A Configurer for Parallel Programs

AUTHOR NASSER SULEIMAN KARABASH

SUBMITTED FOR THE AWARD OF
MASTER OF ENGINEERING

SUPERVISOR Sean Marlow Ph D

School of Electronic Engineering
Dublin City Unievsity

MARCH 1991

The contents of this thesis are based on my own research

CONTENTS

Acknowledgment:

I would like to thank Dr. S. **Marlow** for his support and encouragement, which helped me to complete this work.

My gratitude to the **Scientific Studies and Research Centre (SSRC)** in **Syria** for it's guidance and sponsorship.

Abstract:

Making parallel systems easy to use, and parallel programs easy to write and run, are two major aims for parallel systems designers.

This work describes an automatic configurer which attempts to lessen the effort a user should put to configure and run a parallel program on a network of parallel processors

The configurer configures the program, as required by the operating system and produces the configured source of the program

The network is also configured by the configurer to meet the communication requirements of the configured program

Introduction:

Developments on parallel programming languages are directed to help the programmer to write a parallel program without having to know how the concurrent processes are going to be distributed on the processors or how they are going to communicate with each other.

Parallel architectures provide a high level of computing power although ordinary processing elements are used [1]. Different topologies are being used in parallel architectures but no single one has proved to be better than all the others. Each topology is good for a specific type of problems but may be bad for others.

Performance problems, possibly caused by bad partitioning of a program, result in a very big amount of communication between the different partitions (concurrent processes). In the worst case, running problems like long waiting times or even deadlocks may occur.

The maximum performance is got if all the processors are kept busy as much as possible. To achieve that, a balance in the computational and communication load between all the processors should be achieved, which is not an easy task. Predicting or estimating the required computation time of a process is not an easy task [21].

To configure a program to run on a processor network is to assign the concurrent processes of the partitioned program to the processors and assign the communication channels between the processes to the processors' links.

Information about how many processors there are in the network, how the processors are connected, how the messages are passed between the processors, how many parallel processes there are, and the communication requirements between these processes should be known to enable the configurer to distribute the processes in an efficient way.

This project addresses the detection of the processors in the network and the parallel processes in the program, the generation of the communication map between the processes, the assigning of the parallel processes to the processors (transputers) in the network, the creation of the required interconnections between the processors to meet the interprocess communication needs, and generates a new source program with all the correct configuration data added.

In Chapter One, different interconnection methods used in parallel processing machines are discussed to show the relationship between the applications and the processor network topology. A general idea about the Occam programming language and the transputer is then given. Different message passing techniques used in transputer networks are discussed. Some transputer based systems of different types of topologies are described.

In Chapter Two the procedure of configuring a program to run on a processor network is discussed, and then the TDS configuring procedure is described followed by a case study to clarify the problem the project is trying to solve.

Chapter Three discusses the design of the configurer in detail.

Chapter Four contains the results of configuring programs requiring different types of topologies by the configurer program. This chapter shows how to use the configurer and demonstrates the facilities it provides.

In Chapter Five suggested improvements and further work are mentioned followed by the conclusions.

Chapter 1

Processor Networks

1-1 Introduction:

Partitioning a program into parallel processes and defining the communication pattern between the processes are the steps which come before the configuring step.

Different types of interconnection schemes used in parallel machines are described to show the relationship between the machine architecture and the application program partitions. Selected transputer based architecture examples are mentioned separately to show the variety of network architectures that could be build using the transputer as the basic processing block.

This chapter gives parallel programs designers information which will help them in chosing algorithms, processor network topologies and communication techniques.

Two categories of topologies could be distinguished:

1 - Static connection topology.

2 - Dynamic connection topology.

Static topologies are suitable for applications whose communication pattern can be reasonably predicted. Dynamic topologies, although they are more expensive, are suitable for a wider class of applications.

Static topologies machines are special purpose because high performances on these machines are achieved when dealing with specific problems depending on the match between the topologies and the algorithms being used in solving these problems.

1-2 Static connection topologies:

A static network topology is one in which none of it's connections can be changed after the machine is built. The simplest connecting scheme is a ring (Figure 1-1-a). On the other hand, the most complicated but powerful and ideal connecting scheme is the all-to-all scheme Figure (1-1-d).

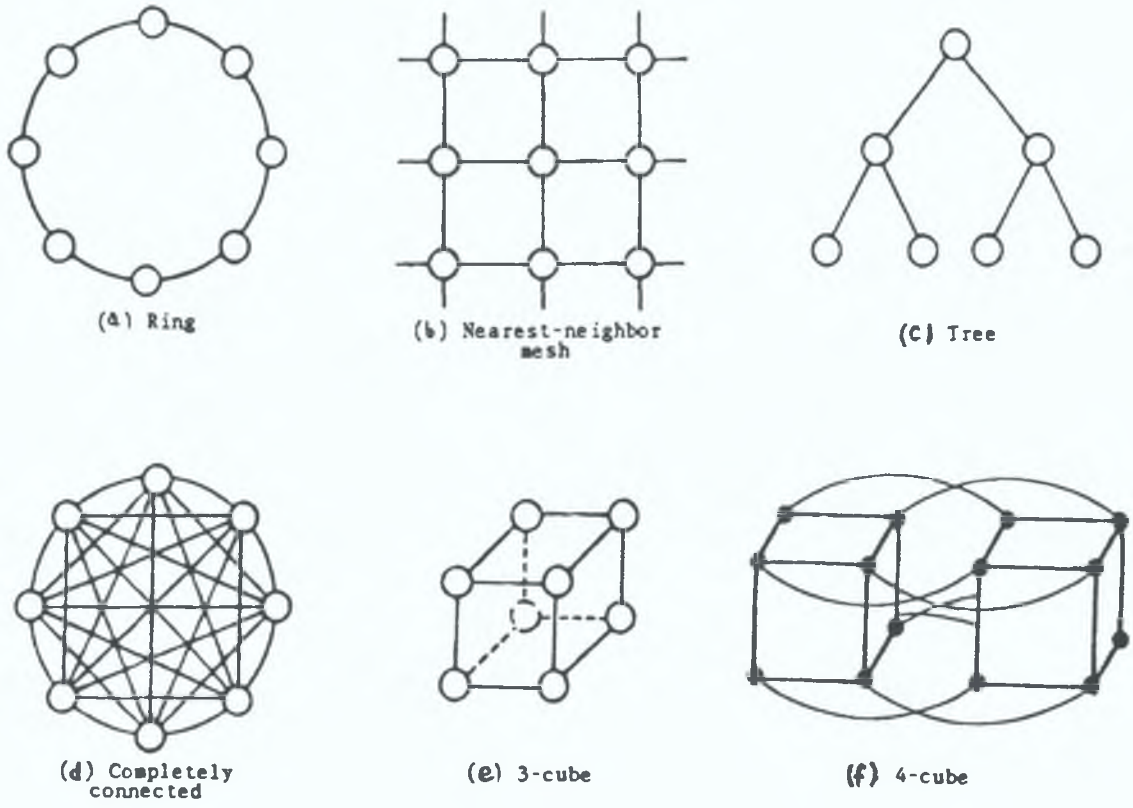


Figure (1-1 a-h) [7]

Different static configurations have been used in parallel systems with an intermediate number of nearest neighbours taking into consideration that the choice of the connection scheme is compromise between wireability and performance [9].

Special purpose systems with a relatively large number of computing nodes are expected to give good performance results when dealing with problems which could be divided into parts which have light and predictable communication patterns, normally data exchanges among neighbouring nodes.

1-2-1 Rings:

The nodes are arranged in a closed line "Necklace" where adjacent nodes are connected. Nodes are of degree 2, Figure (1-1-a) illustrates this scheme. Ring topology is used in systems like the ZMOB [13], which was designed to support image processing [14] and artificial intelligence [15] applications.

1-2-2 Meshes:

The nodes are arranged in a mesh of N dimensions. The interior nodes are of 2N degree and the degree(s) of the nodes at the edges and the corners differs from machine to another. Figure (1-1-b) illustrates a 2-dimensional mesh scheme. Array processors like the MPP and DAP [3] provide a large computing power required for massive computing applications, e.g large matrix operations.

1-2-3 Binary Trees:

The nodes in the basic binary tree are arranged in levels, each level contains double (or half depending on the direction) the number in the previous level. The interior nodes are of degree 3, the root nodes are of degree 2 and the leaves are of degree 1. Figure (1-1-c) illustrates this scheme for three levels. The scheme is used in Reduction machines like FFP [16] and expert systems machines like DADO [17].

1-2-4 Fully connected:

Each node is directly connected to all the others in the network which means all the nodes are of degree N-1 in a network containing N nodes. The total number of connections needed in this scheme is N(N-1)/2 connection.

Figure (1-1-d) illustrates an all-to-all network for N=8.

1-2-5 Hypercubes:

It is a mesh scheme but has a special way of connecting and numbering the nodes. An n-dimensional Hypercube will contain number of nodes N=2 where n is the node degree also. If the nodes are numbered starting from 0 to N-1, by using the binary representation of the nodes numbers and connecting the nodes whose numbers differ by a power of 2, the result will be an n-dimensional Hypercube. Figure (1-1-e,f) illustrates a 3D and a 4D cubes respectively. The Connection Machine which uses a 12-dimensional hypercube [18] is suitable for a broad class of computational problems and knowledge processing.

Many other schemes are used like the Linear Array, the Systolic Array, the Chordal Ring, Shuffle-Exchange and many others. New schemes are always expected to appear trying to increase the performance of the machines and to meet the requirements of new applications.

1-3 Routing algorithms:

Packet switching is used in all of the topologies mentioned above. A routing algorithm will be suggested to find a path between node A and node B for data routing.

1-3-1 Rings:

In a ring of N node clockwise numbered from 0 to N-1 the minimal length path from A to B could be found depending on the table:

Conditions	Length	CW/CCW
(A>B) AND ((|B-A|)>N/2)	N - |B-A|	CCW
(A>B) AND ((|B-A|)<N/2)	|B-A|	CW
(A<B) AND ((|B-A|)>N/2)	N - |B-A|	CW
(A<B) AND ((|B-A|)<N/2)	|B-A|	CCW
|B-A| = N/2 (N is even)	|B-A|	SAME

1-3-2 Meshes:

Routing on a mesh of K dimension is performed one dimension at a time. In a 3D mesh suppose A has the coordinates (a,b,c) and b has the coordinates (x,y,z), a path from A to B is constructed from three paths each of these paths is parallel to one of the dimensions so the route will move along the first dimension to reach the node coordinated at (x,b,c) then it will move along the second dimension to reach the node coordinated at (x,y,c) and then the last part of the path is to move along the third dimension to reach the destination B at (x,y,z).

1-3-3 Binary Trees:

The simplest way is to start from A and then up the tree until the first ancestor of B is found, then down to B. This algorithm can be implemented simply depending on the numbering system of the nodes.

One way of numbering is to assign a number to the node, say (n), and assign the numbers (2n) to it's left child and (2n+1) to it's right child and the root is given the number (1). The path from A to B due to the above algorithm depends on finding an ancestor of B starting from A. Using the binary representation of the numbers of A and B, the number of the first ancestor of B (starting from A) is the result of a chain of logical shift right operations on both A and B until they are equal. Going down to B, after reaching the ancestor, is easy by ignoring the ancestor number in B (from the MSBs) and going down one level for each bit left. Going Right or left depends on the bit; if it is 1 go right and if it is 0 go left. When all the bits are used B will be reached.

1-3-4 Fully connected:

All the nodes have direct paths with all the others, Figure (1-1-d).

1-3-5 Hypercubes:

Since the hypercube is a special case of mesh where each dimension contains only two nodes, the binary numbers of the nodes are actually the coordinates of the node and each bit represents a dimension. In order to make use of this property, performing an Exclusive OR between the numbers A and B will result in a number which contains a 1 in precisely the dimensions along which the data must move.

1-4 Dynamic connection topologies:

A dynamic topology network is one in which all connections can be changed after the machine is built. The goal of using a dynamic topology is to provide the possibility of connecting any node to any other one directly, using circuit switching, without the need to pass through other nodes which have nothing to do with the data being transferred.

1-4-1 Switching networks:

Multiple segments communication path interconnected by switches are used to make complete paths between the nodes, these paths are changed as necessary to cope with the communication traffic among the nodes.

1-4-2-1 Crossbar networks:

A crossbar network enables any node to have a direct path with any other non-busy node at any time with a constant delay which is basically the switching time of one switching element. Crossbar networks have an advantage over the all-to-all network in that the connections it needs are of order N not N . The difficulty with crossbar networks is the number of the crosspoint switches. N crosspoint switches are needed for a network of N nodes and that complicates the design of the crossbar and makes the producing cost high rather than putting limits on the size of the crossbar.

Using current technology, designing large systems with large number of nodes is not yet feasible [10].

1-4-2-2 Single and Multistage interconnection networks:

In order to solve the problem of providing fast, reliable and efficient communications in large parallel processing systems at a reasonable cost many different networks were designed to replace the crossbar. The replacement can be constructed from single or multiple stages of small and cheap crossbar switches.

-In a single-stage network data (messages) may have to be passed through the switches several times before reaching their final destinations. In a multistage network only one pass through the stages is needed. the function of a multistage network can be performed by a single-stage network and the delay will still be the same but in the single-stage network a new wave of messages could not be accepted before the preceding wave has done all the needed passes which is not the case in the multistage networks where a new wave is accepted each time the preceding wave has moved on to the next stage.

The Omega network is an example of the multistage networks based on the shuffle-exchange permutations [19].

The network consists of 2X2 crossbar switches, each of the crossbars can perform four passing functions:

1) Straight-through, Figure (1-2-a).

2) Criss-cross (swap), Figure (1-2-b).

3) Upper broadcast, Figure (1-2-c).

4) Lower broadcast, Figure (1-2-d).

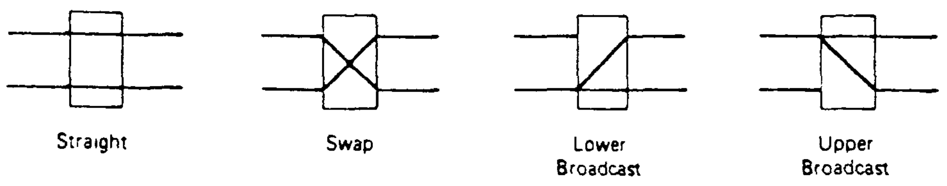


Figure (1-2) [11]

The switches are arranged in an array of N/2 rows by Log_2 N columns and are interconnected in a perfect shuffle. The switching method is packet switching where messages are sent through the network in the form of data plus an address which is read by each switch and then the packet is forwarded to the next appropriate switch according to some control scheme until it arrives at it's destination.

The Omega network uses a distributed control scheme that takes advantage of a simple message routing algorithm which is suitable for both SIMD (synchronous) operation because many different permutations could be performed in a short time and MIMD (asynchronous) operation in which connection requests are issued dynamically.

Many other networks are discussed in [11] and [12] in some detail.

1-5 Transputer networks:

1-5-1 The Occam programming language:

Occam is a high level programming language designed to support concurrent programming [30].

As a concurrent programming language an Occam program is built from a number of **processes** running **sequentially, concurrently** or in **parallel.** In the case of concurrent and parallel processes inter-process communications are done through Occam channels.

Occam has **three primitive processes** :

Process function	**Operator**	**Process Format**
Assignment	=:	var. := exp.
Output to channel	!	chn. ! exp.
Input from channel	?	chn. ? exp.
Where var. is a variable name. exp. is an expression. chn. is a channel name.		

To form constructs Occam processes could be combined in three different ways depending on the execution timing relationship between the processes, these ways are :

Timing Relationship	**Operator**
Sequentially	SEQ
In Parallel	PAR
Alternatively	ALT

Parallel processes could be executed in parallel if they are assigned (placed) to different processors otherwise they will be executed concurrently on the same processor (pseudo-parallel execution).

Alternative execution performs one process out of a number of processes guarded by inputs. This process is the one guarded by the ready input.

Inter-process communication in Occam is synchronous, and can only be done through channels as shared memory is not allowed in Occam. The communication will only take place if both the sender and the receiver are ready to communicate; other-wise the ready to communicate process will be suspended till the other process is ready to communicate [31].

Concurrent programming languages have two major categories of applications. The first is designing multi-tasking programs for single processor systems. The second is designing programs for multi-processor systems where the program processes will run in parallel. Occam supports both categories and is used in the implementation of parallel algorithms [32],[33] in many fields like FFT, signal processing, image processing and display, industrial control, process automation, and simulation.

1-5-2 The transputer:

The transputer is a programmable VLSI device designed to support the Occam model of concurrency.

To run an Occam program which could be constructed from concurrent processes, the transputer has to provide concurrency run-time functions like process scheduling, queue manipulation, context switching, process suspending and releasing. These functions are directly supported by the transputer's hardware and microprogram.

Concurrency is achieved on a single transputer by sharing the processor time between the concurrent processes.

The IMS T800 (Figure 1-3) integrates a 32-bit microprocessor, a 64-bit floating point unit, four standard transputer communications links, 4K Bytes of on-chip RAM, a memory interface and peripheral interfacing on a single chip using a 1.5 CMOS process [22].

The 64-bit floating point unit provides single and double length operations according to the ANSI-IEEE 754-1985 standard for floating point arithmetic and is able to perform it's operations concurrently with the processor. The transputer uses a DMA block transfer mechanism to transfer messages between the memory and another transputer product via an INMOS link. The link interfaces and the processor all operate concurrently allowing the processing to continue while data is being transferred. The 4K Bytes of on-chip RAM provide a maximum data rate of 80 MBytes/Sec with access from both the processor and the links. The external data and address buses are multiplexed on one bus and provides data at rate up to 26.6 MBytes/Sec.

The IMS T800 has a microcoded graphics capabilities for block moving at the speed of memory.

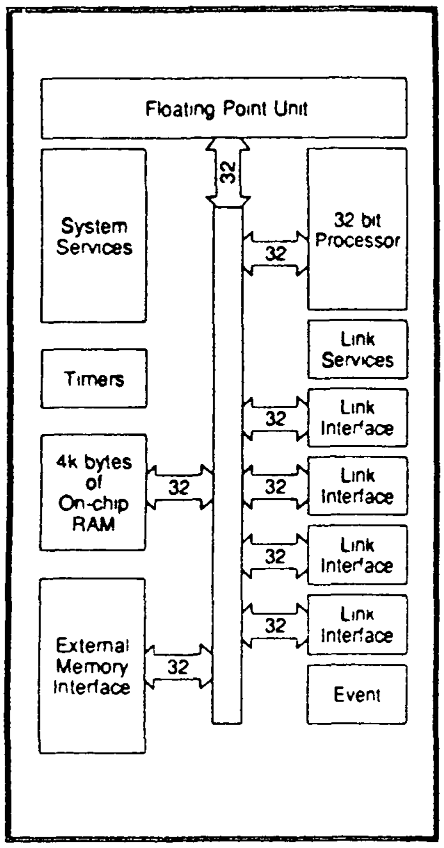


Figure (1-3) Transputer architecture [22]

1-5-3 Message passing techniques:

1-5-3-1 Through routing:

This way of message routing would allow processes running on non-neighbouring processors to communicate.

A routing process is allocated on each processor in the network and has the map of the whole network to be able to forward the messages on the right route.

Buffering is essential for this method because the messages are not being received by the destination. In the Occam model, sending and receiving are synchronous events between the source process and the destination process, while here the sending process can proceed without the message being received by the real destination because the message is received by the routing process. How long the message will take to arrive to the destination, depends on how much the routing processes along the route from the source to the destination are loaded.

To avoid the proceeding of the sender before the message being received by the destination, the sender is blocked waiting for an acknowledgment message to come from the destination. The time the sender speds waiting for this message to come is obviously wasted and still the synchronization is not restored [37] because the destination will proceed.

1-5-3-2 Channel multiplexing:

In the cases where the four channel-pairs of the transputer are not enough and more channels are needed, channels could be multiplexed onto transputer links.

A multiplexing process and a demultiplexing process are required for each link. This method is useful when more than one independent process is running on the same transputer and have to communicate with other processes on another neighbour transputer. The problem of losing synchronization is still there because the messages are still being received by a process other than the destination. The acknowledgment technique could be used to guarantee message arrival.

1-5-3-3 Dynamic link reconfiguration:

Synchronization is only granted when the sender and the receiver are communicating directly.

A solution to the problem created by the limited number of transputer links and the communication between processes on non-neighbour processors is the dynamic topology [24],[28] where the processes can request from the topology controller to create a connection with the other process on the other transputer. This technique enables any process on any transputer to communicate with any other process within the Occam model.

Routing algorithms should always avoid deadlocks by providing the required precautions. Deadlocks are out of the focus of this thesis; refer to [20] for a detailed design of deadlock free networks.

1-5-4 Transputer network topologies:

A transputer network consists of a number of

transputers interconnected via the serial links of the transputers. Transputer networks need less hardware to be implemented because the communication links are serial and need only two wires (tracks on PCBs) one for input and one for output. Also transputers can be directly connected via the links without any additional hardware (driving or interfacing).

As a powerful processor and very suitable for parallel architectures, Transputer based parallel processing systems are being used in a wide range of applications.

Transputer based systems with all types of topologies, Static, Dynamic, and Mixed are available.

1-5-4-1 Static topologies:

1-5-4-1-1 Rings:

- Kilonode:

The Kilonode system [4] is expandable modular and the maximum configuration contains up to 1000 transputer based modules. The interconnection scheme used in the Kilonode is a ring where all the nodes (modules) are connected via the links of the transputers. The system consists of a host computer running the application, a large disk, an SCSI or ethernet interface to the host and the Kilonode.

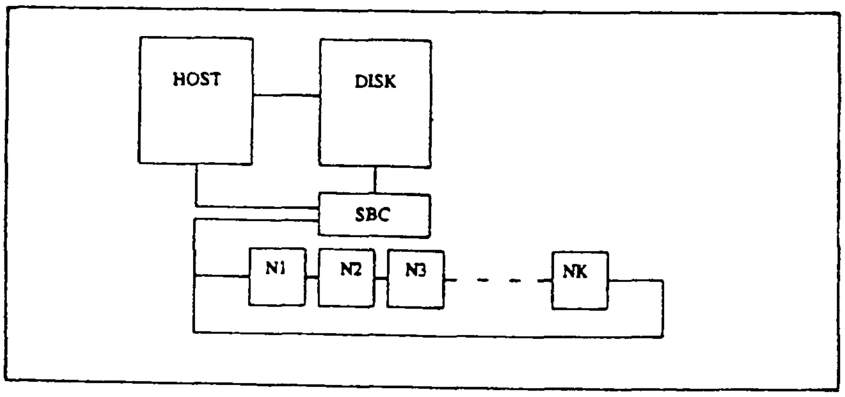


Figure (1-4) [4]

Figure (1-4) shows the architecture of the system where SBC is a very fast Single Board Computer which accesses the disk to read the data and automatically apportion it among the nodes. The data is passed to the nodes from the SBC through the links. Through routing technique is used for data exchange. The system supports ultra-large computing problems like matrix inversion, FFT, circuit analysis.

1-5-4-1-2 Meshes:

- IBM Vector multi-processor:

The interconnection scheme used in the Vector family of transputer based multi-processor systems is 2D mesh. Figure (1-5) shows the overall architecture of the system.

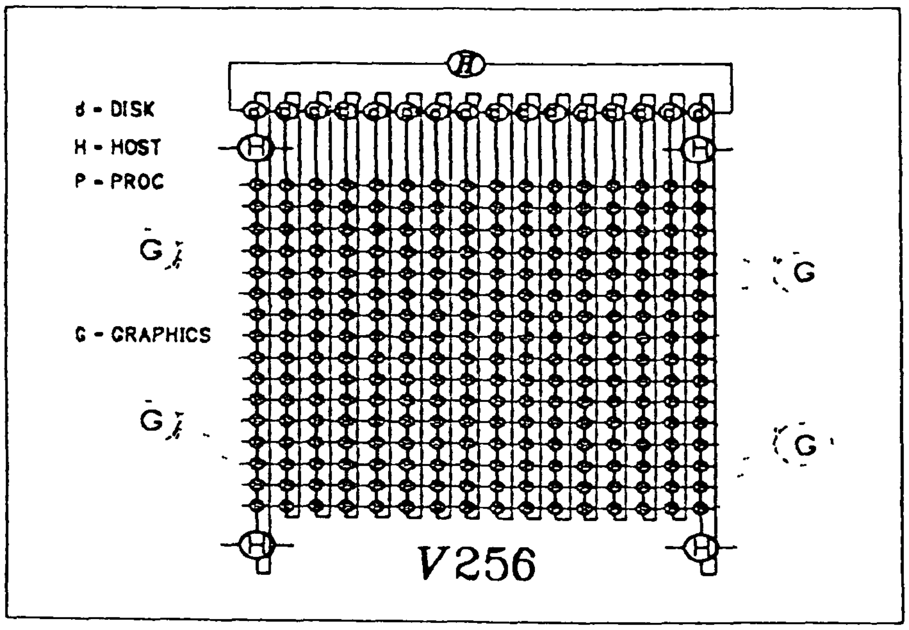


Figure (1-5) [23]

The V16 is the smallest containing 16 nodes and V256 is the largest in the family containing 256 nodes and a total of 1G Bytes of memory. The

V256 is partitionable. Four concurrent users and a superuser could share the system each has his own isolated partition. A user can own a host, an arbitrary shaped multi-processor piece of the network and one graphics node. Special hardware is used to prevent users from interfering with each other. The 16 disk nodes are owned by the superuser and act as servers, handling all file-system requests from the users [23]. The nodes are built around the IMS T800 transputer and communicate with each other via the transputers' links. Messages are passed using the through routing. Large knowledge bases is one of the system's applications.

1-5-4-2 Dynamic topologies:

All the interprocessor connection can be changed in this type of topology. The network may be configured for a certain application before the execution of the program onto the network starts and during the execution the topology may remain as it was configured or changes may occur during the execution (Run-time) to meet the communication requirements at the different stages of the execution.

- **DIOXP2 card:**

A transputer based add-in board was designed to work as an accelerator to the HP 9000-300 series of engineering workstations to increase the capabilities of three dimensional modeling of solid objects (computationally intensive task). Figure (1-6) shows the DIOXP2 card and its relation with the workstation [24].

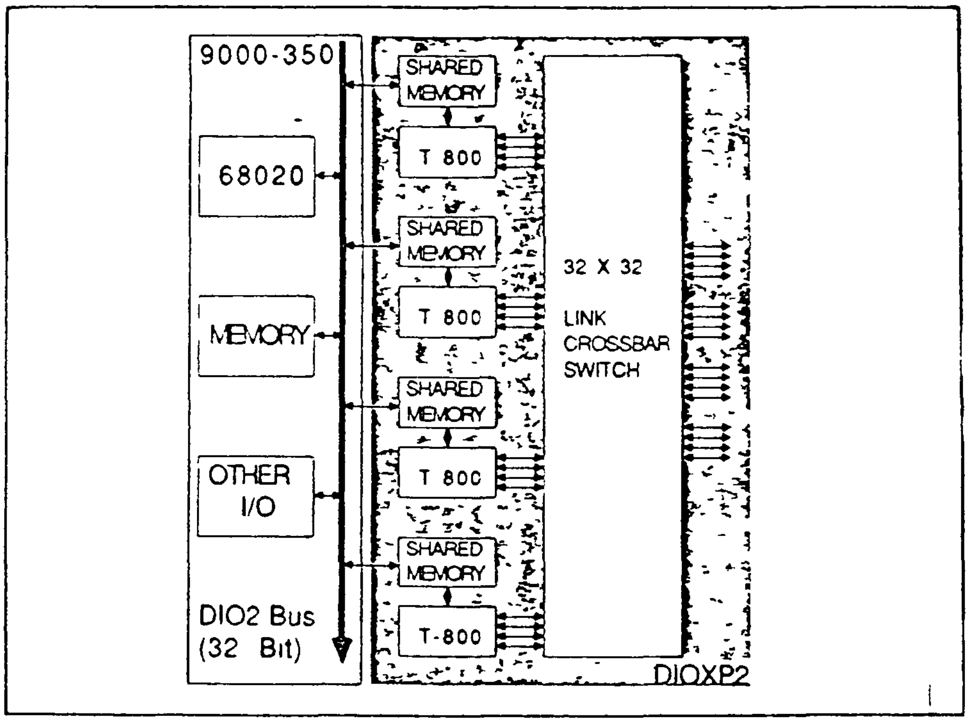


Figure (1-6) [24]

Algorithms being used for the modeling require different operations to be done on identical copies of the object data which is primarily expressed as double precision floating point numbers. To decrease the time consumed in sending the data to the processors the design provides a data broadcasting facility which is able to send the data to some or all the transputers simultaneously. The broadcasting of the data is achieved by using dual ported memory for each transputer and the host has access to all these memories and can only write to all or some of them at the same address. Communications between the processes running on the transputers are done via the transputers' links which are connected to a crossbar switch controlled by the workstation through special circuity.

Messages could be passed between the transputers using packet switching as well as circuit switching because the system allows run-time topology reconfiguration of the transputer network by enabling the user's application to access the crossbar switch control circuity.

1-5-4-3 Mixed topologies:

- The ParSiFal system:

The ParSiFal (Parallel Simulation Facility) has a modular architecture, a fully configured system contains to 1024 transputers each of it with 1M or 2M Bytes of external RAM. The transputers are installed on boards each contains four IMS T800.

Sixteen of these boards comprise one module called the T-Rack machine which was designed as a step in the ParSiFal project [27], Figure (1-7). The T-Rack machine has 64 transputers connected together by links 0 and 1 to form a processor pipe, Link0 of the first transputer is connected to Link3 of the Root transputer (IMS T414) on the interface board

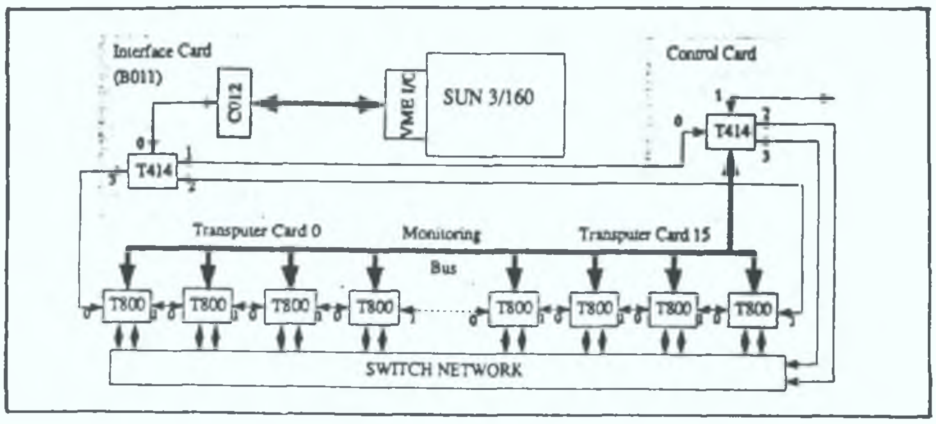


Figure (1-7) [27]

with the host computer (Sun 3/160) and Link1 of the last (which could be connected to Link0 from another T-Rack) is connected to Link2 of the Root transputer. The rest of the links (2,3) are taken to a crossbar switch controlled by a separate board called the Control board. The user program can control the crossbar switch via a Link connects the interface transputer to the control transputer. The T-Rack has a Byte-wide monitoring bus appearing as a register in the address space of each transputer and as an array of registers in the address space of the switch control transputer, this bus provides the mean of communication between the application transputers and the crossbar control transputer (the dynamic topology control transputer). During the Run-Time a process running on a transputer may need to communicate with another process running on a non-neighbour transputer. A message is sent to the control transputer, through the monitoring bus, asking for the required connection [28]. The system is used for parallel architectures simulation.

- **IMS B008 module motherboard:**

The IMS B008 is an IBM-PC plug-in Transputer Module motherboard, Figure (1-8). It provides the interface between the PC and the transputer based system. The board can accommodate **10** transputer modules (TRAMs). A processor pipe is formed by connecting Links 1 and 2 of each TRAM. The rest of the links (0,3) are taken to a software controlled 32X32 crossbar switch, the IMS C004 [22], to allow a variety of topologies to be implemented. The board is cascadable to allow a multi-motherboard system to be built, Figure (1-9).

The structure of the IMS B008 is similar to the structure of the T-Rack with a major difference which is the absence of the monitoring bus.

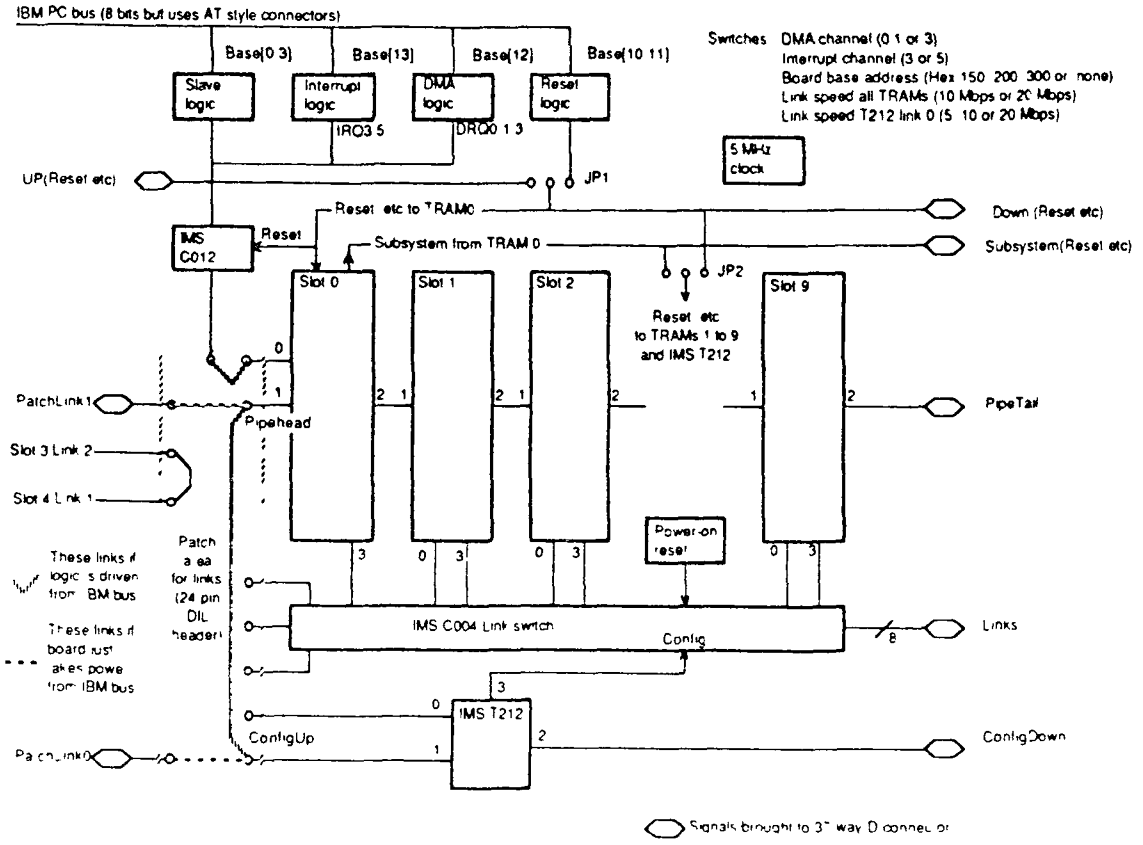

IBM PC bus (8 bits but uses AT style connectors)
Base[0 3]
Base[13]
Base[12]
Base[10 11]
Slave logic
Interrupt logic
DMA logic
Reset logic
IRQ3 5
DRQ0 1 3
Switches DMA channel (0 1 or 3)
Interrupt channel (3 or 5)
Board base address (Hex 150 200 300 or none)
Link speed all TRAMs (10 Mbps or 20 Mbps)
Link speed T212 link 0 (5 10 or 20 Mbps)
5 MHz clock
UP(Reset etc)
JP1
Reset etc to TRAM0
Down (Reset etc)
IMS C012
Reset
Subsystem from TRAM 0
Subsystem(Reset etc)
JP2
Slot 0
Slot 1
Slot 2
Slot 9
Reset etc to TRAMs 1 to 9 and IMS T212
PatchLink1
Pipehead
PipeTail
Slot 3 Link 2
Slot 4 Link 1
These links if logic is driven from IBM bus
These links if board just takes power from IBM bus
Patch area for links (24 pin DIL header)
Power-on reset
IMS C004 Link switch
Config
Links
IMS T212
ConfigUp
ConfigDown
PatchLink0
Signals brought to 37 way D connector

Figure (1-8) [29]

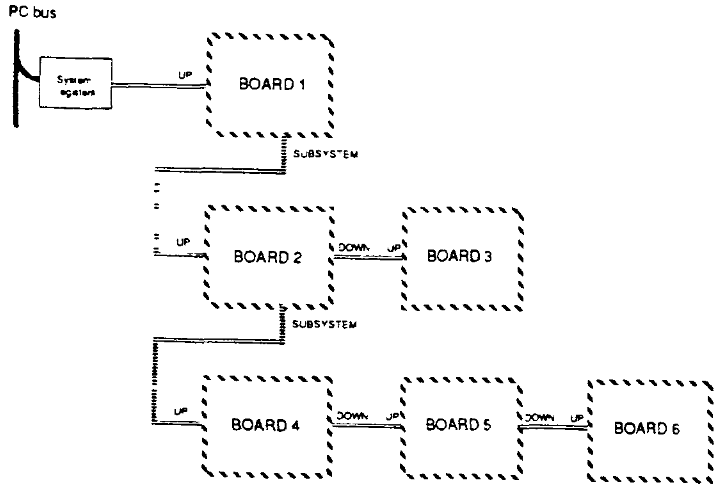


Figure (1-9) [22]

For more details please refer to [29],[36].

The transputer efficiency and the simplicity of implementing transputer based architectures give transputer systems an advantage on other processors based parallel architectures.

As we have seen static topologies are special purpose and to get high performances on such machines, restrictions should be imposed on the programs structures to achieve a kind of match between the program and the machine. Dynamic topologies flexibility allow a wide range of topologies, message passing techniques to be implemented (chapter 4) and that makes it more general purpose.

In the next chapter the procedure of configuring parallel programs to be fitted onto the parallel processor network is described, then the configuring procedure of the TDS (Transputer Development System) to run a program on a transputer network is discused in detail.

Chapter 2

Configuring Parallel Programs

2-1 Introduction:

The design of parallel programs passes through several stages before it could be compiled and run. Configuration is last where the processes are located at the processors and the communication links are located at the interprocessor connections.

2-2 Parallel program configuring stages:

Figure (2-1) represents the configuring procedure.

2-2-1 Parallelism spotting (Partitioning):

This stage is extremely important because the next stages are strongly depend on its results.

In fact most of the computing applications (e.g. real-time systems) are inherently parallel in nature. An application behavior could be described by a collection of concurrent processes, each describing the behavior of a particular aspect of the implementation. Running these processes in parallel and enabling them to communicate together will give an actual representation and better response to the events in real-time. A match between the program structure and the target parallel processor network architecture should be made to achieve high performances.

2-2-3 Processes assigning:

Each concurrent processes is assigned to an individual processor if possible. Often there will be

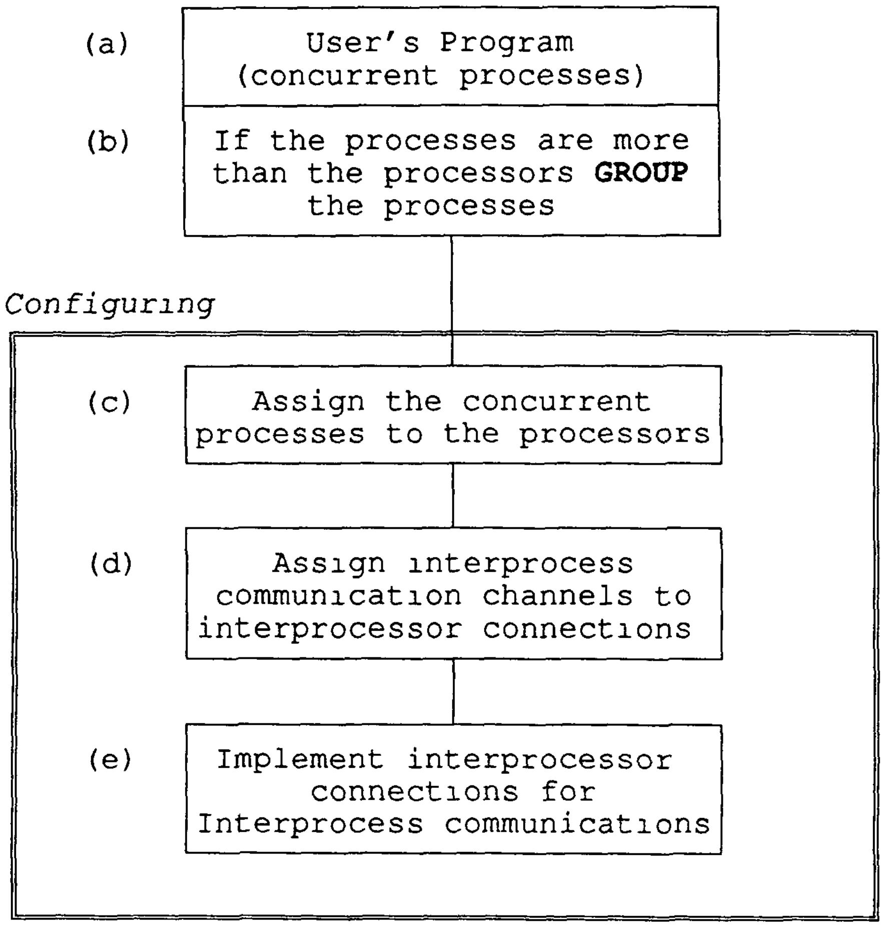

(a)
User's Program (concurrent processes)
(b)
If the processes are more than the processors GROUP the processes
Configuring
(c)
Assign the concurrent processes to the processors
(d)
Assign interprocess communication channels to interprocessor connections
(e)
Implement interprocessor connections for Interprocess communications

Figure (2-1)

more processes than the available processors. In such cases groups of processes are assigned to individual processors. Processes are grouped according to their expected computation time, and communication relations.

The balance of the computational load between the processors increases the system performance, but it is not the only factor. In some applications where the communication traffic is heavy, the communication system efficiency determine the overall system performance.

Computational load balance, and short paths between heavily communicating processes are major factors in process distributing [21].

2-2-4 Communication channels assigning:

After the processes are assigned the communication channels between the processes could be assigned to the interprocessor connections depending on the processes places and the routing algorithm.

2-2-5 Interprocessor connection implementation:

If the system has a dynamic or mixed (dynamic and static) topology, the processor network is configured to meet the previous stages requirements.

2-3 Configuring parallel programs under the TDS:

2-3-1 TDS2 The transputer development system:

The TDS2 is an integrated development system designed to support the development of Occam programs for transputer networks applications. The TDS2 consists of two parts; one contains the editor, the compiler and other utilities. This part runs on a transputer based plug-in board like the IMS B004 or IMS B008. The other part provides the access to the host filling system and the console (screen, keyboard). This part runs on the host system.

Occam programs can be edited, compiled, and run from within the TDS2. Programs to run on a transputer network could be developed as well. The TDS2 has many other utilities like parallel programs debugger, transputer networks tester, mathematical library, Input/Output functions library, and others [34].

2-3-2 TDS configuring procedure:

In order to configure a program constructed from a number of parallel processes under the TDS, a number of steps should be done. These steps are [34]:

1: Each process to be allocated to a transputer should be contained within a fold of type SC.

2: All the SC folds should be collected into a fold of type PROGRAM.

3: Configuration statements are added to the PROGRAM fold, these statements place the processes each at a transputer and describe the interconnections between the processes by mapping the external channels onto the links of the transputers using special constructs.

4: Compile the program and then extract all the code produced by the compiler into on code fold.

5: Before the network could be loaded with the executable code of the configured program the connections between the transputers should be implemented (Dynamic topology) using the proper hardware or software (if the network can be configured by software).

The programmer needs to know some details about the target network to be able to configure a program and to implement the required topology.

In our case where the target hardware is the IMS B008

[29], configuring a program passes through the same steps using the MMS2 [35] for step 5.

2-3-3 MMS2 The module motherboard software:

This package is supplied by INMOS to enable users of the IMS B008 module motherboard to configure the transputer network to meet their programs communication requirements. The MMS2 requires two files as input files, one to describe the new connections the user wants to create, and one to describe the hardwired connections of the board, this file is supplied by INMOS [29]. To write the softwired connections description file, the programmer has to know at which slots, on the board, the TRAMs (Transputer Module) are installed.

The MMS2 runs independently from the TDS2 and from outside the TDS2. If the TDS is suspended and the MMS2 is run, the system can't return back to the TDS. That means the network should be configured before running the TDS and then run the TDS and load the network with the code of a configured program. The whole process must be repeated to run another program which requires a different topology.

The MMS2 is able to create bootable files to configure the network without the need to run the MMS2 every time the network needs to be configured [35]. This file should be used to configure the network before running the TDS.

2-3-3-1 Using the MMS2:

The following steps describes the use of the MMS2 to configure a transputer network on the IMS B008:

1: Draw the topology and name (number) the links.

2: Write the softwired connections description file.

3: Run the MMS2 using his file as the first input file and the hardwired connections description file as the second input.

4: If the topology is going to be frequently used, it's better to produce a bootable file. Reset the subsystem, and initialize the C004s (the crossbar switch), before setting the C004.

5: The TDS could be run now and the network could be loaded with the program code.

2-3-4 Case study (Configuring an example program):

Suppose a program of five parallel processes, is be configured to run on a network of five transputers, (stages a & b, Figure (2-1), are already done).

The processes external channels are as in Table 2-1, and the required network topology to run the program is illustrated in Figure (2-2). The first step is to place (assign) the concurrent processes to the available processors (transputers), Figure (2-1,c). Under the TDS special configuring instructions are added to the program source to place the processes at the transputers. Numbers are given to the transputers, but they have no relation with the actual numbers of the TRAMs they represent.

Process name	Input channels name	Output channels name
process.0	from.1.to.0	from.0.to.1

process.1	from.0.to.1
	from.2.to.1
	from.3.to.1
	from.4.to.1
		from.1.to.0
		from.1.to.2
		from.1.to.3
		from.1.to.4

process.2	from.1.to.2
	from.3.to.2
	from.4.to.2
		from.2.to.1
		from.2.to.3
		from.2.to.4

process.3	from.1.to.3
	from.2.to.3
	from.4.to.3
		from.3.to.1
		from.3.to.2
		from.3.to.4

process.4	from.1.to.4
	from.2.to.4
	from.3.to.4
		from.4.to.1
		from.4.to.2
		from.4.to.3

Table (2-1)

The external channels are placed at the transputers' links using a supposed placing of the processes at the SLOTs (the process placed at PROCESSOR 0 in the previous step is placed at the first occupied SLOT after SLOT 0).

After the external channels are placed the softwired links description file is written to be used by the MMS2 to configure the network. The following is the contents of the softwired file:

```
    SOFTWIRE
      SLOT 1, LINK 3 TO SLOT 6, LINK 0
      SLOT 1, LINK 3 TO SLOT 5, LINK 0
      SLOT 5, LINK 3 TO SLOT 6, LINK 3
    END
```

-Softwire description file for Figure (2-2) topology

Now the program is configured and could be compiled and extracted. The network should be configured before loading it with program executable code.

Appendix A contains the configured program listing.

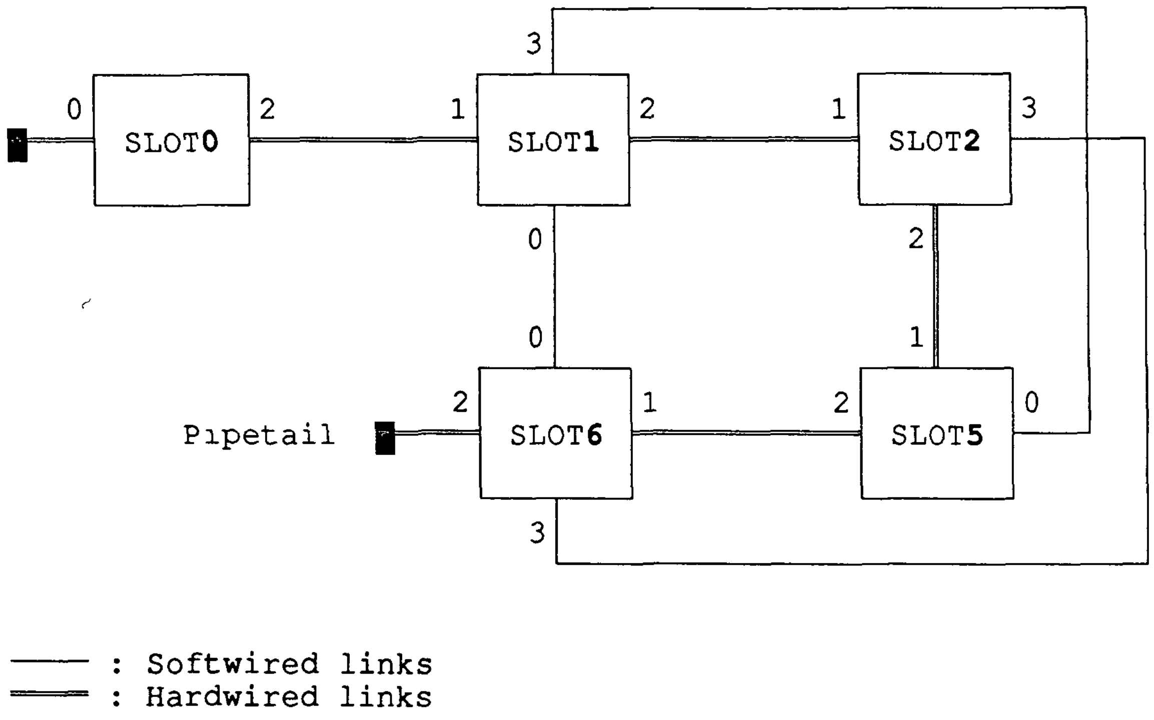


Figure (2-2)

For a network of small number of transputers this procedure could done by the programmer, but for larger networks it will become complicated and will consume a long time.

These difficulties are not faced when working on a sequential machine or developing a sequential program which we are accustomed to. I believe that the job of parallel machines users should end after writing their programs, which are parallel processes communicate with each other. The lesser the restrictions put on programs writing, the closer to a user friendly easy to use parallel machine we are.

Why not have the configuration done by the computer? The next chapter describes a configurer for programs developed under the TDS and will run on the IMS B008.

A Configurer for Programs Running Under the TDS2 on the IMS B008

3-1 Introduction:

Attempts to design a transparent parallel machine and to make the already available ones transparent to the programmer are on going, and the need for that is growing as the number of such machines' users is becoming bigger and bigger.

The IMS B008 is made transparent to the users by the configurer described below. All hardware related work is carried out by the configurer.

3-2 The environment:

The program was designed and tested on the IMS B008 under the TDS2 operating system, hosted by an IBM PC-AT.

3-3 Implementing message passing techniques on the IMS B008:

3-3-1 Through routing:

This technique is applied to a static topology network. A communication handling process (router) should be placed at each transputer either by adding the process to the program before the compilation or by designing a general routing process. This process is placed at each transputer. Before loading the network with any program code, the routing processes should be fed with the processes map and their links, this maybe done by another process running on the host. After that the network could be loaded with the executable code of the program.

The "neighbours identification" operation should be done at first by all the routers and each routing process should have complete information about which process is running on which transputer. Packet switching is used so each message should contain information about it's length, source identity, and destination. The router will accept any message on any link and then the message will be redirected to another transputer via the proper link according to the process map of the network.

3-3-2 Channel multiplexing:

Similar to the Through-routing from the implementation point of view except that communication handling processes multiplex and demultiplex channels on the appropriate link. In this technique each process should know its neighbours only, unless both techniques, Through-routing and Channel multiplexing, are used together where it becomes necessary for the communication processes to know the process map of the whole network.

3-3-3 Dynamic links reconfiguration:

The IMS B008 is very similar to the T-Rack machine but on a small scale and with a very important difference which is the absence of the monitoring bus.

The monitoring bus plays the major role in the run-time link-switching in the T-Rack machine where it carries the processes requests for links switching to the controller where it is dealt with. Something else should play the role of the monitoring bus in the T-Rack machine on the IMS B008. In the IMS B008 two links from each TRAM are connected to software controlled crossbar switch, and two are used to form the processor pipeline. To configure the network, the configuration data are sent on the configure input of the crossbar switch (IMS C004) via Link3 of the IMS T212 (refer to [29] for more details).

Each TRAM has two links connected to the crossbar switch (Link0 and Link3) except TRAM0 where Link3 only is connected to the crossbar switch. By sacrificing one of these links of each TRAM, say Link0 and lets call it "configure link", this link will take the place of the monitoring bus of the T-Rack machine. The other link, Link3, of each TRAM is called "flexible link".

A process is allocated at each TRAM and called the "Organizer", its job will be described later, and another process running on the IMS T212 is also needed. This process acts like a telephone exchange operator. The process is called the "Controller".

Any process, no matter on which TRAM it is running, wanting to communicate with any process which is not running on a neighbour TRAM (processes running on neighbour TRAMs are accessible via Link1 and Link2), will send a message to the organizer of it's TRAM. The message contains information about the channel the process wants to communicate on and the direction (send or receive). In the case of receiving, the organizer keeps the channel number (channels are numbered to ease the searching task) and the process is then blocked on a message from the organizer to resume. In the case of sending, the organizer sends a message to the controller (on T212) via the configure link requesting a linkage with the TRAM on which there is a process waiting to receive a message on the same channel.

The controller scans all the configure links continuously, and on receiving a message from an organizer it sends searching messages to all the organizers looking for the process waiting to receive on the wanted channel (blocked on a message from the local organizer). If the process is found, the organizer on the TRAM on which the process waiting to receive was found, sends a message to the controller. Then the controller connects the flexible links of both TRAMs (if not already in use) and inform both organizers. Both organizers will then send a resume message to both processes and the communication take place in the conventional way. After the message is sent the sending process should inform the organizer that it finished and the organizer in it's turn informs the controller which disconnects the flexible links. The organizer process is identical on all TRAMs.

This design was tried on a simple example which contains four processes running on four TRAMs excluding the Host.

As we saw, this method requires some modifications on the processes to insert the handshaking between the processes and the organizers. Another problem is caused by the size of the controller process which should fit in the internal memory of the T212 because it has no external memory on the IMS B008. For these reasons and specially the first one this technique, dynamic link reconfiguring, was not used in the configurer.

The choice of which technique to use is left to the user. The configurer adds no new restrictions on programs to be configured (other) than the TDS restrictions.

3-4 The configurer:

3-4-1 What is requisted from the configurer?

The configurer should be able to do the following:

1: Discover the target transputer network.

2: Discover the structure of the program (parallel processes).

3: Map and place the processes at the transputers and then place the channels at the links.

4: Configure the network (set the crossbar switch) to meet the required topology.

5: Produce the configured output as an acceptable input to the TDS compiler.

This version of the configurer is able to configure a program constructed from a number of processes **less than or equal** to the number of the available TRAMs on one IMS B008. Each process could have a number of external channels

less than or equal to the number of the available links of the TRAM of both input and output. The configurer does not support the replicated PAR instruction in this version.

The program deals with one IMS B008, but with some modifications on some parts of it, the program would be able to deal with a number of cascaded IMS B008 boards.

3-4-2 The configurer general description:

The program consists of two programs running in parallel. The first runs on TRAM0, and accesses the file system of the host computer (PC) and does the most of the job. The second runs on the IMS T212 (T2) controls the crossbar switch according to messages sent by the first, Figure (3-1) shows the major flowchart of the configurer.

The program starts with discovering the network to define the number of the installed TRAMs and the slots they occupied using the crossbar switch.

After that the configurer searches for the source of the input program (the program to be configured). If the source was found, the program reads the information required in the configuration process. This information in addition to the network information is then used to configure the program.

The configurer has three phases. In the first phase the configuring data is checked. If the data is valid, the next phase is performed through which the processes and their external channels are mapped on the network.

The network is then configured if the mapping was

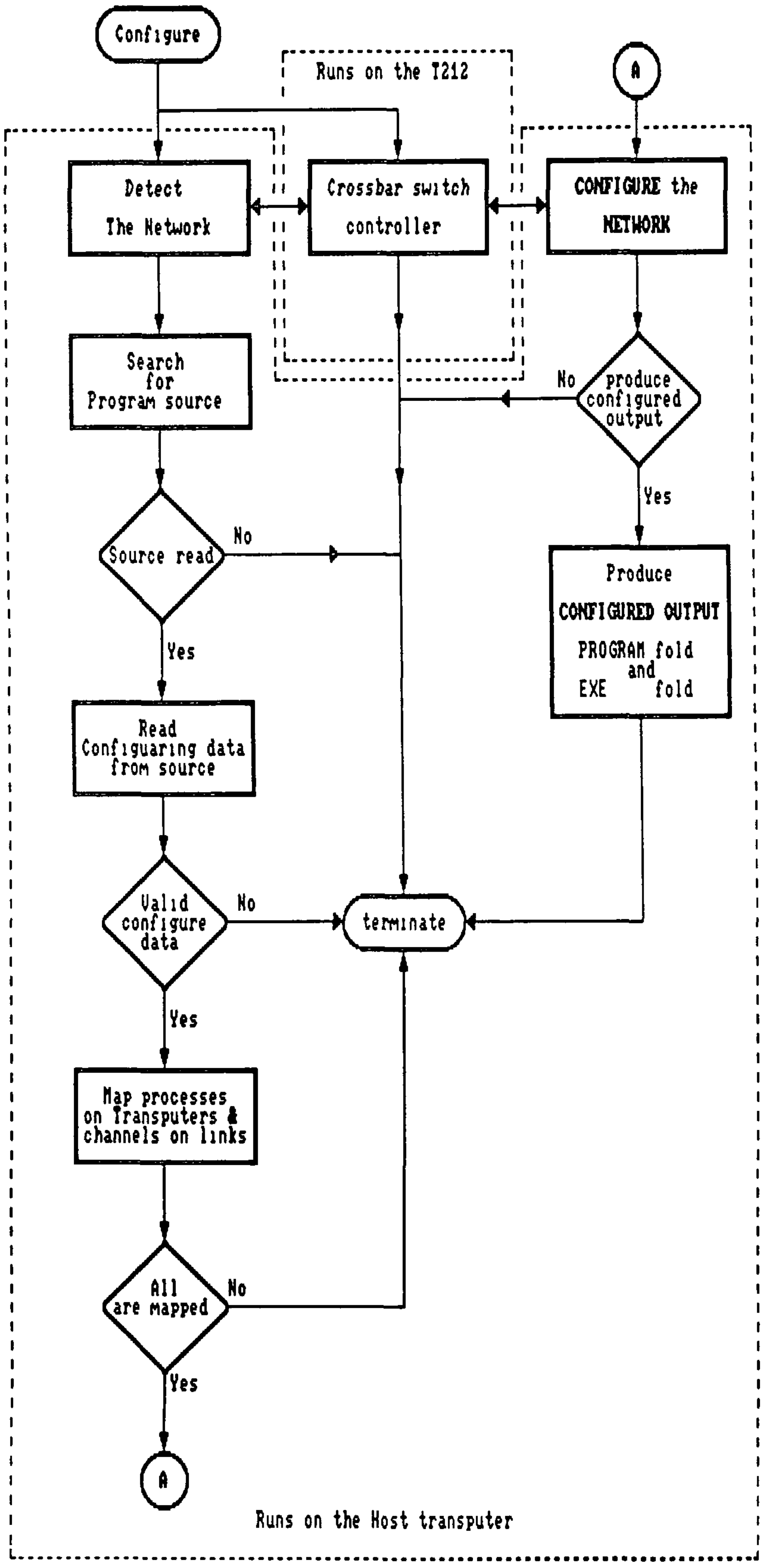

Configure
Runs on the T212
A
Detect The Network
Crossbar switch controller
CONFIGURE the NETWORK
Search for Program source
No
produce configured output
Yes
Source read
No
Yes
Produce CONFIGURED OUTPUT PROGRAM fold and EXE fold
Read Configuaring data from source
Valid configure data
No
terminate
Yes
Map processes on Transputers & channels on links
All are mapped
No
Yes
A
Runs on the Host transputer

Figure (3-1), Configurer general flowchart successful. At this point the configurer program could be stopped if the configured output is not required (the configurer could be used to configure the network to run a pre-configured program). The configurer outputs have all the required configuring instructions added.

3-4-3 The general algorithm:

The general algorithm is actually the summary of the previous paragraph.

```
Begin;
  Detect the available TRAMs in the network.
  Search the program source for processes.
  FOR each found process
    DO:
      Get the process name.
      Read the external channels information.
      Check the channels declarations.
      Check the number of the channels.
      Check input and output channels matching.
  Place the processes at the TRAMs.
  Place the channels at the links.
  Configure the network.
  If required
    Produce the configured output.
End;
```

3-4-4 The configurer requirements in the input program:

The input program should be a foldset of type PROGRAM and fulfil the following points:

1: A fold literally called **"EXTERNAL CHANNELS PROTOCOLS"** which contains the declarations of all protocols of the external channels.

2: Each process should be contained within an SC type foldset including the process which will run on the host.

3: Each process should contain a fold of type COMMENT and literally called **"PROCESS <the process name> EXTERNAL CHANNELS"** contains the declarations of the external channels of the process. This fold should preced the outer PROC statement of the process.

4: External channels should be declared, in the previous fold, using the same format of Occam, but with an addition which is the direction of the channel (input or output) using the format:

CHAN OF <protocol> <channel name,,,> <?/!>:

"?" indicates input channels and "!" indicates output channels.

5: A fold literally called **"CONFIGURE"** contains a PLACED PAR instruction and the process's calling statements starting with the call statement of the host process (the process which has access to the system). The indentation of all the above folds should be zero relative to the PROGRAM fold (please refer to [27] for details about the folds types and indentation).

3-4-5 Special cases:

Sometimes the conditions of the configurer are fulfilled by the program but the configuration failed. If that happened it will be caused by the IMS B008 design as will be discussed.

Because of the hardwired links on the board some difficulties may be encountered while configuring a program and sometimes the required topology could not be implemented on the IMS B008.

3-4-5-1 Three non-neighbouring processes:

Figure (3-2-a) shows the required topology for a program, but because two links of each TRAM are hardwired the topology could not be implemented using this distribution of processes.

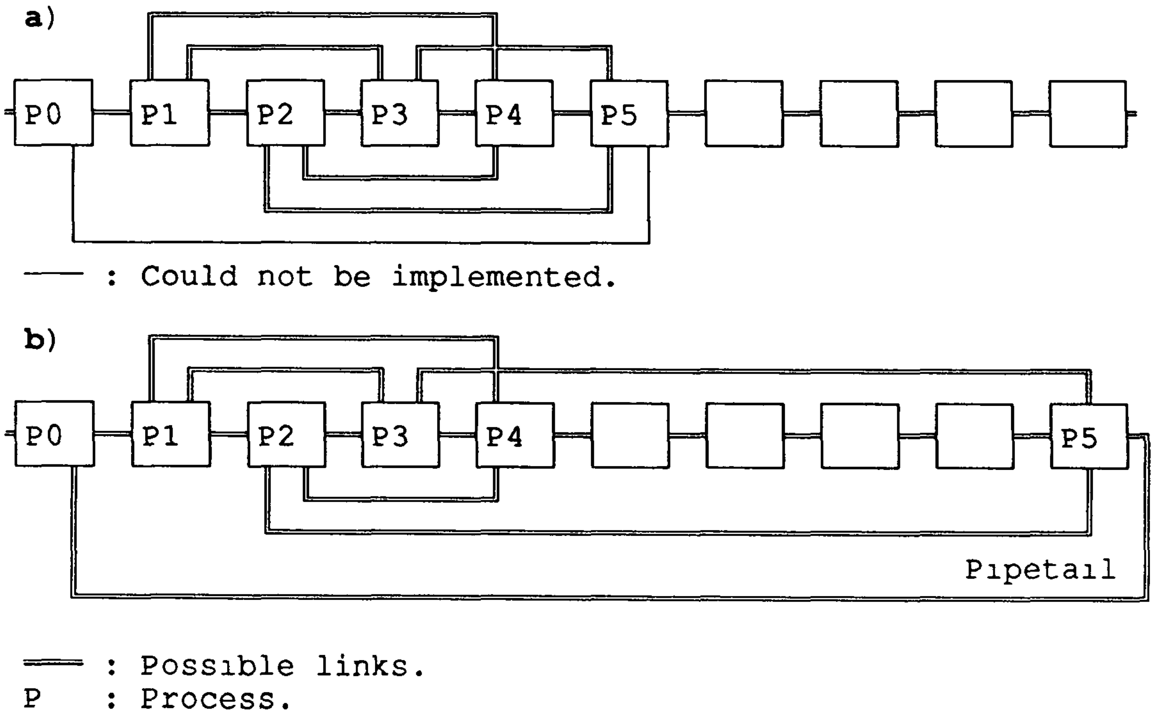


Figure (3-2)

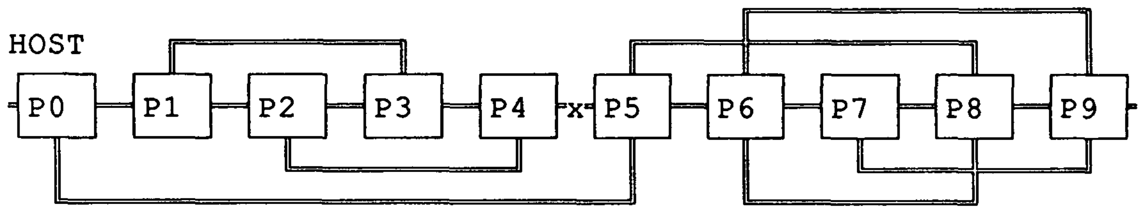
In Figure (3-2-b) P5 has been shifted to the end of the pipe to make use of the pipetail link through the edge connector. This solution is avoided by the program because of the complexity of it's implementation if cascaded boards are used. The program makes use of the pipetail link in the case where the number of the processes is equal to the number of the TRAMs number.

3-4-5-2 All the processes need four links:

In the case where all the processes need four links and the number of the processes is **less** than the number of the TRAMs the topology is impossible on the IMS B008. A channel at least should be cancelled to make the topology implementable.

3-4-5-3 No loading route from the root TRAM:

The loader sends all the code to the root TRAM to be distributed from there. In some cases like the one illustrated in Figure (3-3) the network could not be loaded and a loader failure message will be sent.



x : No channels are placed at these links.

Figure (3-3)

In such cases the configurer adds a channel and places it at the proper links to connect the loading route, this channel is called "JUST.FOR.LOADING". That could happen once in the topology because the host TRAM has two links only.

3-4-6 The configurer program:

The configurer design tries to minimize the work the the programmer has to do to configure a program. All the programmer has to do before configuring a program (apart from writing the processes) is to add the external channels declaration folds to each process in the way described above, and after the configuration is done he (she) has to make the output folds as foldsets, all the other requirements are TDS requirements.

3-4-6-1 Network detection:

This task is carried out by the two parallel parts of the configurer.

3-4-6-1-1 The program on T2 (controller):

The program runs on the T2 controls the crossbar switch which is connected to the T2 via Link3 refer to Figure (2-1).

The controller receives messages from the program running on the host via Link1 and sends control data (configure data) to the crossbar switch via Link3.

The controller could be easily modified to act as a worm program for applications where cascaded IMS B008s are used. Figure (3-4) is the general flowchart of the controller program.

At the beginning, the program on the host resets

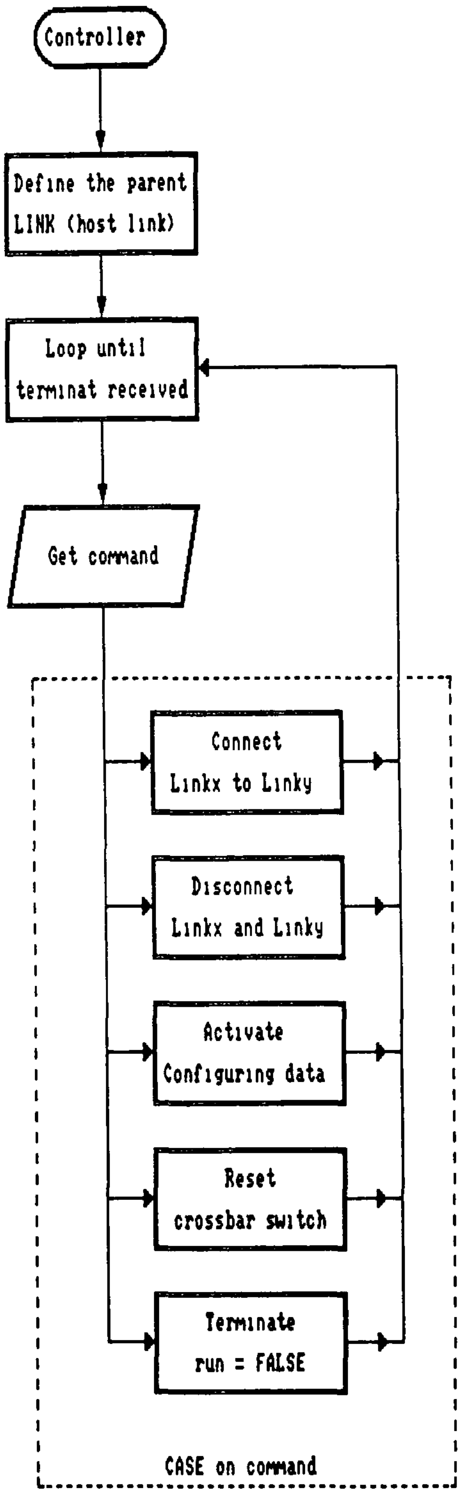

Controller
Define the parent LINK (host link)
Loop until terminat received
Get command
Connect Linkx to Linky
Disconnect Linkx and Linky
Activate Configuring data
Reset crossbar switch
Terminate run = FALSE
CASE on command

Figure (3-4), Crossbar switch controller flowchart the subsystem which includes T2 and the crossbar switch, then it boots T2 with the controller code.

When the controller starts to run it waits for a number to be sent on any link, this link will be defined as the host link. After that the controller enters a loop which is the body of the program. This loop starts with receiving a command from the host and then a CASE construct on the command is executed.

No processing is done by the controller to keep it's size as small as possible to fit in the internal memory of T2. The program is needed twice for the configuration, firstly to discover the network, and secondly to set the crossbar switch (configure the network).

The program supports all the IMS C004 commands except the enquiry command, but only four commands are actually used:

1: Connect Link<x> to Link<y>.

2: Disconnect Link<x> and Link<y>.

3: Activate the configuring data.

4: Reset, all the outputs are disconnected.

x,y are IMS C004 links numbers [14].

When the configuration is done, the host sends a terminating command to terminate the controller.

3-4-6-1-2 Network detecting procedure (Host):

The configurer starts by calling this procedure "network" to define the number of the available TRAMs

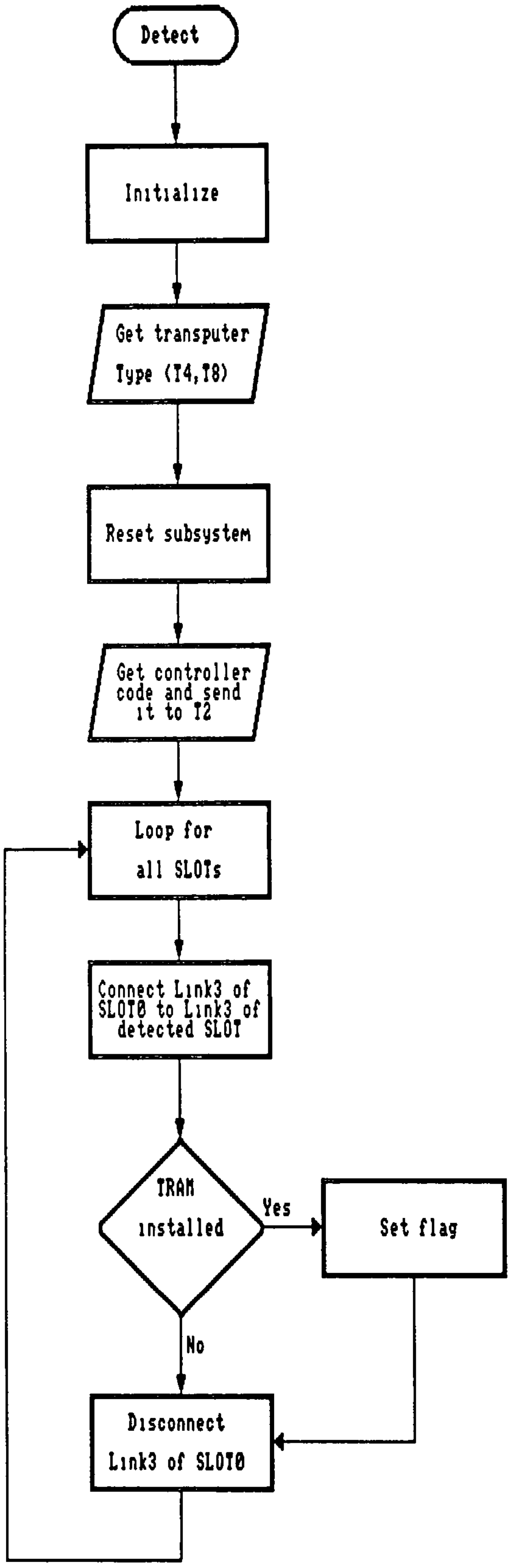

Detect
Initialize
Get transputer Type (T4,T8)
Reset subsystem
Get controller code and send it to T2
Loop for all SLOTs
Connect Link3 of SLOT0 to Link3 of detected SLOT
TRAM installed
Yes
Set flag
No
Disconnect Link3 of SLOT0

Figure (3-5), Network detector flowchart in the network and the slots they are occupying, Figure (3-5). The number of the TRAMs is kept in the variable "nTransputers" and the occupied slots are kept in the Boolein array "slots".

The procedure starts with initializations and then gets the transputer type from the user (T4 or T8). After that the subsystem (network) is reset and the code of the controller is got from a file called "C004byT2.cpr" by the local procedure "read.T2.code". Then T2 is booted with the controller code. If everything goes well, the length of the controller is sent to T2 to enable the controller to identify the host link on which the communication will take place. A trial command is then sent to the controller to assure that every thing there is OK, the local procedure "is.loaded" is used for that. Then the network discovering process starts with resetting the crossbar switch to cancel any previous connections.

The slots are then scanned by connecting Link3 of the host (slot0) to Link3 of each slot in turn, by sending the appropriate commands and data to the controller. A poke operation trying to write anything in the internal memory of the transputer at a certain address is then performed followed by a peek operation to read the contents of the same address. If the read value is the same written value a TRAM is known to be installed at that slot, "nTransputers" is incremented and the corresponding item of "slots" is set to TRUE. After that the links are disconnected (Link3 of slot0 and Link3 of the tested slot) and a connection is done with Link3 of the next slot and so on until the whole slots are tested. Then the number of TRAMs found is displayed on the screen and the network discovering phase is finished.

3-4-6-2 Searching for the program source fold:

The program looks for the source in the first fold of the bundle which should be a foldset of type PROGRAM. Procedure "search.for.source" is used for that.

At first the procedure gets the number of the folds in the bundle to check if there is any or not. If the number is one or more the attributes of the fold are read and checked where the acceptable attributes are:

TYPE	:	foldset or voidset.
CONTENT	:	fc.occam2.prg.

If the attributes are accepted then the attributes of the first fold are read to check it's contents which should be "fc.source.text". If it is, then "prog.source.found" is set to TRUE and the search process succeeded, otherwise the procedure report on the screen with an error message.

3-4-6-3 Reading the program source:

The program requires special information folds to

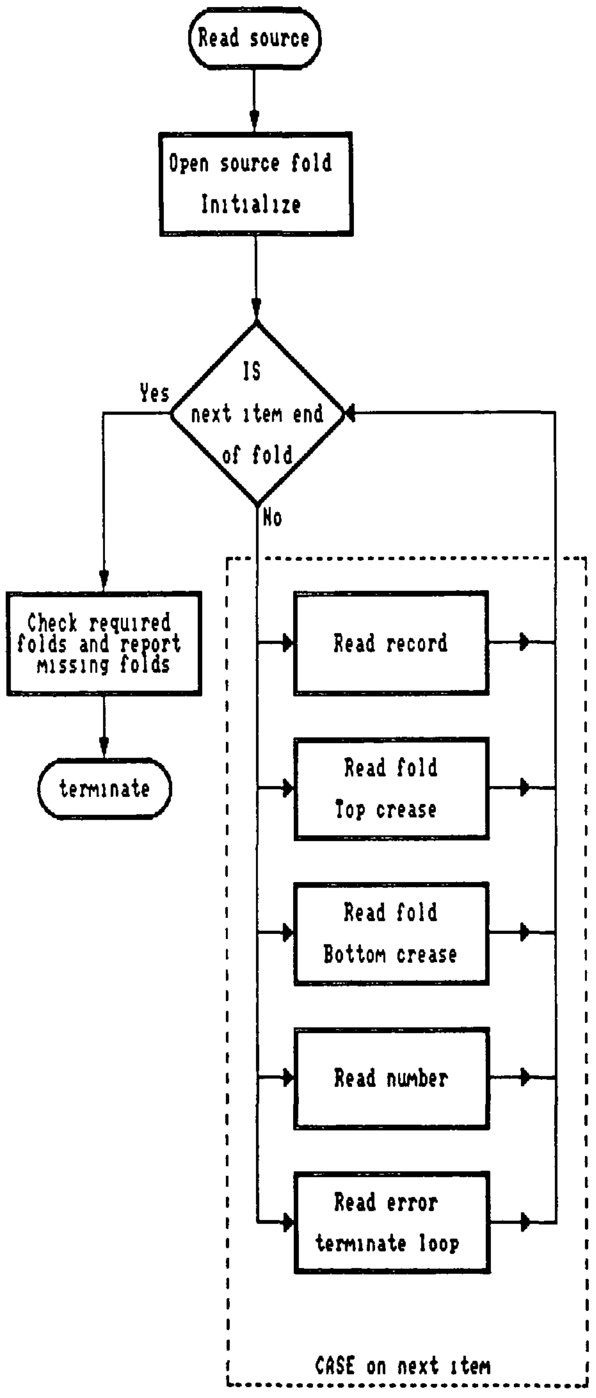


Figure (3-6), Input program source read

be added by the programmer to the program in order to decrease the needed amount of processing. The program only reads the folds which contain the information necessary for configuring, which is found in the added external channels folds and the configure fold. Missed folds, if any, are reported and the programs stops.

Procedure "source.read" is used to read the required information from the source of the program to be configured, Figure (3-6).

The procedure tries at first to open the first fold for reading, if it failed an error message is sent to screen and the program stops. If it passed, some initializations are done and the program enters the reading loop contains a CASE construct which forms the core of the procedure. The reading loop ends on reaching the end of the opened fold, or an error occurred during the reading. Using some library procedures for folds reading, the program source is read. Special processing is done when a "top crease" or a "bottom crease" and if a record is the next item to be read.

On reaching a "top crease" the comment of the fold is read and checked to see whether this fold contains configuration data or not.

If the fold is an external channels fold, the process name of which the external channels fold was found is obtained from the comment using the local procedure "get.proc.name" which keeps the process name in the byte array "proc.names" and the length of the name in bytes in the integer array "proc.names.lens". The Boolean "comment.read" is set to TRUE to indicate to "read.record" procedure to put the contents of the current fold into the buffer. The current value of the buffer pointer is saved in the integer array "ext.chn.ptr" to indicate the offset of the start of the external channels information of the current process in the buffer.

If the fold is the configure fold (the fold contains the processes calling statements), the Boolean "configure.read" is set to TRUE to indicate to "read.record" procedure to put the contents of the current fold into the buffer. The current value of the buffer pointer is saved in the integer array "ext.chn.ptr" to indicate the offset of the start of the calling statements in the buffer.

If the fold is the external channels protocols declaring fold, the boolean "ext.protocol.read" is set to TRUE to indicate the availability of the fold.

On reaching a "bottom crease" a check is done to find out if the end of a configuring data fold is reached or not depending on the depth of the fold, which is increased on entering a fold and decreased on exiting. If an end is reached the appropriate boolean is set to FALSE and the current value of the buffer pointer is saved in the integer array "ext.chn.ptr" to indicate the offset of the end of the external channels information of the current process or the calling statements. If the end was of an external channels fold the processes counter "sc.number" is incremented.

The procedure "read.record" is used to read record items from the source, but it adds the record to the buffer only if "comment.read" or "configure.read" is TRUE and the record itself is not a comment record.

The procedure "source.read" will report any missed configuring fold. Any process does not have an external channels fold will not be regarded by the configurer.

3-4-6-4 Processes and channels mapping and placing:

The configurer divided this job into three tasks, each task is carried out by a special procedure called Phase.

3-4-6-4-1 Phase I:

After the available configuration data is read from the program source, the configurer enters a new phase which includes checking major points in the read data. These points are checked by the procedure "phase.I" which itself is consisted of a number of procedures each checks a different point, Figure (3-7).

The points checked during this phase are:

1: "PLACED PAR" instruction.

2: For each called process there is an

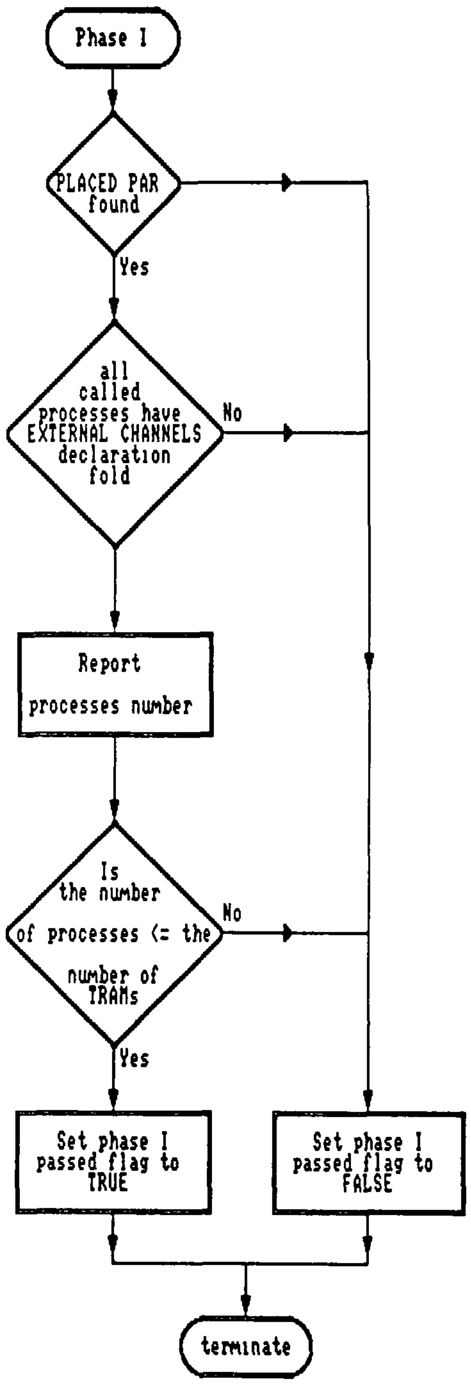

Phase I
PLACED PAR found
Yes
all called processes have EXTERNAL CHANNELS declaration fold
No
Report processes number
Is the number of processes <= the number of TRAMs
No
Yes
Set phase I passed flag to TRUE
Set phase I passed flag to FALSE
terminate

Figure (3-7)

external channels fold which holds the same process name.

3: The number of the called processes with the number of the available TRAMs.

The procedure starts with checking the number of the SC type folds found during the reading operation as there is no point to go on if there is nothing to place. After that the procedure "CONFIGURE.inst" is used to look for the "PLACED PAR" instruction in the data read from the fold called "CONFIGURE". If the instruction was found (step 1) the names of the called processes (to be placed) are obtained from the calling statements by the procedure "get.called.sc.name" and then compared to those previously got while reading the source by the procedure "match.sc.name". The names matching operation is done for each calling statement in turn. During the names matching the number of the called processes is updated every time a called process passed the check, this number is then compared with the number of the available TRAMs, and the order in which the processes are called is also saved. Error messages are issued on each error to help the user to identify the error. These error messages are as specific as possible. Phase one according to the configurer is now over and data required for phase two to be executed is available (but not checked).

3-4-6-4-2 Phase II:

This phase is the major phase in the program and most of the job is done here, Figure (3-8). At first the channels definition information of each process are got from the read data, then the channels map of the processes is produced, then the possibility of accessing all the processes from the host is checked. After that the processes map is produced to be used in the next step which is placing the processes followed by mapping the channels on the TRAMs links and the last step is configuring the network according to the results of the previous two steps.

Phase two starts with getting the channels' names, protocols, and directions for each process from the external channels data of that process. The input channels and the output channels are separated and counted for checking. A process could have up to four input and four output channels (except the host process can have two of each) and should have one of each at least (what is the point of a parallel process without input or output?). The procedure "channels.names" does all the checking on the channels in addition to some syntax checking on the channels declaration statements.

After that the procedure "io.or.i.or.o" is called to produce the channels map. The procedure tries to find the match input channel for each output channel,

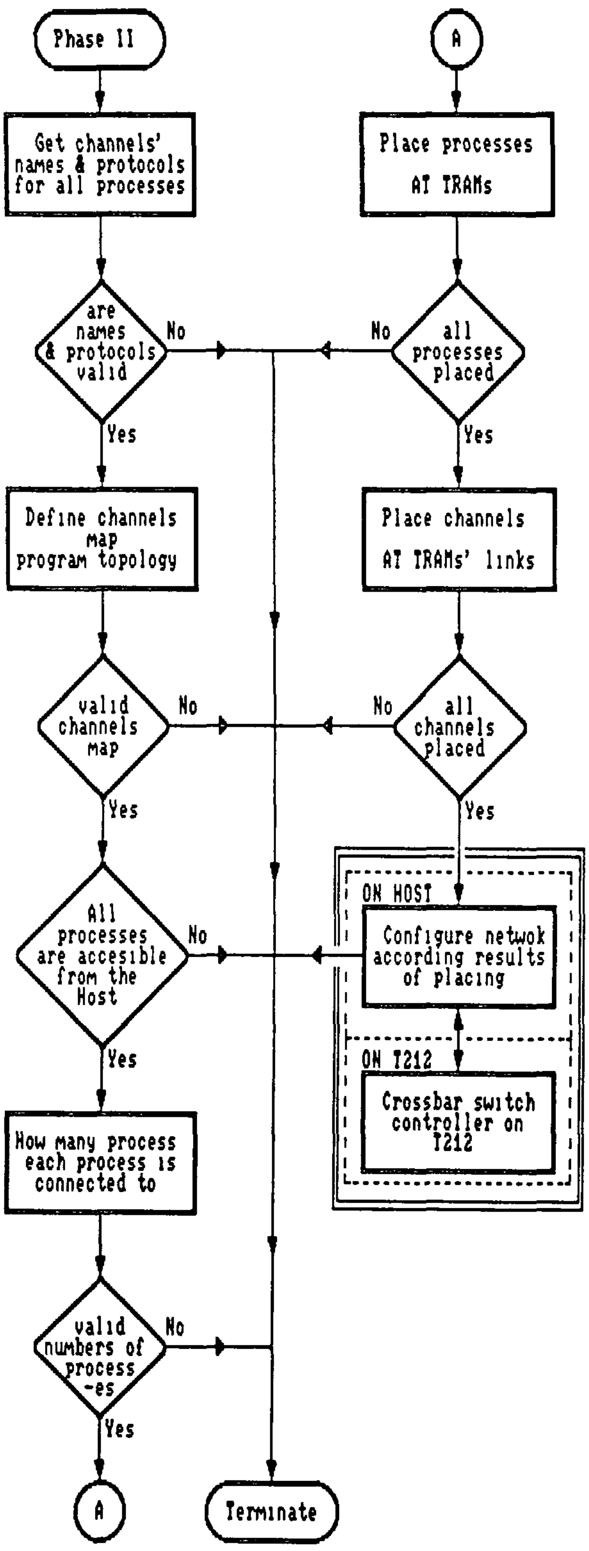

Phase II
Get channels' names & protocols for all processes
are names & protocols valid
No
Yes
Define channels map program topology
valid channels map
No
Yes
All processes are accesible from the Host
No
Yes
How many process each process is connected to
valid numbers of process -es
No
Yes
A
Terminate
A
Place processes AT TRAMs
all processes placed
No
Yes
Place channels AT TRAMs' links
all channels placed
No
Yes
ON HOST
Configure netwok according results of placing
ON T212
Crossbar switch controller on T212

Figure (3-8)

and checks that the channels are not already used to match different channels, either the input or the output channel. If the output channel is already used for another input channel, the procedure reports a duplicated input channel and a duplicated output channel.

The last check before the channel is mapped or put in the channels map is that the channel is not connected to the same process it belongs to (I could not find a reason to connect two links of the same TRAM, together).

After passing all the previous checks the channel is mapped and after the procedure "io.or.i.or.o" finishes, the channels map is ready unless there is a mistake of some kind in the external channels data.

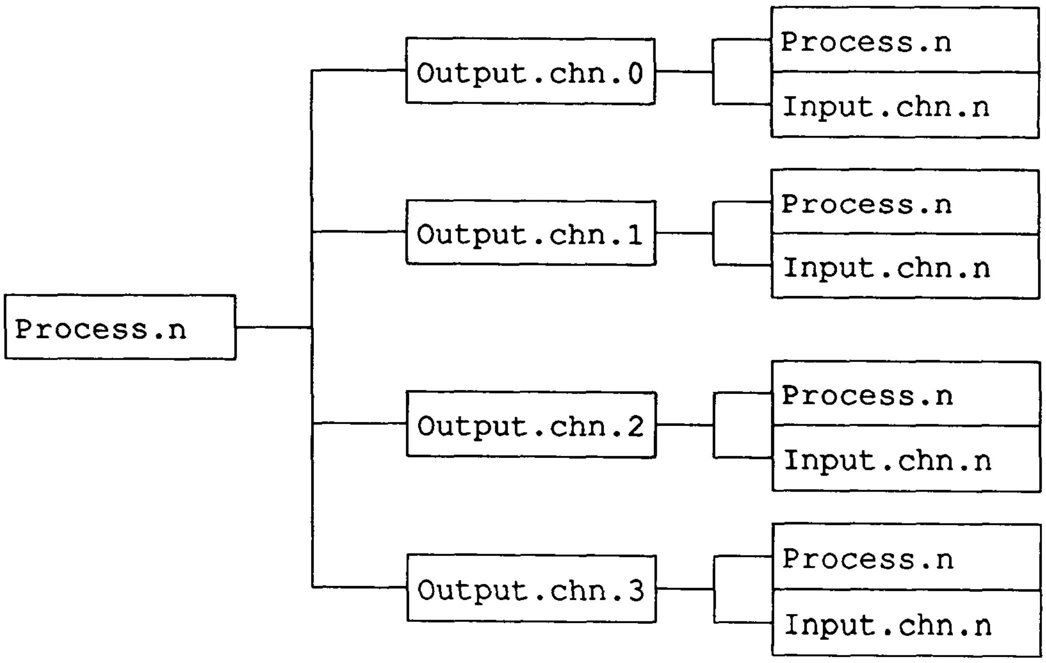


Figure (3-9)

The channels map "chn.map" is a three dimensional array and could be seen in the way illustrated in Figure (3-9) which shows the part of the array of process number <n>. The output and the input channels are numbered within the processes from 0 - 3.

Another three dimensional array is also used to keep the destinations (for the output channels) and the sources (for the input channels) for each process. The array called "connected.to" is used, the array structure is similar to the one of the array "chan.map" but each process has eight elements (the first four are used for the output channels and the last four are used for the input channels of the process) instead of the four in "chan.map". Figure (3-10) shows one element of the array which describes the channels of process <n>.

Depending on the channels map, the procedure "loading.route" detects inaccessible processes (processes which could not be accessed from the host TRAM). These processes's channels will pass all the previous checks without being discovered that they are isolated. The procedure starts with the host process and marks it as an accessible process, then depending on the channels map, it marks each process connected to the host process (by any output) as an accessible process.

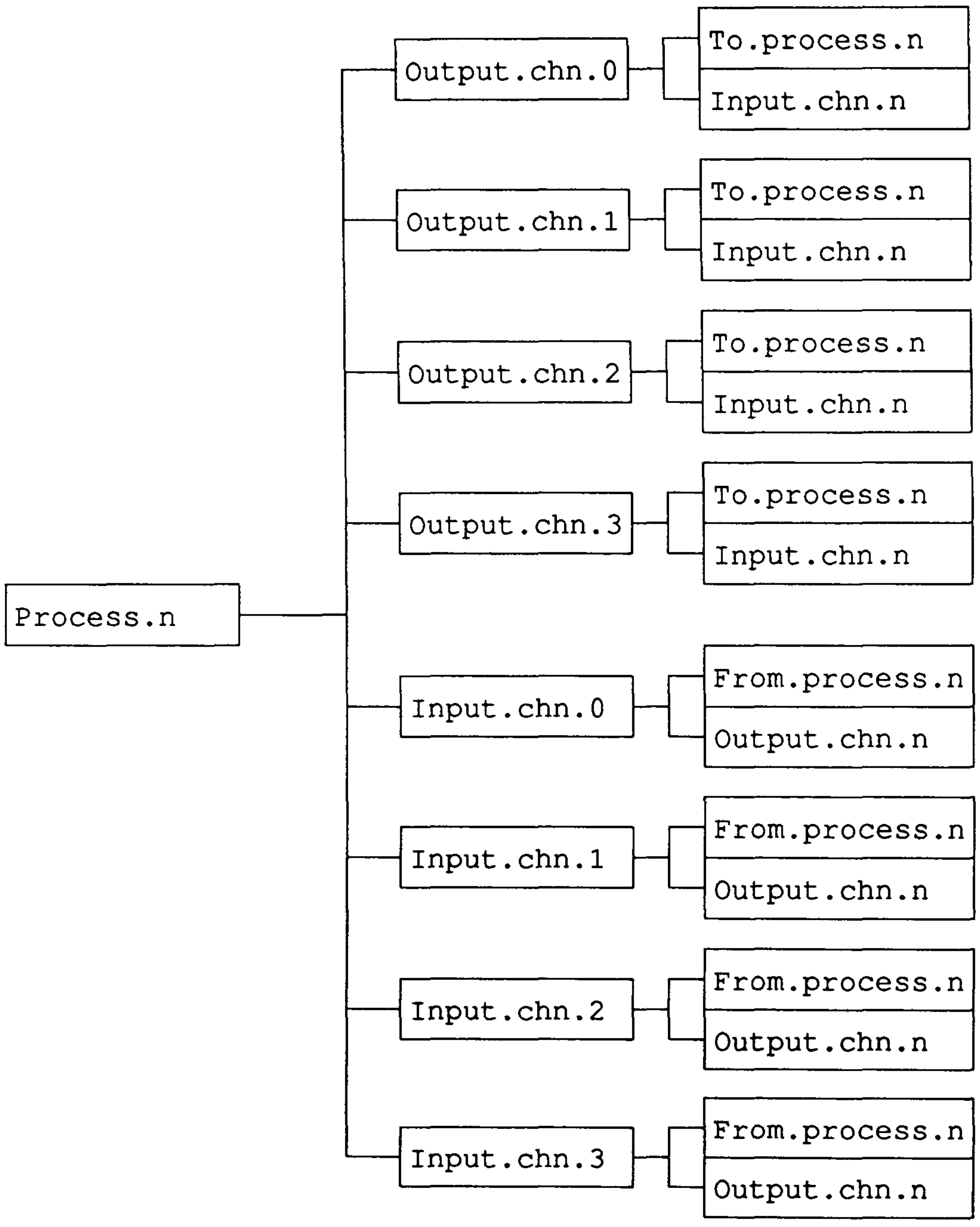


Figure (3-10)

After that the processes are scanned and all the processes connected to a marked process are marked as well. At the end a check is done to see if there ıs any unmarked process which is, ıf found, reported.

The next step which is the last step before placing the processes is producing the channels map and defining the number of processes each process is connected to and the number of links it needs.

The procedure "num.proc.conn.to" is now executed to produce the processes map "proc.proc.map" which is a two dimensional array and during that another two dimensional array "processes.n.links.n" is used to keep the number of the different processes connected to each process and the number of links it uses. An element of the processes map array (for one process) can be seen as in Figure (3-11) where (-1) is the initial value of all "with.proc.n".

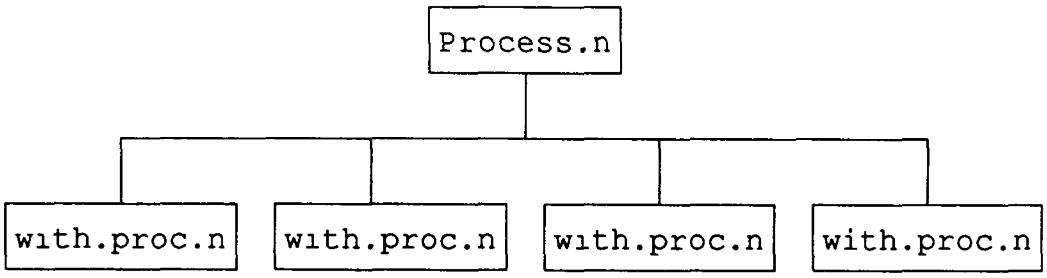


Figure (3-11)

The procedure uses the array "connected.to" created with the channels map. At first the outputs of the process are scanned, and for each used output all the inputs (of the same process) are scanned trying to find an input from the same process to which the output is connected. By this means the input and the output of each link between two

University
of Ulster
at Jordanstown

with Compliments

Professor M. Elizabeth C. Hull, B.Sc., Ph.D., C.Eng., F.B.C.S.
Head of Department of Computing Science

Newtownabbey Co. Antrim BT37 0QB Northern Ireland Telephone Belfast (0232) 365131

processes will be used if possible. If an input was found, both the output and the input are marked (cancelled from the array) and a link is reserved. The array "link.with.proc" is used to keep the number of the required links between all the processes. The structure of "link.with.proc" is identical to the structure of "chan.map" replacing the outputs with the links. After all the outputs of the process are scanned, the inputs are scanned looking for any input left. A link is reserved in the three possible cases, input and output, output only, or input only. Each time a link is needed (for input, output, or both) the processes number is updated (incremented) if this is the first link with this process. After the channels of a process are scanned, the procedure checks the number of the different processes to which the current process is connected, and the number of links required. If any of these numbers exceeds the allowed number an error message is issued.

The data required for placing the processes are now ready and correct and there should be no problem in placing the processes unless a special case (like the one mentioned above) occurred.

Processes placing algorithm:

The processes placing algorithm, which is the core of the configurer, was designed to suit the IMS B008 design and the extensibility of the system.

In a fully dynamic (reconfigurable) topology system where all the links are flexible, the needed processes placing algorithm is simple. The complexity of the algorithm grows as the restrictions on processes placing grow. As described earlier, any topology to be implemented on the IMS B008 should consider the hardwired links between the TRAMs.

The algorithm is relatively simple and could be used for placing a large number of processes at a similar number of TRAMs. The processes are placed one after another to form a chain "necklace" of processes similar to the processors chain on the board(s). Starting with the Host process (the process which access the system) placed on the Host TRAM (TRAM0 on the first IMS B008, which has two free links only) and then the other processes are placed on the other TRAMs which have four free links each.

The algorithm depends on the number of links needed by each process. Processes needing larger number of links are placed at first in the proper place to make the processes chain as long as possible.

The terms "last process" indicates the last placed process, "next process" indicates the next process to be placed, and "N links process" indicates a process which uses N links. At the beginning the host process is the last process. The next process is chosen from the processes connected to the last process according to the number of links it needs and to the position of the TRAM, at which it will be placed, in the processor chain.

A cut in the processes chain could happen if two processes are neighbours but they have no connections between them, in this case the next process could not be a four links process. A cut in the processes chain is discovered by checking the processes connected to the last process, if all of them are placed, then there is a cut in the processor chain.

The procedure "place.processes", Figure (3-12) handles the task of placing the processes according to the algorithm. Starting from the host process which is placed at the host TRAM (TRAM0) and considered the last process, the next process is defined by the procedure "define.nect.process" which selects the next process from the processes connected to the last process. The next process definition is done as follows: the processes connected to the last processes are known from the processes map and the numbers of links each of them required are known from the array

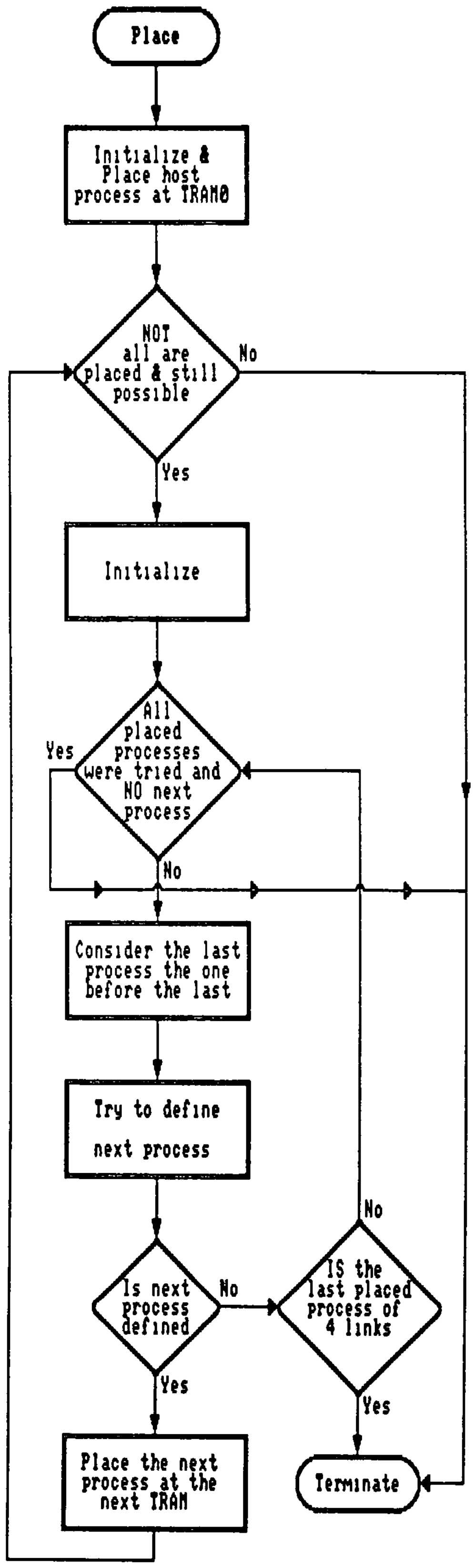

Place
Initialize & Place host process at TRAM0
NOT all are placed & still possible
No
Yes
Initialize
All placed processes were tried and NO next process
Yes
No
Consider the last process the one before the last
Try to define next process
Is next process defined
No
Yes
IS the last placed process of 4 links
No
Yes
Place the next process at the next TRAM
Terminate

Figure (3-12), Processes placing procedure "processes.n.links.n".

Two possibilities are faced here: is there a cut in the processes chain or not?

If the chain is cut (a special Boolean variable "cut" is set to TRUE) the maximum number of links the next process could have is three. If the next process is a three links process, and it is going to be placed after a cut, the possibility of all the processes, to which the next process is connected, being already placed should be considered. If they are, the next process could not be a three links process unless it will be placed at the last TRAM in the processor chain, where it could use the pipetail link. A two link process or a one link process could be placed without any problem expected.

If there is no cut, a four links process is preferable, then a three links process, then a two links process, then a one link process.

The Boolean function "all.are.placed" is used to check if the processes chain is cut after each placing, except when the last process is a one link process where the chain is definitely cut.

When a cut occurred and the next process could not be defined by "define.next.process", it returns a failure message (FALSE) to "place.processes" which goes back in the chain to the already placed processes trying to find a process connected to them and not yet placed. Unless a process is found the procedure keeps going back until the host process is reached where a process should be found because all the processes have passed the loading route test. The used TRAMs (at which the processes were placed) are marked using the Boolein array "assigned", where the corresponding element of the array is set to TRUE if a process was placed at that TRAM.

After placing the processes, the used links number(s) of each TRAM should be defined to be used by the next step, which is configuring the network.

The procedure "define.links" defines the links to be used to connect the TRAMs using the results of "num.proc.conn.to" and "place.processes".

The procedure tries at first to use the hardwired links on the IMS B008 (Link1 and Link2 of each TRAM) before using the other links (Link0 and Link3). The elements of the array "assigned" are scanned to detect the used TRAMs. If a process was placed at a TRAM, the links of this process are checked. Using the array "link.with.proc" to see with which process a link is needed. The TRAM at which the other process is placed, is then defined by the procedure "at.which.tram.is". The two TRAMs are now known, the procedure "connect.T1.T2" is used to define which link from each TRAM should be used to make the connection. The procedure considers the hardwired links when selecting the links. Link0 and Link3 are used if the TRAMs are not neighbours or Link1 (of the first) and Link2 (of the second) are used. In the case where TRAM0 is involved, Link3 is the only option. The pipetail link is used if the TRAM is the last in the processor chain. The results are kept in the array "tram.link" which is a three dimensional array. Figure (3-13) illustrates an element of "tram.link".

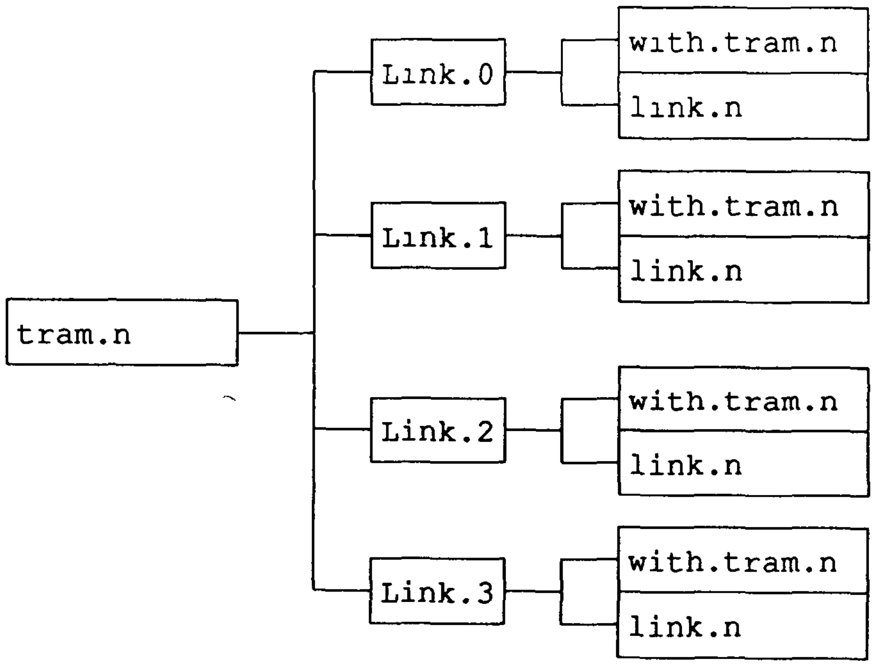


Figure (3-13)

The last step in phase two is configuring the network according to the results of the previous steps (the required topology).

Procedure "network.connector" reads the array "tram.link" produced by the previous step and makes the new connections (not hardwired). If Link0 or Link3 is used in a TRAM, a new connection should be made to connect this link to the appropriate link in the other TRAM (Link0 or Link3). The first thing which should be done before the connection could be made is to define at which slots the TRAMs are. Procedure "at.which.slot.is" defines the slot which accommodates the TRAMs, then the links of the crossbar switch are defined, and after that the configure data is sent to the controller process on the T2 to set the crossbar switch.

3-4-6-4-3 Phase III:

The last phase of configuring a program is optional. This phase produces the configured output folds which should be made foldsets before being compiled, Figure (3-14) .

The first one contains the program processes, the external channels declarations, the protocols of the external channels, and the required configuring statements.

The second fold contains the host process, the external channels declaration of the host process, the protocols of the external channels, and the configuring statements.

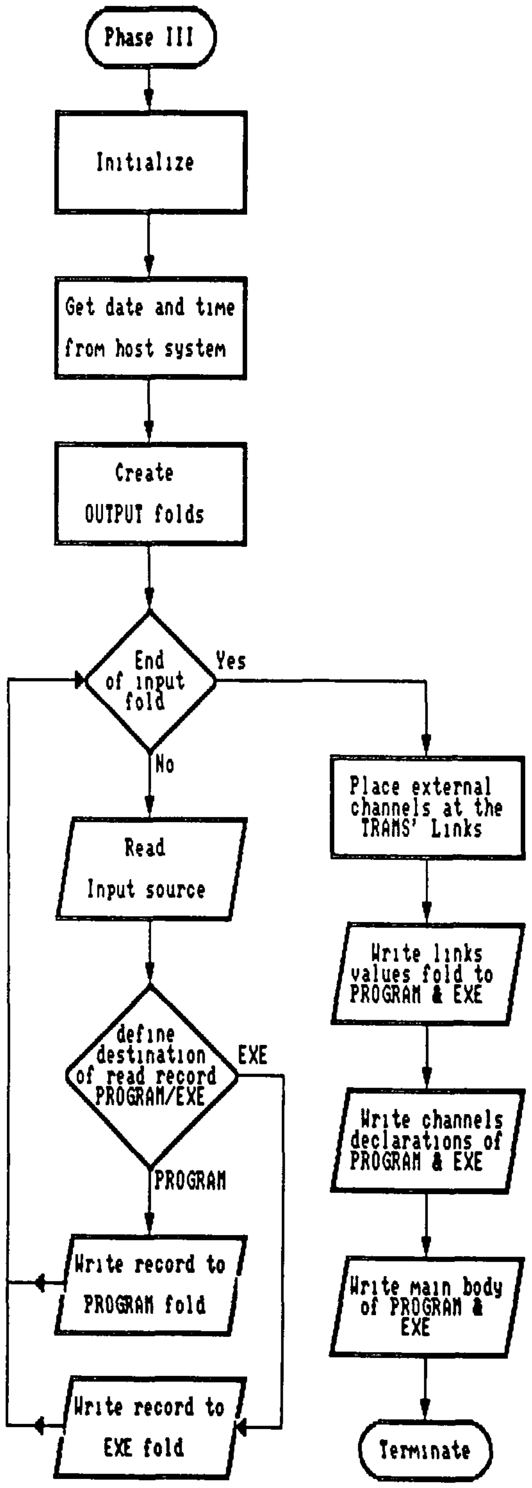

Phase III
Initialize
Get date and time from host system
Create OUTPUT folds
End of input fold
Yes
No
Read Input source
define destination of read record PROGRAM/EXE
EXE
PROGRAM
Write record to PROGRAM fold
Write record to EXE fold
Place external channels at the TRANS' Links
Write links values fold to PROGRAM & EXE
Write channels declarations of PROGRAM & EXE
Write main body of PROGRAM & EXE
Terminate

Figure (3-14)

The first fold should be made of type "PROGRAM" and the second fold should be made of type "EXE" by the "make.fold.set" utility of the compiler.

The execution of this phase was made optional to allow the user to configure the transputer network before running previously configured programs (by the configurer) because any time the program is used to configure a program the results will be same unless the program source was changed.

The configurer will copy the source of the processes only, even if they were compiled, to make sure that everything is checked before the network is loaded with the executable code of the configured program. The creation time and the date of the output folds are added to the folds comment to help the user to differentiate between the outputs of multiple times of configuring for the same program, although the TDS adds the new created folds to the end of the bundle. The external channels placing step was made a part of phase three because there is no point in placing the channels if the output is not required.

The phase starts with getting the date and the time from the host system by creating a new file and reading it's characteristics, which includes the time of creation and the date, and then the file is deleted. The procedure "date.time" is used to get the date and the time. After that the comment and the attributes of the fold contains the program source are read to be used for creating the output folds.

Two fold are then created to contain the configured output. The source of the input program is then read and the contents are copied to the proper new fold.

Then the procedure "place.channels" places the external channels at the links of the transputers.

The channels are placed on the links which was defined earlier by "define.links". The array "tram.link" describes the link connections between the TRAMs, and the array "chn.map" describes the channel connections between the processes. From both arrays in addition to "assigned" array, the channels of the processes are placed at the links. The results of the channels placing are kept in two arrays, one for output channels "o.channel.at.link" and for input channels "i.channel.at.link", both arrays are two dimensional, the first dimension is the transputer number, and the second dimension is the link number, at which the output or input channel number of the process placed at that transputer is placed.

After the input source is read, both the new folds will contain a copy of the fold called the "EXTERNAL CHANNELS PROTOCOLS". The first will also contain a copy of the source of each process. The second will also contain a copy of the host process.

The configuring statements are now added to both folds. A fold contains definitions of the links output and input placing values, to be used instead of the values which are meaningless, Figure (3-7).

Then a fold contains the declarations of the channels used by the process(es) contained in the fold is added.

```
{{{  Links
VAL Link0Out IS 0:
VAL Link1Out IS 1:
VAL Link2Out IS 2:
VAL Link3Out IS 3:
VAL Link0In  IS 4:
VAL Link1In  IS 5:
VAL Link2In  IS 6:
VAL Link3In  IS 7:
}}}
```

The values of the links output and input parts.

Figure (3-7)

The procedure "place.EXE.ext.chnls" adds the placing statements of the host process external channels, written in a fold called "channels placing", to the second fold.

The procedure "PROGRAM.prog.body" adds the configuring statements to the first fold. For each processor (transputer) used, the placing statements of the external channels of the process placed at that processor's links are added in a fold called "channels placing" followed by a fold containing the calling statement of the process placed at that processor, Figure (3-8).

```
      {{{  program body
      PLACED PAR
        {{{  processor 0
        PROCESSOR 0 T8
          {{{  channels placing
          PLACE channel name AT LinkxOut:
          PLACE channel name AT LinkyOut:
            .
            .
          PLACE channel name AT LinkxIn :
            .
            .
          }}}
          {{{  process calling
          process.name (calling parameters§)
          }}}
        }}}
        ...  processor 1
         .
         .
        ...  processor n
      }}}

  x,y are links numbers.
§ All the process external channels should be among the
  calling parameters.

Contents of the program body fold after configuring
```

Figure (3-8)

Chapter 4

Using the Configurer

4-1 Introduction:

The program was tried out on experimental programs in order to prove it's correctness. Four different programs requiring four different topologies are discussed as examples of using the program to configure programs to run on the IMS B008.

4-2 A program requiring a hypercube topology:

The required topology is illustrated in Figure (4-1) where the numbers are the processes numbers (names) and the hypercube consists of the processes 1,2,3,4,5,6,7,8.

Process 0 is the Host process. A channel from process 8 to process 0 was added.

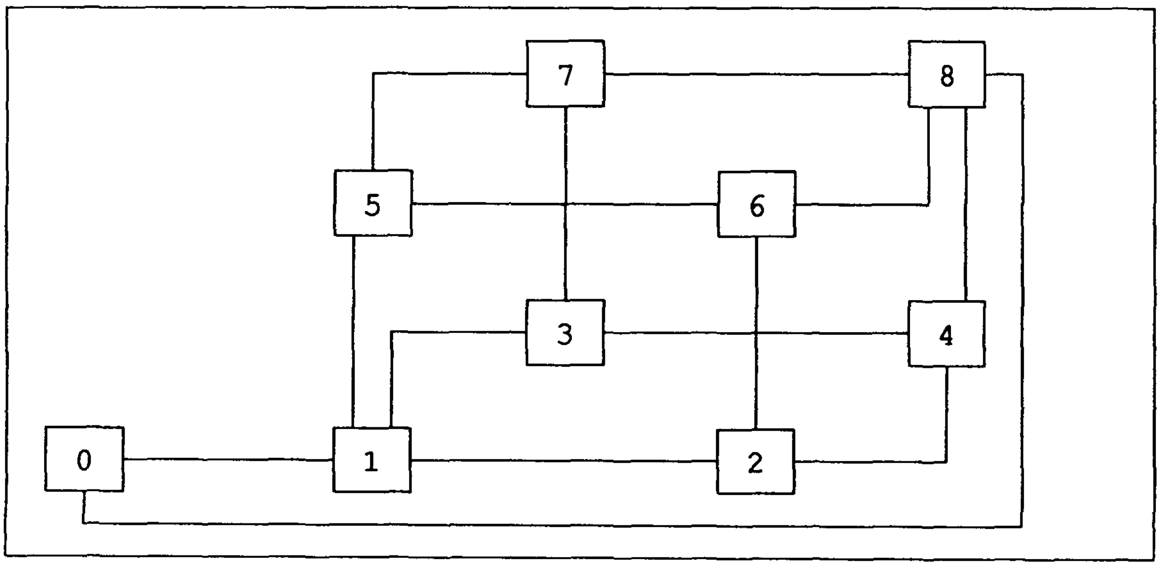
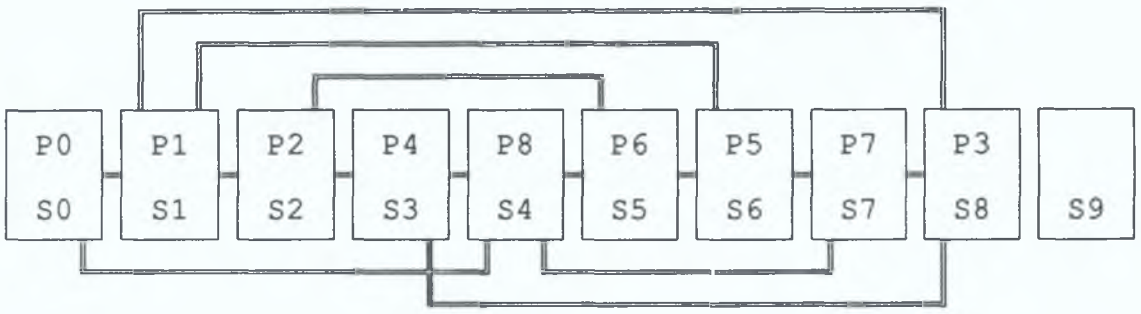


Figure (4-1)

From the configured output (program) of the configurer the processes were placed at the TRAMs as illustrated in Figure (4-2).



P: Process
S: Slot

Figure (4-2)

Refer to Appendix B for full listing of the input program and the configured outputs.

4-3 A program requiring a mesh topology:

The required topology is illustrated in Figure (4-3).

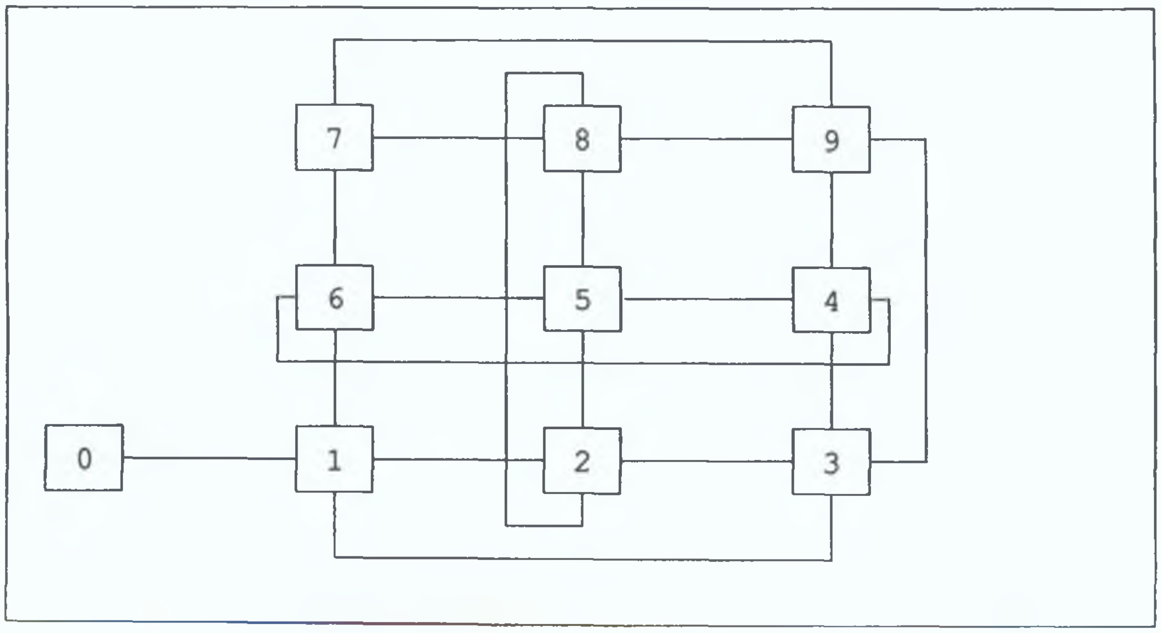


Figure (4-3)

the processes were placed at the TRAMs as illustrated in Figure (4-4).

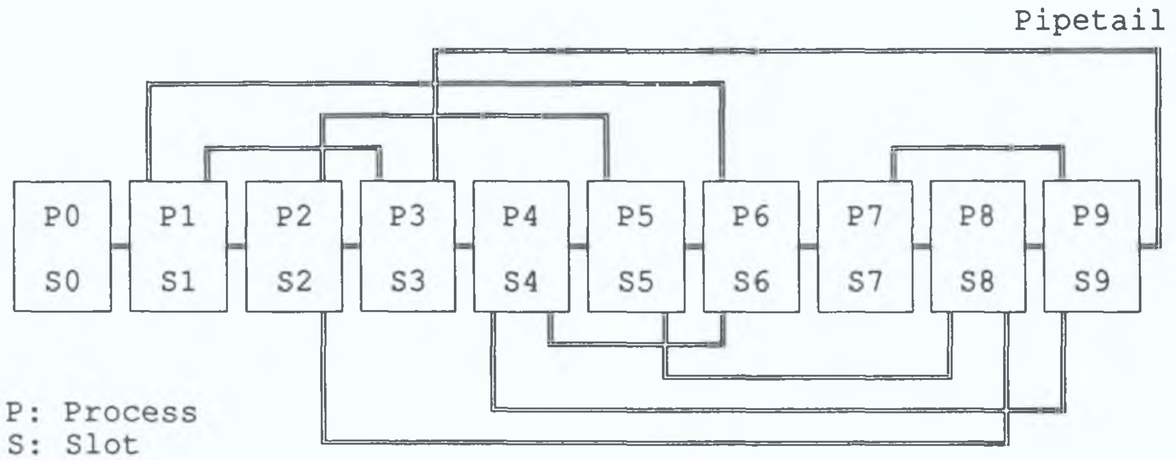


Figure (4-4)

In this example the pipetail (link 2 of the last TRAM) was used to implement the required topology and it supposed to be connected to the crossbar switch through the edge connector using the link EDGE 0.

Refer to Appendix C for full listing of the input program and the configured outputs.

4-4 A program requiring a binary tree topology:

The required topology is illustrated in Figure (4-5).

This example shows clearly the difficulties imposed by the hardwired links, which made these links unusable in this topology. A look at the configurer results (Appendix D, D-1-1-2) will tell that Link 0 is used in process 5,7,9 although the processes have only one external channel. That indicates that the process pipe were cut three times during the mapping of the processes on the TRAMs. The effect of the hardwired links could have been stronger if more than one link is required between two processes.

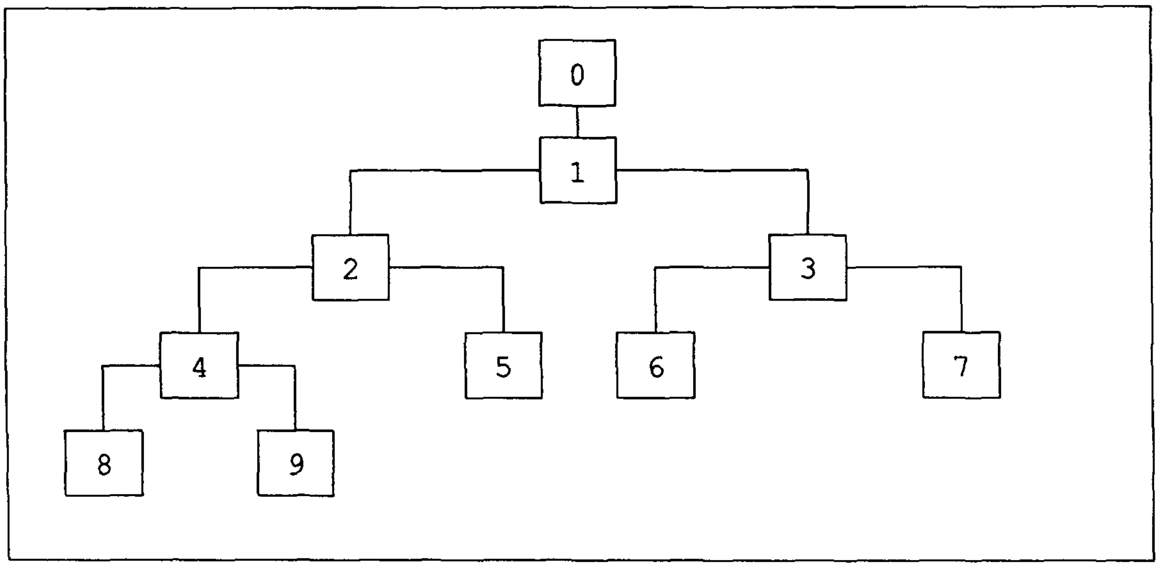
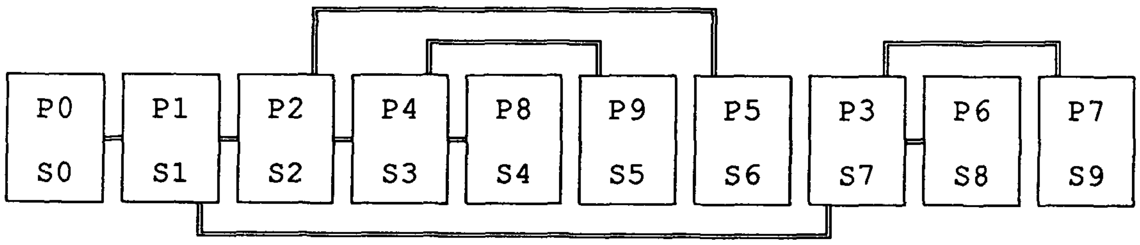


Figure (4-5)

From the configured output (program) of the configurer the processes were placed at the TRAMs as illustrated in Figure (4-6).



P: Process
S: Slot

Figure (4-6)

4-5 A program requiring a user defined topology:

The required topology is illustrated in Figure (4-7). The topology divided the processes into two parts each part is connected to the host process and not there is no connectıon between the two parts.

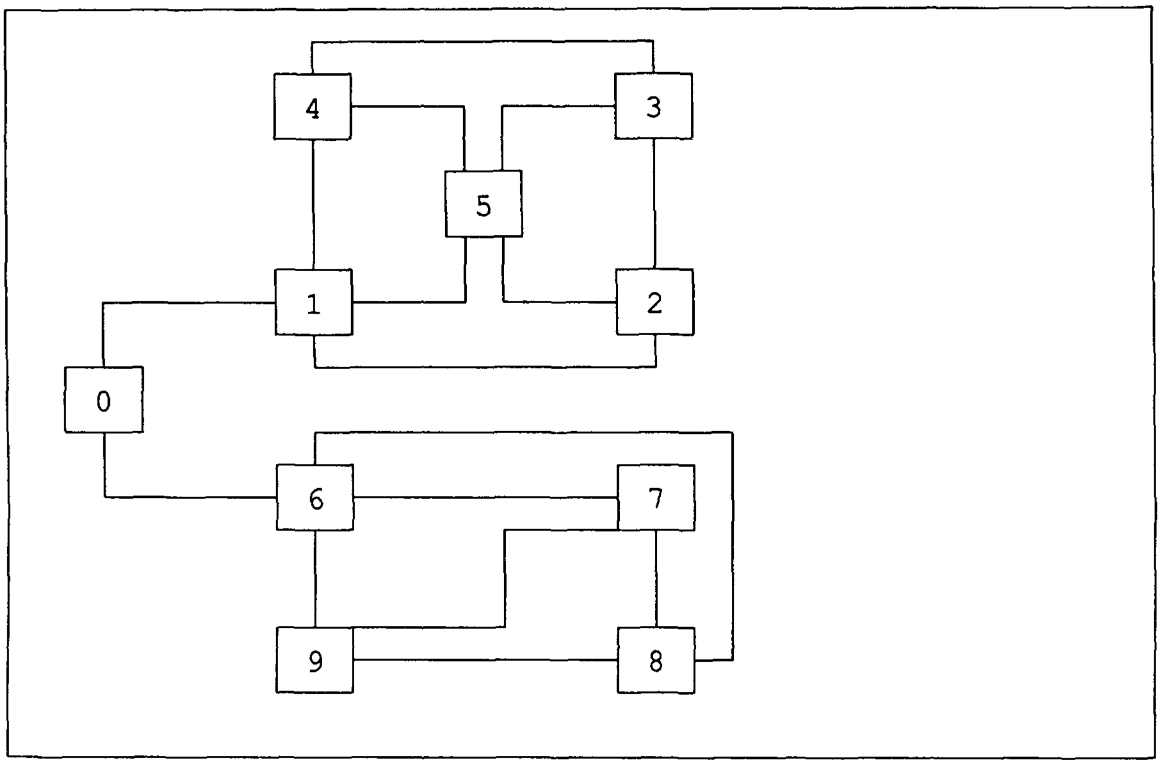
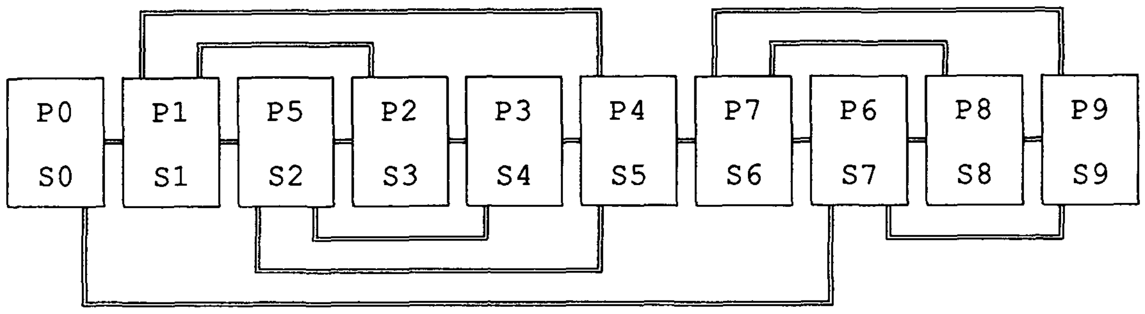


Fıgure (4-7)

The loader will fail to send the executable code to one of the parts (the one placed far from the host) no matter how the processes are mapped on the TRAMs. To avoıd the loader failure the configurer adds a channel called "JUST FOR LOADING" and places it so that the two parts are connected. Appendix E contains the input program and the configured outputs. The loadıng channel was added to process 4 and process 7 and was placed at the hardwired lınk.

From the configured output (program) of the confıgurer the processes were placed at the TRAMs as illustrated ın

Figure (4-8).



P: Process
S: Slot

Figure (4-6)

The user at this stage should make the fold marked with (program) a Foldset of type PROGRAM and the fold marked with (exe) a Foldset of type EXE and then follow the TDS procedure of compiling and running a program on a network.

4-6 Configuring the network:

The program could be used to configure the transputer network before running a program previously configured by the configurer. Any number of programs could be run from within the TDS even if they require different topologies.

All the user should do to get the system tailored to his program requirements is to run the configurer with its input being the program he/she would like to run.

Modifications on the program processes are highly expected during the development stages of any program, but these modifications will not affect the configurer results, and that eliminate the need to configure the program each time it is modified.

Modifications on the external channels require reconfiguring as it will definitely change the configurer results. Changing the calling order of the processes may change the configurer results, so its advisable to reconfigure the program if such modification is done.

The program was run many times on the above examples and the results were the same for each of them each time the configurer was run on it.

Improvements and Further Development

5-1 Introduction:

Designing and improving parallel programs configuring facilities will always be required to ease the use of parallel machines and make parallel programs writing as simple as possible.

The way is now opened for anyone who wants to improve the configurer or even to design a new one, with a different strategy, using the available configurer as a starting point.

Improvements could be done on the configurer to increase it's ability to deal with larger programs, and larger networks, to support replicated PLACED PAR constructs, to deal with networks of mixed transputer types, and to be more friendly to the user.

5-2 Improvements on the configurer program:

5-2-1 Handle larger number of processes and processors:

In its first version the configurer is able to deal with relatively small number of processes and small network size, which could run on one IMS B008. With some modifications on parts of the program, it will be able to deal with larger number of processes and processors.

Parts like the network detector, the network connector, and the links definer needs to be modified in order to achieve a configurer for larger programs.

5-2-2 Handle replicated PLACED PAR construct:

The configurer at this time does not accept a replicated "PAR" construct, which is essential for some applications, where identical processes should run in parallel (on different processors). A suggestion to solve this problem is to have a procedure which translates the replicated "PLACED PAR" construct into a number of process calling statements, after reading the program source. Another procedure is required to re-build the replicated "PLACED PAR" construct to be written in the output fold.

5-2-3 Mixed transputer types:

To make the configurer able to deal with a transputer network which contains mixed transputer types, the network detecting procedure should be modified to detect the transputer type while detecting the network. That may create new problems to the user in cases like when a process should run on a specific transputer type, because of it's capabilities (eg floating point). The programmer has then to specify in the program the processor type required for specific process(es) which will complicate the processes placing algorithm.

5-2-4 Unlimited number of processes:

To make the system hardware more transparent to the user, the configurer should be able to use with an unlimited number of processes (i.e., the number of processes is not related to the number of transputers in the network). This will be at the price of real parallelism which is not acceptable for some applications where time is a crucial factor. To achieve that, the configurer will have to combine (group) some processes, according to their communication requirements and if possible to their computational requirements (short processes), in one process. The combined processes will run concurrently sharing the time of one processor.

Another difficulty will also be faced in this approach. This is placing the external channels of the combined processes on the transputer links. In general the transputer links will not be enough, and that leads to the use of a message passing technique (discussed in chapter 4). Using any message passing technique requires new processes to be added, by the configurer, to the program to handle the message passing between the processors. This will achieve the transparency goal but, it also has the disadvantages of the technique used, and may decrease the performance of the application program, even if it uses a very good and efficient parallel algorithm.

To make the process grouping approach more efficient, the user could be asked, by the configurer, to define the processes to be grouped. This will increase the efficiency of the system but will not get rid of the used message passing technique disadvantages.

5-2-5 Use the configurer instead of the MMS2:

The configurer configures the transputer network from within the TDS. This facility, if made accessible to the user (as an option), will allow the user to configure the network as desired, for programs which are not configured by the configurer.

5-2-6 Channels protocols check:

To avoid compiler error messages about unmatched protocols, protocols of channels connected together could be checked.

5-2-7 User friendly:

To give the user more confidence, the configurer could show (on the screen) the network while it is being configured or after being configured. The TRAMs could be drawn as boxes and then the configurer will draw lines between the boxes to represent the links. The names of the processes and the channels could also be placed at the boxes and the lines to give the user a realistic image of the network.

Some other improvements could also be done, but these were my next steps.

Conclusions

The configurer program was used to configure a number of different programs and proves to be efficient.

Configuring a program by the configurer results in two folds. The first fold, which should be made a Foldset of type PROGRAM, contains the processes of the input program placed at the transputers and the channels placed at the transputers links. The second fold, which should be made a Foldset of type EXE, contains the host process and the channels placed at the links of the host transputer. The network is also configured to meet the communication requirements of the configured program processes. All the special cases mentioned in chapter three were tried and the configurer gave the expected results. All errors which could affect the progress of the configuring process were also tried, and the configurer proved it's ability to discover and report them.

The configurer detects and reports most of the expected syntax errors in the specific folds which contain the configuration data. The configurer could be used to configure the network before running a program which was previously configured by the configurer because multiple runs of the configurer on the same input file gave always the same results. Programs written in any languge could be configured.

References:

[1] - **D. Aspinall** " Structures for parallel processing ", Computing & Control Engineering Journal, JAN. 1990.

[2] - **Brian Oakley** " The limits to growth in IT ", Computing & Control Engineering Journal, JAN. 1990.

[3] - **D. Parkinson** "The Processing Array Approach", Major Advances in Parallel Processing, pp 120-129, Edt. Chris Jesshope, Technical Press 1987.

[4] - **F.H. Schlereth, B.F. Schlereth** " KILONODE A Transputer Based Parallel Computer ", Transputer Research And Applications 1, pp 82-88, IOS Press 1990.

[5] - **J. Clifton Hughes** " ParSiFal - the Parallel Simulation Facility ", Major Advances in Parallel Processing, pp 146-154, UNICOM 1987.

[6] - **INMOS Ltd.** " The Transputer Development and *i*q Systems Databook ", pp 98-101, INMOS Ltd. 1989.

[7] - **Almasi/Gottlieb** "Highly parallel computing", The Benjamin/commings 1989.

[8] - **C. Rieger, J. Bane, R. Trigg** "ZMOB: a highly parallel multiprocessor", Proceedings of the Workshop on Picture Data Description and Management, page 281, Aug. 1980.

[9] - **Charles L. Seitz** "Concurrent VLSI Architectures", IEEE Trans. on computers, Vol C-33, No 12, Dec. 1984.

[10] - **Howard Jay Siegel,William Tsun-Yuk Hsu** "Interconnection Networks", Computer Architecture Concepts and Systems, pp 225-264, Edt. Veljko M. Milutinovic, North-Holland 1988.

[11] - **Angel L. DeCegama** "THE TECHNOLOGY OF PARALLEL PROCESSING, Parallel Processing Architectures and VLSI Hardware,

Volume 1", Prentice Hall, 1989.

[12] - **Reed,D., Grunwald,D.,** "The performance of multiprocessor interconnection networks", Computer, pp 63-73, Vol 20, No 6, June 1987.

[13] - **Bhuyan,L., Yang,Q., Agrawal,D.** "Performance of multiprocessor interconnection networks", Computer, pp 25-37, Vol 22, No 2, Feb 1989

[14] - **Rieger C** "ZMOB A mob of 256 cooperative Z80-based microcomputers", Proc of the DARPA Image Understanding Workshop, Los Angeles, CA, 1979

[15] - **Rieger C,** et al "ZMOB A new computing engine for AI", Proc Seventh International Joint Conference on Artificial Intelligence, IJCAI, pp 955-960, Aug 1981 [16] - **G. Mago** "A Cellular Computer Architecture for Functional Programming", IEEE COMPCON, pp 179-187, 1980

[17] - **S. J. Stolfo** and **D. P. Miranker** "DADO A parallel Processor for Expert Systems", IEEE International conference on parallel processing, pp 74-82, 1984

[18] - **W. Daniel Hillis** "The Connection Machine", MIT press 1985.

[19] - **D. H. Lawrie** "Access and Alignment of Data in Array Processor", IEEE Trans. on Computers, pp 55-56, Jan. 1976.

[20] - **Martin Shumway** "Deadlock-Free Packet Network", Transputer Research and Applications 2, pp 139-177, IOS Press 1990.

[21] - **Dick Pountain** "Configuring parallel programs", Byte pp 349-352, Dec. 1989

[22] - **INMOS Ltd.** " The Transputer Databook ",INMOS Ltd. 1989.

[23] - **D.G.Shea, R.C.Booth, D.H.Brown, M.E.Giampapa, G.R.Irwin, T.T.Murakami, F.T.Tong, P.R.Varker, V.W.Wilcke** and **D.J.Zukowski,**

"Monitoring and Simulation of Processing Strategies for Large Knowledge Bases on the IBM Victor Multiprocessor", Transputer Research and Applications 2, pp 11-26, IOS Press 1990.

[24] - **Mark Smith, Jonathan Yen,** and **Susan Spach,** "A Transputer Based Architecture for Data Broadcasting", Transputer Research and Applications 1, pp 67-72, IOS Press 1990

[25] - **Thomas B.Henderson, Jerome J.Symanski,** and **Keith Bromley,** "Software Development on the Video Analysis Transputer Array", Transputer Research and Applications 1, pp 88-97, IOS Press 1990.

[26] - **Jack Harper** "Variable Topology Parallel Processing on the Sun A Graphics Based, Mouse Driven Approach", Transputer Research and Applications 1, pp 98-105, IOS Press 1990

[27] - **Peter C.Capon,** and **Alan E.Knowles** "Using Algorithmic Parallilism in the Manchester ParSiFal Syatem", Transputer Research and Applications 1, pp 57-66, IOS Press 1990.

[28] - **Peter Jones,** and **Alan Murta** "The Implementation of a Run-Time Link-Switching Environment for Multi-Transputer Machines", Transputer Research and Applications 2, pp 107-122, IOS Press 1990.

[29] - **INMOS** "IMS B008 User guide and reference manual", INMOS June 1988.

[30] - **INMOS Ltd.** " Occam 2 Reference Manual ", PRENTICE HALL 1988.

[31] - **David May** " The Transputer ", Major Advances in Parallel Processing, pp 33-43, Technical Press 1987.

[32] - **Alan Burns** " Programming In Occam 2 ", Addison-Wesley 1988

[33] - **Jon Kerridge** " Occam Programming· A Practical Approach ", Blackwell Scientific Publications 1988

[34] - **INMOS** "Transputer development system", PRENTICE HALL, 1988

[35] - **INMOS** "Module motherboard software", INMOS 1988

[36] - **INMOS** "The Transputer Application Notebook, systems and performance", INMOS 1989.

[37] - **Manas Mandal, Prasad Vishnubhotla, Kalluri Eswar,** and **Chandrasekhar Gollamud,** "ALPS on a transputer network Kernel Support for Topology-Independent Programming", Transputer Research and Applications 2, pp 229-252, IOS Press 1990

Appendix A

The text in italic is the text added for configuring the example program in chapter 2.

A-1 Program listing with configuring instructions:

```
...  External channels protocols declaration fold
...  Paralle processes fold
{{{  External channels declaration
CHAN OF <protocol.name> from.0.to.1:
CHAN OF <protocol.name> from.1.to.0:
CHAN OF <protocol.name> from.1.to.2:
CHAN OF <protocol.name> from.1.to.3:
CHAN OF <protocol.name> from.1.to.4:
CHAN OF <protocol.name> from.2.to.1:
CHAN OF <protocol.name> from.2.to.3:
CHAN OF <protocol.name> from.2.to.4:
CHAN OF <protocol.name> from.3.to.1:
CHAN OF <protocol.name> from.3.to.2:
CHAN OF <protocol.name> from.3.to.4:
CHAN OF <protocol.name> from.4.to.1:
CHAN OF <protocol.name> from.4.to.2:
CHAN OF <protocol.name> from.4.to.3:
}}}
PLACED PAR

  PROCESSOR 0 T8
    {{{  process.1 external channels placing
    PLACE from.1.to.0 AT Link1Out:
    PLACE from.1.to.2 AT Link2Out:
    PLACE from.1.to.3 AT Link0Out:
    PLACE from.1.to.4 AT Link3Out:
    PLACE from.0.to.1 AT Link1In:
    PLACE from.2.to.1 AT Link2In:
    PLACE from.3.to.1 AT Link0In:
    PLACE from.4.to.1 AT Link3In:
    }}}
    process.1 (External channels of the process, ...)

  PROCESSOR 1 T8
    {{{  process.2 external channels placing
    PLACE from.2.to.1 AT Link1Out:
    PLACE from.2.to.3 AT Link2Out:
    PLACE from.2.to.4 AT Link3Out:
    PLACE from.1.to.2 AT Link1In:
    PLACE from.3.to.2 AT Link2In:
    PLACE from.4.to.2 AT Link3In:
    }}}
    process.2 (External channels of the process, ...)

  PROCESSOR 2 T8
    {{{  process.3 external channels placing
    PLACE from.3.to.1 AT Link0Out:
    PLACE from.3.to.2 AT Link1Out:
      PLACE from.3.to.4 AT Link2Out:
      PLACE from.1.to.3 AT Link0In:
      PLACE from.3.to.3 AT Link1In:
      PLACE from.4.to.3 AT Link2In:
      }}}
      process.3 (External channels of the process, ...)

    PROCESSOR 3 T8
      {{{  process.4 external channels placing
      PLACE from.4.to.1 AT Link0Out:
      PLACE from.4.to.2 AT Link3Out:
      PLACE from.4.to.3 AT Link1Out:
      PLACE from.1.to.4 AT Link0In:
      PLACE from.2.to.4 AT Link3In:
      PLACE from.3.to.4 AT Link1In:
      }}}
      process.4 (External channels of the process, ...)
```

A-2 Host program listing with configuring instructions:

```
  ...  External channels protocols declaration fold
  ...  Host program processes
  {{{  Host external channels declaration
  CHAN OF <protocol.name> from.0.to.1:
  CHAN OF <protocol.name> from.1.to.0:
  }}}
  {{{  process.1 external channels placing
  PLACE from.0.to.1 AT Link2Out
  PLACE from.1.to.0 AT Link2In:
  }}}
  ...  Host program body
```

HYPERCUBE

For simplicity all the channels have the same protocol (ANY), but any other protocol could be used.

B-1 The required part of the program source:

```
...  EXTERNAL CHANNELS PROTOCOLS
{{{  PROCESSES
{{{  SC process.0
{{{  COMMENT PROCESS process.0 EXTERNAL CHANNELS
CHAN OF ANY from.0.to.1 ,
            from.0.to.8 !: -- output channels from 0
CHAN OF ANY from.1.to.0 ,
            from.8.to.0 ?: -- input channels to 0
}}}
{{{  process.0
PROC process.0 (...)
  ...  process body
:
}}}
}}}
{{{  SC process.1
{{{  COMMENT PROCESS process.1 EXTERNAL CHANNELS
CHAN OF ANY from.1.to.0,
            from.1.to.2,
            from.1.to.3,
            from.1.to.5 !:
CHAN OF ANY from.0.to.1,
            from.2.to.1,
            from.3.to.1,
            from.5.to.1 ?:
}}}
{{{  process.1
{{{
PROC process.1 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.2
{{{  COMMENT PROCESS process.2 EXTERNAL CHANNELS
CHAN OF ANY from.2.to.1,
            from.2.to.4,
            from.2.to.6 !:
CHAN OF ANY from.1.to.2,
            from.4.to.2,
            from.6.to.2 ?:
}}}
```

```
{{{  process.2
{{{
PROC process.2 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.3
{{{  COMMENT PROCESS process.3 EXTERNAL CHANNELS
CHAN OF ANY from.3.to.1,
            from.3.to.4,
            from.3.to.7 !:
CHAN OF ANY from.1.to.3,
            from.4.to.3,
            from.7.to.3 ?:
}}}
{{{  process.3
{{{
PROC process.3 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.4
{{{  COMMENT PROCESS process.4 EXTERNAL CHANNELS
CHAN OF ANY from.4.to.2,
            from.4.to.3,
            from.4.to.8 !:
CHAN OF ANY from.2.to.4,
            from.3.to.4,
            from.8.to.4 ?:
}}}
{{{  process.4
{{{
PROC process.4 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.5
{{{  COMMENT PROCESS process.5 EXTERNAL CHANNELS
CHAN OF ANY from.5.to.1,
            from.5.to.6,
            from.5.to.7 !:
CHAN OF ANY from.1.to.5,
            from.6.to.5,
            from.7.to.5 ?:
}}}
{{{  process.5
{{{
PROC process.5 (...)
```

```
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.6
{{{  COMMENT PROCESS process.6 EXTERNAL CHANNELS
CHAN OF ANY from.6.to.2,
            from.6.to.5,
            from.6.to.8 !:
CHAN OF ANY from.2.to.6,
            from.5.to.6,
            from.8.to.6 ?:
}}}
{{{  process.6
{{{
PROC process.6 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.7
{{{  COMMENT PROCESS process.7 EXTERNAL CHANNELS
CHAN OF ANY from.7.to.3,
            from.7.to.5,
            from.7.to.8 !:
CHAN OF ANY from.3.to.7,
            from.5.to.7,
            from.8.to.7 ?:
}}}
{{{  process.7
{{{
PROC process.7 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.8
{{{  COMMENT PROCESS process.8 EXTERNAL CHANNELS
CHAN OF ANY from.8.to.0,
            from.8.to.4,
            from.8.to.6,
            from.8.to.7 !:
CHAN OF ANY from.0.to.8,
            from.4.to.8,
            from.6.to.8,
            from.7.to.8 ?:
}}}
{{{  process.8
{{{
PROC process.8 (...)
  ...  process body
```

```
:
}}}
}}}
}}}
}}}
{{{  CONFIGURE
PLACED PAR
  process.0 (...) -- Host process should be the first
  process.1 (...)
  process.2 (...)
  process.3 (...)
  process.4 (...)
  process.5 (...)
  process.6 (...)
  process.7 (...)
  process.8 (...)
}}}
```

B-1-1 The configurer results:

B-1-1-1 The configured program:

```
{{{  EXTERNAL CHANNELS PROTOCOLS
{{{  protocol declaration
PROTOCOL REAL.ARRAY IS INT::[]REAL64:
}}}
}}}
{{{  PROCESSES
...  SC process.1
...  SC process.2
...  SC process.3
...  SC process.4
...  SC process.5
...  SC process.6
...  SC process.7
...  SC process.8
}}}
{{{  Links
VAL Link0Out IS 0:
VAL Link1Out IS 1:
VAL Link2Out IS 2:
VAL Link3Out IS 3:
VAL Link0In  IS 4:
VAL Link1In  IS 5:
VAL Link2In  IS 6:
VAL Link3In  IS 7:
}}}
{{{  external channels declaration
CHAN OF ANY from.0.to.1:
CHAN OF ANY from.0.to.8:
CHAN OF ANY from.8.to.0:
CHAN OF ANY from.8.to.4:
CHAN OF ANY from.8.to.6:
CHAN OF ANY from.8.to.7:
```

```
CHAN OF ANY from.1.to.0:
CHAN OF ANY from.1.to.2:
CHAN OF ANY from.1.to.3:
CHAN OF ANY from.1.to.5:
CHAN OF ANY from.2.to.1:
CHAN OF ANY from.2.to.4:
CHAN OF ANY from.2.to.6:
CHAN OF ANY from.3.to.1:
CHAN OF ANY from.3.to.4:
CHAN OF ANY from.3.to.7:
CHAN OF ANY from.4.to.2:
CHAN OF ANY from.4.to.3:
CHAN OF ANY from.4.to.8:
CHAN OF ANY from.5.to.1:
CHAN OF ANY from.5.to.6:
CHAN OF ANY from.5.to.7:
CHAN OF ANY from.6.to.2:
CHAN OF ANY from.6.to.5:
CHAN OF ANY from.6.to.8:
CHAN OF ANY from.7.to.3:
CHAN OF ANY from.7.to.5:
CHAN OF ANY from.7.to.8:
}}}
{{{  program body
PLACED PAR
  {{{  processor 0
  PROCESSOR 0 T8
    {{{  channels placing
    PLACE from.1.to.0 AT Link1Out:
    PLACE from.1.to.2 AT Link2Out:
    PLACE from.1.to.3 AT Link0Out:
    PLACE from.1.to.5 AT Link3Out:
    PLACE from.0.to.1 AT Link1In:
    PLACE from.2.to.1 AT Link2In:
    PLACE from.3.to.1 AT Link0In:
    PLACE from.5.to.1 AT Link3In:
    }}}
    {{{  process calling
    process.1 (any)
    }}}
  }}}
  {{{  processor 1
  PROCESSOR 1 T8
    {{{  channels placing
    PLACE from.2.to.1 AT Link1Out:
    PLACE from.2.to.4 AT Link2Out:
    PLACE from.2.to.6 AT Link0Out:
    PLACE from.1.to.2 AT Link1In:
    PLACE from.4.to.2 AT Link2In:
    PLACE from.6.to.2 AT Link0In:
    }}}
    {{{  process calling
    process.2 (any)
    }}}
```

```
}}}
{{{  processor 2
PROCESSOR 2 T8
  {{{  channels placing
  PLACE from.4.to.2 AT Link1Out:
  PLACE from.4.to.3 AT Link0Out:
  PLACE from.4.to.8 AT Link2Out:
  PLACE from.2.to.4 AT Link1In:
  PLACE from.3.to.4 AT Link0In:
  PLACE from.8.to.4 AT Link2In:
  }}}
  {{{  process calling
  process.4 (any)
  }}}
}}}
{{{  processor 3
PROCESSOR 3 T8
  {{{  channels placing
  PLACE from.8.to.0 AT Link0Out:
  PLACE from.8.to.4 AT Link1Out:
  PLACE from.8.to.6 AT Link2Out:
  PLACE from.8.to.7 AT Link3Out:
  PLACE from.0.to.8 AT Link0In:
  PLACE from.4.to.8 AT Link1In:
  PLACE from.6.to.8 AT Link2In:
  PLACE from.7.to.8 AT Link3In:
  }}}
  {{{  process calling
  process.8 (any)
  }}}
}}}
{{{  processor 4
PROCESSOR 4 T8
  {{{  channels placing
  PLACE from.6.to.2 AT Link0Out:
  PLACE from.6.to.5 AT Link2Out:
  PLACE from.6.to.8 AT Link1Out:
  PLACE from.2.to.6 AT Link0In:
  PLACE from.5.to.6 AT Link2In:
  PLACE from.8.to.6 AT Link1In:
  }}}
  {{{  process calling
  process.6 (any)
  }}}
}}}
{{{  processor 5
PROCESSOR 5 T8
  {{{  channels placing
  PLACE from.5.to.1 AT Link0Out:
  PLACE from.5.to.6 AT Link1Out:
  PLACE from.5.to.7 AT Link2Out:
  PLACE from.1.to.5 AT Link0In:
  PLACE from.6.to.5 AT Link1In:
  PLACE from.7.to.5 AT Link2In:
      }}}
      {{{  process calling
      process.5 (any)
      }}}
    }}}
    {{{  processor 6
    PROCESSOR 6 T8
      {{{  channels placing
      PLACE from.7.to.3 AT Link2Out:
      PLACE from.7.to.5 AT Link1Out:
      PLACE from.7.to.8 AT Link0Out:
      PLACE from.3.to.7 AT Link2In:
      PLACE from.5.to.7 AT Link1In:
      PLACE from.8.to.7 AT Link0In:
      }}}
      {{{  process calling
      process.7 (any)
      }}}
    }}}
    {{{  processor 7
    PROCESSOR 7 T8
      {{{  channels placing
      PLACE from.3.to.1 AT Link0Out:
      PLACE from.3.to.4 AT Link3Out:
      PLACE from.3.to.7 AT Link1Out:
      PLACE from.1.to.3 AT Link0In:
      PLACE from.4.to.3 AT Link3In:
      PLACE from.7.to.3 AT Link1In:
      }}}
      {{{  process calling
      process.3 (any)
      }}}
    }}}
  }}}
```

B-1-1-2 The host program:

```
  ...  EXTERNAL CHANNELS PROTOCOLS
  ...  SC process.0
  {{{  Links
  VAL Link0Out IS 0:
  VAL Link1Out IS 1:
  VAL Link2Out IS 2:
  VAL Link3Out IS 3:
  VAL Link0In  IS 4:
  VAL Link1In  IS 5:
  VAL Link2In  IS 6:
  VAL Link3In  IS 7:
  }}}
  {{{  external channels declaration
  CHAN OF ANY from.0.to.1:
  CHAN OF ANY from.0.to.8:
  CHAN OF ANY from.8.to.0:
  CHAN OF ANY from.1.to.0:
}}}
{{{  channels placing
PLACE from.0.to.1 AT Link2Out:
PLACE from.0.to.8 AT Link3Out:
PLACE from.1.to.0 AT Link2In:
PLACE from.8.to.0 AT Link3In:
}}}
{{{  program body
SEQ
  {{{  process calling
  process.0 (any)
  }}}
}}}
```

MESH

For simplicity all the channels have the same protocol (ANY), but any other protocol could be used.

C-1 The required part of the program source:

```
...   EXTERNAL CHANNELS PROTOCOLS
{{{   PROCESSES
{{{   SC process.0
{{{   COMMENT PROCESS process.0 EXTERNAL CHANNELS
CHAN OF ANY from.0.to.1 !:
CHAN OF ANY from.1.to.0 ?:
}}}
{{{   process.0
PROC process.0 (...)
  ...  process body
:
}}}
}}}
{{{   SC process.1
{{{   COMMENT PROCESS process.1 EXTERNAL CHANNELS
CHAN OF ANY from.1.to.0,
            from.1.to.2,
            from.1.to.3,
            from.1.to.6 !:
CHAN OF ANY from.0.to.1,
            from.2.to.1,
            from.3.to.1,
            from.6.to.1 ?:
}}}
{{{   process.1
{{{
PROC process.1 (...)
  ...  process body
:
}}}
}}}
}}}
{{{   SC process.2
{{{   COMMENT PROCESS process.2 EXTERNAL CHANNELS
CHAN OF ANY from.2.to.1,
            from.2.to.3,
            from.2.to.5,
            from.2.to.8 !:
CHAN OF ANY from.1.to.2,
            from.3.to.2,
            from.5.to.2,
            from.8.to.2 ?:
}}}
```

```
{{{  process.2
{{{
PROC process.2 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.3
{{{  COMMENT PROCESS process.3 EXTERNAL CHANNELS
CHAN OF ANY from.3.to.1,
            from.3.to.2,
            from.3.to.4,
            from.3.to.9 !:
CHAN OF ANY from.1.to.3,
            from.2.to.3,
            from.4.to.3,
            from.9.to.3 ?:
}}}
{{{  process.3
{{{
PROC process.3 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.4
{{{  COMMENT PROCESS process.4 EXTERNAL CHANNELS
CHAN OF ANY from.4.to.3,
            from.4.to.5,
            from.4.to.6,
            from.4.to.9 !:
CHAN OF ANY from.3.to.4,
            from.5.to.4,
            from.6.to.4,
            from.9.to.4 ?:
}}}
{{{  process.4
{{{
PROC process.4 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.5
{{{  COMMENT PROCESS process.5 EXTERNAL CHANNELS
CHAN OF ANY from.5.to.2,
            from.5.to.4,
            from.5.to.6,
            from.5.to.8 !:
CHAN OF ANY from.2.to.5,
            from.4.to.5,
            from.6.to.5,
            from.8.to.5 ?:
}}}
{{{  process.5
{{{
PROC process.5 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.6
{{{  COMMENT PROCESS process.6 EXTERNAL CHANNELS
CHAN OF ANY from.6.to.1,
            from.6.to.4,
            from.6.to.5,
            from.6.to.7 !:
CHAN OF ANY from.1.to.6,
            from.4.to.6,
            from.5.to.6,
            from.7.to.6 ?:
}}}
{{{  process.6
{{{
PROC process.6 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.7
{{{  COMMENT PROCESS process.7 EXTERNAL CHANNELS
CHAN OF ANY from.7.to.6,
            from.7.to.8,
            from.7.to.9 !:
CHAN OF ANY from.6.to.7,
            from.8.to.7,
            from.9.to.7 ?:
}}}
{{{  process.7
{{{
PROC process.7 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.8
{{{  COMMENT PROCESS process.8 EXTERNAL CHANNELS
CHAN OF ANY from.8.to.2,
            from.8.to.5,
            from.8.to.7,
            from.8.to.9 !:
CHAN OF ANY from.2.to.8,
                from.5.to.8,
                from.7.to.8,
                from.9.to.8 ?:
}}}
{{{  process.8
{{{
PROC process.8 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.9
{{{  COMMENT PROCESS process.9 EXTERNAL CHANNELS
CHAN OF ANY from.9.to.3,
            from.9.to.4,
            from.9.to.7,
            from.9.to.8 !:
CHAN OF ANY from.3.to.9,
            from.4.to.9,
            from.7.to.9,
            from.8.to.9 ?:
}}}
{{{  process.9
PROC process.9 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  CONFIGURE
PLACED PAR
  process.0 (any)
  process.1 (any)
  process.2 (any)
  process.3 (any)
  process.4 (any)
  process.5 (any)
  process.6 (any)
  process.7 (any)
  process.8 (any)
  process.9 (any)
}}}
```

C-1-1 The configurer results:

C-1-1-1 The configured program:

```
{{{  EXTERNAL CHANNELS PROTOCOLS
{{{  protocol declaration
PROTOCOL REAL.ARRAY IS INT::[]REAL64:
}}}
}}}
{{{  PROCESSES
```

```
...  SC process.1
...  SC process.2
...  SC process.3
...  SC process.4
...  SC process.5
...  SC process.6
...  SC process.7
...  SC process.8
...  SC process.9
}}}
{{{  Links
VAL Link0Out IS 0:
VAL Link1Out IS 1:
VAL Link2Out IS 2:
VAL Link3Out IS 3:
VAL Link0In  IS 4:
VAL Link1In  IS 5:
VAL Link2In  IS 6:
VAL Link3In  IS 7:
}}}
{{{  external channels declaration
CHAN OF ANY from.0.to.1:
CHAN OF ANY from.8.to.2:
CHAN OF ANY from.8.to.5:
CHAN OF ANY from.8.to.7:
CHAN OF ANY from.8.to.9:
CHAN OF ANY from.1.to.0:
CHAN OF ANY from.1.to.2:
CHAN OF ANY from.1.to.3:
CHAN OF ANY from.1.to.6:
CHAN OF ANY from.2.to.1:
CHAN OF ANY from.2.to.3:
CHAN OF ANY from.2.to.5:
CHAN OF ANY from.2.to.8:
CHAN OF ANY from.3.to.1:
CHAN OF ANY from.3.to.2:
CHAN OF ANY from.3.to.4:
CHAN OF ANY from.3.to.9:
CHAN OF ANY from.4.to.3:
CHAN OF ANY from.4.to.5:
CHAN OF ANY from.4.to.6:
CHAN OF ANY from.4.to.9:
CHAN OF ANY from.5.to.2:
CHAN OF ANY from.5.to.4:
CHAN OF ANY from.5.to.6:
CHAN OF ANY from.5.to.8:
CHAN OF ANY from.6.to.1:
CHAN OF ANY from.6.to.4:
CHAN OF ANY from.6.to.5:
CHAN OF ANY from.6.to.7:
CHAN OF ANY from.7.to.6:
CHAN OF ANY from.7.to.8:
CHAN OF ANY from.7.to.9:
CHAN OF ANY from.9.to.3:
```

```
CHAN OF ANY from.9.to.4:
CHAN OF ANY from.9.to.7:
CHAN OF ANY from.9.to.8:
}}}
{{{  program body
PLACED PAR
  {{{  processor 0
  PROCESSOR 0 T8
    {{{  channels placing
    PLACE from.1.to.0 AT Link1Out:
    PLACE from.1.to.2 AT Link2Out:
    PLACE from.1.to.3 AT Link0Out:
    PLACE from.1.to.6 AT Link3Out:
    PLACE from.0.to.1 AT Link1In:
    PLACE from.2.to.1 AT Link2In:
    PLACE from.3.to.1 AT Link0In:
    PLACE from.6.to.1 AT Link3In:
    }}}
    {{{  process calling
    process.1 (any)
    }}}
  }}}
  {{{  processor 1
  PROCESSOR 1 T8
    {{{  channels placing
    PLACE from.2.to.1 AT Link1Out:
    PLACE from.2.to.3 AT Link2Out:
    PLACE from.2.to.5 AT Link0Out:
    PLACE from.2.to.8 AT Link3Out:
    PLACE from.1.to.2 AT Link1In:
    PLACE from.3.to.2 AT Link2In:
    PLACE from.5.to.2 AT Link0In:
    PLACE from.8.to.2 AT Link3In:
    }}}
    {{{  process calling
    process.2 (any)
    }}}
  }}}
  {{{  processor 2
  PROCESSOR 2 T8
    {{{  channels placing
    PLACE from.3.to.1 AT Link0Out:
    PLACE from.3.to.2 AT Link1Out:
    PLACE from.3.to.4 AT Link2Out:
    PLACE from.3.to.9 AT Link3Out:
    PLACE from.1.to.3 AT Link0In:
    PLACE from.2.to.3 AT Link1In:
    PLACE from.4.to.3 AT Link2In:
    PLACE from.9.to.3 AT Link3In:
    }}}
    {{{  process calling
    process.3 (any)
    }}}
  }}}
```

```
{{{  processor 3
PROCESSOR 3 T8
  {{{  channels placing
  PLACE from.4.to.3 AT Link1Out:
  PLACE from.4.to.5 AT Link2Out:
  PLACE from.4.to.6 AT Link0Out:
  PLACE from.4.to.9 AT Link3Out:
  PLACE from.3.to.4 AT Link1In:
  PLACE from.5.to.4 AT Link2In:
  PLACE from.6.to.4 AT Link0In:
  PLACE from.9.to.4 AT Link3In:
  }}}
  {{{  process calling
  process.4 (any)
  }}}
}}}
{{{  processor 4
PROCESSOR 4 T8
  {{{  channels placing
  PLACE from.5.to.2 AT Link0Out:
  PLACE from.5.to.4 AT Link1Out:
  PLACE from.5.to.6 AT Link2Out:
  PLACE from.5.to.8 AT Link3Out:
  PLACE from.2.to.5 AT Link0In:
  PLACE from.4.to.5 AT Link1In:
  PLACE from.6.to.5 AT Link2In:
  PLACE from.8.to.5 AT Link3In:
  }}}
  {{{  process calling
  process.5 (any)
  }}}
}}}
{{{  processor 5
PROCESSOR 5 T8
  {{{  channels placing
  PLACE from.6.to.1 AT Link0Out:
  PLACE from.6.to.4 AT Link3Out:
  PLACE from.6.to.5 AT Link1Out:
  PLACE from.6.to.7 AT Link2Out:
  PLACE from.1.to.6 AT Link0In:
  PLACE from.4.to.6 AT Link3In:
  PLACE from.5.to.6 AT Link1In:
  PLACE from.7.to.6 AT Link2In:
  }}}
  {{{  process calling
  process.6 (any)
  }}}
}}}
{{{  processor 6
PROCESSOR 6 T8
  {{{  channels placing
  PLACE from.7.to.6 AT Link1Out:
  PLACE from.7.to.8 AT Link2Out:
  PLACE from.7.to.9 AT Link0Out:
      PLACE from.6.to.7 AT Link1In:
      PLACE from.8.to.7 AT Link2In:
      PLACE from.9.to.7 AT Link0In:
      }}}
      {{{  process calling
      process.7 (any)
      }}}
    }}}
    {{{  processor 7
    PROCESSOR 7 T8
      {{{  channels placing
      PLACE from.8.to.2 AT Link0Out:
      PLACE from.8.to.5 AT Link3Out:
      PLACE from.8.to.7 AT Link1Out:
      PLACE from.8.to.9 AT Link2Out:
      PLACE from.2.to.8 AT Link0In:
      PLACE from.5.to.8 AT Link3In:
      PLACE from.7.to.8 AT Link1In:
      PLACE from.9.to.8 AT Link2In:
      }}}
      {{{  process calling
      process.8 (any)
      }}}
    }}}
    {{{  processor 8
    PROCESSOR 8 T8
      {{{  channels placing
      PLACE from.9.to.3 AT Link0Out:
      PLACE from.9.to.4 AT Link3Out:
      PLACE from.9.to.7 AT Link2Out: -- pipe tail
      PLACE from.9.to.8 AT Link1Out:
      PLACE from.3.to.9 AT Link0In:
      PLACE from.4.to.9 AT Link3In:
      PLACE from.7.to.9 AT Link2In:  -- pipe tail
      PLACE from.8.to.9 AT Link1In:
      }}}
      {{{  process calling
      process.9 (any)
      }}}
    }}}
  }}}
```

C-1-1-2 The host program:

```
  ...  EXTERNAL CHANNELS PROTOCOLS
  ...  SC process.0
  {{{  Links
  VAL Link0Out IS 0:
  VAL Link1Out IS 1:
  VAL Link2Out IS 2:
  VAL Link3Out IS 3:
  VAL Link0In  IS 4:
  VAL Link1In  IS 5:
  VAL Link2In  IS 6:
```

```
VAL Link3In  IS 7:
}}}
{{{  external channels declaration
CHAN OF ANY from.0.to.1:
CHAN OF ANY from.1.to.0:
}}}
{{{  channels placing
PLACE from.0.to.1 AT Link2Out:
PLACE from.1.to.0 AT Link2In:
}}}
{{{  program body
SEQ
  {{{  process calling
  process.0 (any)
  }}}
}}}
```

BINARY TREE

For simplicity all the channels have the same protocol (ANY), but any other protocol could be used.

D-1 The required part of the program source:

```
...  EXTERNAL CHANNELS PROTOCOLS
{{{  PROCESSES
{{{  SC process.0
{{{  COMMENT PROCESS process.0 EXTERNAL CHANNELS
CHAN OF ANY from.0.to.1 !:
CHAN OF ANY from.1.to.0 ?:
}}}
{{{  process.0
PROC process.0 (...)
  ...  process body
:
}}}
}}}
{{{  SC process.1
{{{  COMMENT PROCESS process.1 EXTERNAL CHANNELS
CHAN OF ANY from.1.to.0,
            from.1.to.2,
            from.1.to.3 !:
CHAN OF ANY from.0.to.1,
            from.2.to.1,
            from.3.to.1 ?:
}}}
{{{  process.1
{{{
PROC process.1 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.2
{{{  COMMENT PROCESS process.2 EXTERNAL CHANNELS
CHAN OF ANY from.2.to.1,
            from.2.to.4,
            from.2.to.5 !:
CHAN OF ANY from.1.to.2,
            from.4.to.2,
            from.5.to.2 ?:
}}}
{{{  process.2
{{{
PROC process.2 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.3
{{{  COMMENT PROCESS process.3 EXTERNAL CHANNELS
CHAN OF ANY from.3.to.1,
            from.3.to.6,
            from.3.to.7 !:
CHAN OF ANY from.1.to.3,
            from.6.to.3,
            from.7.to.3 ?:
}}}
{{{  process.3
{{{
PROC process.3 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.4
{{{  COMMENT PROCESS process.4 EXTERNAL CHANNELS
CHAN OF ANY from.4.to.2,
            from.4.to.8,
            from.4.to.9 ':
CHAN OF ANY from.2.to.4,
            from.8.to.4,
            from.9.to.4 ?:
}}}
{{{  process.4
{{{
PROC process.4 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.5
{{{  COMMENT PROCESS process.5 EXTERNAL CHANNELS
CHAN OF ANY from.5.to.2 !:
CHAN OF ANY from.2.to.5 ?:
}}}
{{{  process.5
{{{
PROC process.5 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.6
{{{  COMMENT PROCESS process.6 EXTERNAL CHANNELS
CHAN OF ANY from.6.to.3 !:
```

```
CHAN OF ANY from.3.to.6 ?:
}}}
{{{  process.6
{{{
PROC process.6 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.7
{{{  COMMENT PROCESS process.7 EXTERNAL CHANNELS
CHAN OF ANY from.7.to.3 !:
CHAN OF ANY from.3.to.7 ?:
}}}
{{{  process.7
{{{
PROC process.7 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.8
{{{  COMMENT PROCESS process.8 EXTERNAL CHANNELS
CHAN OF ANY from.8.to.4 !:
CHAN OF ANY from.4.to.8 ?:
}}}
{{{  process.8
{{{
PROC process.8 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.9
{{{  COMMENT PROCESS process.9 EXTERNAL CHANNELS
CHAN OF ANY from.9.to.4 !:
CHAN OF ANY from.4.to.9 ?:
}}}
{{{  process.9
PROC process.9 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  CONFIGURE
PLACED PAR
  process.0 (any)
  process.1 (any)
  process.2 (any)
  process.3 (any)
  process.4 (any)
  process.5 (any)
  process.6 (any)
  process.7 (any)
  process.8 (any)
  process.9 (any)
}}}
```

D-1-1 The configurer results:

D-1-1-2 The configured program:

```
...  EXTERNAL CHANNELS PROTOCOLS
{{{  PROCESSES
...  SC process.1
...  SC process.2
...  SC process.3
...  SC process.4
...  SC process.5
...  SC process.6
...  SC process.7
...  SC process.8
...  SC process.9
}}}
{{{  Links
VAL Link0Out IS 0:
VAL Link1Out IS 1:
VAL Link2Out IS 2:
VAL Link3Out IS 3:
VAL Link0In  IS 4:
VAL Link1In  IS 5:
VAL Link2In  IS 6:
VAL Link3In  IS 7:
}}}
{{{  external channels declaration
CHAN OF ANY from.0.to.1:
CHAN OF ANY from.1.to.0:
CHAN OF ANY from.1.to.2:
CHAN OF ANY from.1.to.3:
CHAN OF ANY from.2.to.1:
CHAN OF ANY from.2.to.4:
CHAN OF ANY from.2.to.5:
CHAN OF ANY from.3.to.1:
CHAN OF ANY from.3.to.6:
CHAN OF ANY from.3.to.7:
CHAN OF ANY from.4.to.2:
CHAN OF ANY from.4.to.8:
CHAN OF ANY from.4.to.9:
CHAN OF ANY from.5.to.2:
CHAN OF ANY from.6.to.3:
CHAN OF ANY from.7.to.3:
CHAN OF ANY from.8.to.4:
CHAN OF ANY from.9.to.4:
}}}
```

```
{{{  program body
PLACED PAR
  {{{  processor 0
  PROCESSOR 0 T8
    {{{  channels placing
    PLACE from.1.to.0 AT Link1Out:
    PLACE from.1.to.2 AT Link2Out:
    PLACE from.1.to.3 AT Link0Out:
    PLACE from.0.to.1 AT Link1In:
    PLACE from.2.to.1 AT Link2In:
    PLACE from.3.to.1 AT Link0In:
    }}}
    {{{  process calling
    process.1 (any)
    }}}
  }}}
  {{{  processor 1
  PROCESSOR 1 T8
    {{{  channels placing
    PLACE from.2.to.1 AT Link1Out:
    PLACE from.2.to.4 AT Link2Out:
    PLACE from.2.to.5 AT Link0Out:
    PLACE from.1.to.2 AT Link1In:
    PLACE from.4.to.2 AT Link2In:
    PLACE from.5.to.2 AT Link0In:
    }}}
    {{{  process calling
    process.2 (any)
    }}}
  }}}
  {{{  processor 2
  PROCESSOR 2 T8
    {{{  channels placing
    PLACE from.4.to.2 AT Link1Out:
    PLACE from.4.to.8 AT Link2Out:
    PLACE from.4.to.9 AT Link0Out:
    PLACE from.2.to.4 AT Link1In:
    PLACE from.8.to.4 AT Link2In:
    PLACE from.9.to.4 AT Link0In:
    }}}
    {{{  process calling
    process.4 (any)
    }}}
  }}}
  {{{  processor 3
  PROCESSOR 3 T8
    {{{  channels placing
    PLACE from.8.to.4 AT Link1Out:
    PLACE from.4.to.8 AT Link1In:
    }}}
    {{{  process calling
    process.8 (any)
    }}}
  }}}
```

```
{{{  processor 4
PROCESSOR 4 T8
  {{{  channels placing
  PLACE from.9.to.4 AT Link0Out:
  PLACE from.4.to.9 AT Link0In:
  }}}
  {{{  process calling
  process.9 (any)
  }}}
}}}
{{{  processor 5
PROCESSOR 5 T8
  {{{  channels placing
  PLACE from.5.to.2 AT Link0Out:
  PLACE from.2.to.5 AT Link0In:
  }}}
  {{{  process calling
  process.5 (any)
  }}}
}}}
{{{  processor 6
PROCESSOR 6 T8
  {{{  channels placing
  PLACE from.3.to.1 AT Link0Out:
  PLACE from.3.to.6 AT Link2Out:
  PLACE from.3.to.7 AT Link3Out:
  PLACE from.1.to.3 AT Link0In:
  PLACE from.6.to.3 AT Link2In:
  PLACE from.7.to.3 AT Link3In:
  }}}
  {{{  process calling
  process.3 (any)
  }}}
}}}
{{{  processor 7
PROCESSOR 7 T8
  {{{  channels placing
  PLACE from.6.to.3 AT Link1Out:
  PLACE from.3.to.6 AT Link1In:
  }}}
  {{{  process calling
  process.6 (any)
  }}}
}}}
{{{  processor 8
PROCESSOR 8 T8
  {{{  channels placing
  PLACE from.7.to.3 AT Link0Out:
  PLACE from.3.to.7 AT Link0In:
  }}}
  {{{  process calling
  process.7 (any)
  }}}
}}}
```

```
}}}
```

D-1-1-2 The host program:

```
...  EXTERNAL CHANNELS PROTOCOLS
...  SC process.0
{{{  Links
VAL Link0Out IS 0:
VAL Link1Out IS 1:
VAL Link2Out IS 2:
VAL Link3Out IS 3:
VAL Link0In  IS 4:
VAL Link1In  IS 5:
VAL Link2In  IS 6:
VAL Link3In  IS 7:
}}}
{{{  external channels declaration
CHAN OF ANY from.0.to.1:
CHAN OF ANY from.1.to.0:
}}}
{{{  channels placing
PLACE from.0.to.1 AT Link2Out:
PLACE from.1.to.0 AT Link2In:
}}}
{{{  program body
SEQ
  {{{  process calling
  process.0 (any)
  }}}
}}}
```

USER DEFINED

For simplicıty all the channels have the same protocol (ANY), but any other protocol could be used. **E-1 The required part of the program source:**

```
...  EXTERNAL CHANNELS PROTOCOLS
{{{  PROCESSES
{{{  SC process.0
{{{  COMMENT PROCESS process.0 EXTERNAL CHANNELS
CHAN OF ANY from.0.to.1,
            from.0.to.6 !:
CHAN OF ANY from.1.to.0,
            from.6.to.0 ?:
}}}
{{{  process.0
PROC process.0 (...)
  ...  process body
:
}}}
}}}
{{{  SC process.1
{{{  COMMENT PROCESS process.1 EXTERNAL CHANNELS
CHAN OF ANY from.1.to.0,
            from.1.to.2,
            from.1.to.4,
            from.1.to.5 !:
CHAN OF ANY from.0.to.1,
            from.2.to.1,
            from.4.to.1,
            from.5.to.1 ?:
}}}
{{{  process.1
{{{
PROC process.1 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.2
{{{  COMMENT PROCESS process.2 EXTERNAL CHANNELS
CHAN OF ANY from.2.to.1,
            from.2.to.3,
            from.2.to.5 !:
CHAN OF ANY from.1.to.2,
            from.3.to.2,
            from.5.to.2 ?:
}}}
```

```
{{{  process.2
{{{
PROC process.2 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.3
{{{  COMMENT PROCESS process.3 EXTERNAL CHANNELS
CHAN OF ANY from.3.to.2,
            from.3.to.4,
            from.3.to.5 !:
CHAN OF ANY from.2.to.3,
            from.4.to.3,
            from.5.to.3 ?:
}}}
{{{  process.3
{{{
PROC process.3 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.4
{{{  COMMENT PROCESS process.4 EXTERNAL CHANNELS
CHAN OF ANY from.4.to.1,
            from.4.to.3,
            from.4.to.5 !:
CHAN OF ANY from.1.to.4,
            from.3.to.4,
            from.5.to.4 ?:
}}}
{{{  process.4
{{{
PROC process.4 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.5
{{{  COMMENT PROCESS process.5 EXTERNAL CHANNELS
CHAN OF ANY from.5.to.1,
            from.5.to.2,
            from.5.to.3,
            from.5.to.4 !:
CHAN OF ANY from.1.to.5,
            from.2.to.5,
            from.3.to.5,
            from.4.to.5 ?:
}}}
{{{  process.5
```

```
{{{
PROC process.5 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.6
{{{  COMMENT PROCESS process.6 EXTERNAL CHANNELS
CHAN OF ANY from.6.to.0,
            from.6.to.7,
            from.6.to.8,
            from.6.to.9 !:
CHAN OF ANY from.0.to.6,
            from.7.to.6,
            from.8.to.6,
            from.9.to.6 ?:
}}}
{{{  process.6
{{{
PROC process.6 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.7
{{{  COMMENT PROCESS process.7 EXTERNAL CHANNELS
CHAN OF ANY from.7.to.6,
            from.7.to.8,
            from.7.to.9 !:
CHAN OF ANY from.6.to.7,
            from.8.to.7,
            from.9.to.7 ?:
}}}
{{{  process.7
{{{
PROC process.7 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.8
{{{  COMMENT PROCESS process.8 EXTERNAL CHANNELS
CHAN OF ANY from.8.to.6,
            from.8.to.7,
            from.8.to.9 !:
CHAN OF ANY from.6.to.8,
            from.7.to.8,
            from.9.to.8 ?:
}}}
{{{  process.8
{{{
```

```
PROC process.8 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  SC process.9
{{{  COMMENT PROCESS process.9 EXTERNAL CHANNELS
CHAN OF ANY from.9.to.6,
            from.9.to.7,
            from.9.to.8 !:
CHAN OF ANY from.6.to.9,
            from.7.to.9,
            from.8.to.9 ?:
}}}
{{{  process.9
PROC process.9 (...)
  ...  process body
:
}}}
}}}
}}}
{{{  CONFIGURE
PLACED PAR
  process.0 (any)
  process.1 (any)
  process.2 (any)
  process.3 (any)
  process.4 (any)
  process.5 (any)
  process.6 (any)
  process.7 (any)
  process.8 (any)
  process.9 (any)
}}}
```

E-1-1 The configurer results:

E-1-1-1 The configured program:

```
    {{{  EXTERNAL CHANNELS PROTOCOLS
    {{{  protocol declaration
    PROTOCOL REAL.ARRAY IS INT::[]REAL64:
    }}}
    }}}
    {{{  PROCESSES
    ...  SC process.1
    ...  SC process.2
    ...  SC process.3
    ...  SC process.4
    ...  SC process.5
    ...  SC process.6
    ...  SC process.7
    ...  SC process.8
```

```
...  SC process.9
}}}
{{{  Links
VAL Link0Out IS 0:
VAL Link1Out IS 1:
VAL Link2Out IS 2:
VAL Link3Out IS 3:
VAL Link0In  IS 4:
VAL Link1In  IS 5:
VAL Link2In  IS 6:
VAL Link3In  IS 7:
}}}
{{{  external channels declaration
CHAN OF ANY from.0.to.1:
CHAN OF ANY from.0.to.6:
CHAN OF ANY from.1.to.0:
CHAN OF ANY from.1.to.2:
CHAN OF ANY from.1.to.4:
CHAN OF ANY from.1.to.5:
CHAN OF ANY from.2.to.1:
CHAN OF ANY from.2.to.3:
CHAN OF ANY from.2.to.5:
CHAN OF ANY from.3.to.2:
CHAN OF ANY from.3.to.4:
CHAN OF ANY from.3.to.5:
CHAN OF ANY from.4.to.1:
CHAN OF ANY from.4.to.3:
CHAN OF ANY from.4.to.5:
CHAN OF ANY from.5.to.1:
CHAN OF ANY from.5.to.2:
CHAN OF ANY from.5.to.3:
CHAN OF ANY from.5.to.4:
CHAN OF ANY from.6.to.0:
CHAN OF ANY from.6.to.7:
CHAN OF ANY from.6.to.8:
CHAN OF ANY from.6.to.9:
CHAN OF ANY from.7.to.6:
CHAN OF ANY from.7.to.8:
CHAN OF ANY from.7.to.9:
CHAN OF ANY from.8.to.6:
CHAN OF ANY from.8.to.7:
CHAN OF ANY from.8.to.9:
CHAN OF ANY from.9.to.6:
CHAN OF ANY from.9.to.7:
CHAN OF ANY from.9.to.8:
CHAN OF ANY JUST.FOR.LOADING: -- added for loading }}}
{{{  program body
PLACED PAR
  {{{  processor 0
  PROCESSOR 0 T8
    {{{  channels placing
    PLACE from.1.to.0 AT Link1Out:
    PLACE from.1.to.2 AT Link0Out:
    PLACE from.1.to.4 AT Link3Out:
```

```
  PLACE from.1.to.5 AT Link2Out:
  PLACE from.0.to.1 AT Link1In:
  PLACE from.2.to.1 AT Link0In:
  PLACE from.4.to.1 AT Link3In:
  PLACE from.5.to.1 AT Link2In:
  }}}
  {{{  process calling
  process.1 (any)
  }}}
}}}
{{{  processor 1
PROCESSOR 1 T8
  {{{  channels placing
  PLACE from.5.to.1 AT Link1Out:
  PLACE from.5.to.2 AT Link2Out:
  PLACE from.5.to.3 AT Link0Out:
  PLACE from.5.to.4 AT Link3Out:
  PLACE from.1.to.5 AT Link1In:
  PLACE from.2.to.5 AT Link2In:
  PLACE from.3.to.5 AT Link0In:
  PLACE from.4.to.5 AT Link3In:
  }}}
  {{{  process calling
  process.5 (any)
  }}}
}}}
{{{  processor 2
PROCESSOR 2 T8
  {{{  channels placing
  PLACE from.2.to.1 AT Link0Out:
  PLACE from.2.to.3 AT Link2Out:
  PLACE from.2.to.5 AT Link1Out:
  PLACE from.1.to.2 AT Link0In:
  PLACE from.3.to.2 AT Link2In:
  PLACE from.5.to.2 AT Link1In:
  }}}
  {{{  process calling
  process.2 (any)
  }}}
}}}
{{{  processor 3
PROCESSOR 3 T8
  {{{  channels placing
  PLACE from.3.to.2 AT Link1Out:
  PLACE from.3.to.4 AT Link2Out:
  PLACE from.3.to.5 AT Link0Out:
  PLACE from.2.to.3 AT Link1In:
  PLACE from.4.to.3 AT Link2In:
  PLACE from.5.to.3 AT Link0In:
  }}}
  {{{  process calling
  process.3 (any)
  }}}
}}}
```

```
{{{  processor 4
PROCESSOR 4 T8
  {{{  channels placing
  PLACE from.4.to.1 AT Link0Out:
  PLACE from.4.to.3 AT Link1Out:
  PLACE from.4.to.5 AT Link3Out:
  PLACE JUST.FOR.LOADING AT Link2Out:
  PLACE from.1.to.4 AT Link0In:
  PLACE from.3.to.4 AT Link1In:
  PLACE from.5.to.4 AT Link3In:
  }}}
  {{{  process calling
  process.4 (any)
  }}}
}}}
{{{  processor 5
PROCESSOR 5 T8
  {{{  channels placing
  PLACE from.7.to.6 AT Link2Out:
  PLACE from.7.to.8 AT Link0Out:
  PLACE from.7.to.9 AT Link3Out:
  PLACE from.6.to.7 AT Link2In:
  PLACE from.8.to.7 AT Link0In:
  PLACE from.9.to.7 AT Link3In:
  PLACE JUST.FOR.LOADING AT Link1In:
  }}}
  {{{  process calling
  process.7 (any)
  }}}
}}}
{{{  processor 6
PROCESSOR 6 T8
  {{{  channels placing
  PLACE from.6.to.0 AT Link0Out:
  PLACE from.6.to.7 AT Link1Out:
  PLACE from.6.to.8 AT Link2Out:
  PLACE from.6.to.9 AT Link3Out:
  PLACE from.0.to.6 AT Link0In:
  PLACE from.7.to.6 AT Link1In:
  PLACE from.8.to.6 AT Link2In:
  PLACE from.9.to.6 AT Link3In:
  }}}
  {{{  process calling
  process.6 (any)
  }}}
}}}
{{{  processor 7
PROCESSOR 7 T8
  {{{  channels placing
  PLACE from.8.to.6 AT Link1Out:
  PLACE from.8.to.7 AT Link0Out:
  PLACE from.8.to.9 AT Link2Out:
  PLACE from.6.to.8 AT Link1In:
  PLACE from.7.to.8 AT Link0In:
      PLACE from.9.to.8 AT Link2In:
      }}}
      {{{  process calling
      process.8 (any)
      }}}
    }}}
    {{{  processor 8
    PROCESSOR 8 T8
      {{{  channels placing
      PLACE from.9.to.6 AT Link3Out:
      PLACE from.9.to.7 AT Link0Out:
      PLACE from.9.to.8 AT Link1Out:
      PLACE from.6.to.9 AT Link3In:
      PLACE from.7.to.9 AT Link0In:
      PLACE from.8.to.9 AT Link1In:
      }}}
      {{{  process calling
      process.9 (any)
      }}}
    }}}
  }}}
```

E-1-1-2 The host program:

```
  ...  EXTERNAL CHANNELS PROTOCOLS
  ...  SC process.0
  {{{  Links
  VAL Link0Out IS 0:
  VAL Link1Out IS 1:
  VAL Link2Out IS 2:
  VAL Link3Out IS 3:
  VAL Link0In  IS 4:
  VAL Link1In  IS 5:
  VAL Link2In  IS 6:
  VAL Link3In  IS 7:
  }}}
  {{{  external channels declaration
  CHAN OF ANY from.0.to.1:
  CHAN OF ANY from.0.to.6:
  CHAN OF ANY from.1.to.0:
  CHAN OF ANY from.6.to.0:
  }}}
  {{{  channels placing
  PLACE from.0.to.1 AT Link2Out:
  PLACE from.0.to.6 AT Link3Out:
  PLACE from.1.to.0 AT Link2In:
  PLACE from.6.to.0 AT Link3In:
  }}}
  {{{  program body
  SEQ
    {{{  process calling
    process.0 (any)
    }}}
  }}}
```