procedure
 ... definition of the render procedure
 ... definition of the feedback procedure

 CHAN toLocal, fromLocal, requestMore:
 PRI PAR

  -- run these at high priority for
  -- fastes response to messages

  PAR
   throughput ( fromPrev, toNext, toPrev,
                toLocal, requestMore )
   feedback ( fromLocal, fromNext, toPrev )

  -- and this is at low priority

  render ( toLocal, fromLocal, requestMore )
```

5. CONCLUDING REMARKS

5.1. Performances

Whithout doubt, the processing speed of the system is directly related to the number of transputers used. A number of factors contribute to this aspect of the system. Most of these were carefully worked out design decisions but one had to be determined empirically. The transputers require only two words of data to specify the position of the all picture elements in the patch. If the work were distributed on a picture element by element basis then two words of data would be required for every element. This would mean a worst ratio of communication to processing. A more intelligent approuch use of segment communication for data (paragraph 4.2.). These means less processor overhead per word sent and allows a greater amount of concurrency between the link engines and the processor. It is important to say that the message routing processes are had to be run at high priority to ensure that incoming messages can be examined and forwarded immediately it is received. Carefully ordered priority in the ALT constructs of these processes are en-

sure that patches are returned to the control transputer as quickly as possible. Withholdings in the work in throughput and feedback processes, are reduced by introducing software buffers into two input channels of these processes. Channel buffers are frequently used, and easy to implement, in occam programs. Buffers introducing made a significant difference to the performance from

$$speed = (transputers + 1) * K/2$$

to very nearly

$$speed = transputers * K,$$

where K is the performance of a single transputer.

5.2. Robustness and reliability

The system described above is already remarkably robust. It should be possible to exploit the number of transputers with some degree of redundancy. If a calculating transputer fails then the system will progressively deadlock only if the neighbour, on the controller side, attempts to communicate with it. In order to make the system more robust it must be possible to detect when a failure has occured. This reqire the using a timeout on all communications. Secondly it must be possible to ensure that, if a communication does fail, all the input and output processes will terminate. These desired function are performed by a number of predefined procedures which allow an input or output to be attempted within a time limit, an recovery from a failed communication /9/. The use of these procedures means that failure of a transputer can be detected by its neighbour. The controlling transputer could then be informed and so take action to recover or regenerate the lost data. Detection of the failure of a transputer implies that facilities could be added to allow the defective transputer to be bypassed. As we remember, on each transputer two communication links are unused. So this can be done with no extra hardware in such a way that the precedent of fail transputer switches it to the other link to communicate with the next transputer along.

6. REFERENCES

/1/ Whitby-Strevens, C., The Transputer, Proc. 12th Annual Int'l Symp. Computer Arch., Boston, Massachusetts, 1985, pp. 292-300.

/2/ Taylor, R., Transputer Communication Link, Microprocessors and Microsystems, Vol.10, No.4, 1986, pp. 211-215.

/3/ Mihovilović, B., S. Mavrič, P. Kolbezen, Transputer - The Basic Componnent of Multiprocessor Systems, Informatica, Vol.10, No.4, 1986, pp. 81-84.

/4/ Mihovilović, B., P. Kolbezen, J. Silc, Communicating Processes in Transputer Systems, Informatica, Vol.11, No.2, 1987, pp. 74-77.

/5/ May, D., R. Taylor, Occam - An Overview, Microprocessors and Microsystems, Vol.8, No.2, 1984, pp. 73-79.

/6/ Curry, B. J., Occam Solves Classical Operating System Problems, Microprocessors and Microsystems, Vol.8, No.6, 1984, pp. 280-283.

/7/ Whitted, T., An Inproved Illumination Model for Shaded Displey, Comm. ACM, Vol.23, No.6, 1980, pp. 343-349.

/8/ Packer, J., Exploiting Concurrency; A Ray Tracing Example, Tech. Note 7, INMOS Ltd, Bristol.

/9/ Shepherd, R., Extraordinary Use of Transputer Links, Tech. Note 1, INMOS Ltd, Bristol.