# NETWORK CONTROL FOR A MULTI-USER TRANSPUTER-BASED SYSTEM

Aurona J. Gerber

Number 89-11448/G

Promoter : Prof. A.J. Walker
Department Electrical Engineering
WITS

Assistant Promoter : Prof. A.L. Du Plesses
Department Computer Science
UNISA

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering

Pretoria, 1991

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____
( signature of candidate )

_____ day of _____ 1991

**Abstract**

The $MC^2/64$ system is a configureable multi-user transputer- based system which was designed using a modular approach. The $MC^2/64$ consists of $MC^2$ Clusters which are connected using a modified Clos network. The $MC^2$ Clusters were designed and realised as completely configurable modules using and extending an algorithm based on Eulerian cycles through a requested graph. This dissertation discusses the configuration algorithm and the extensions made to the algorithm for the $MC^2$ Clusters.

The total $MC^2/64$ system is not completely configurable as a $MC^2$ Cluster releases only a limited number of links for inter-cluster connections. This dissertation analyses the configurability of $MC^2/64$, but also presents algorithms which enhance the usability of the system from the user's point of view.

The design and the implementation of the network control software are also submitted as topics in this dissertation. The network control software must allow multiple users to use the system, but without them influencing each other's transputer domains.

This dissertation therefore seeks to give an overview of network control problems and the solutions implemented in current $MC^2/64$ systems. The results of the research done for this dissertation will hopefully aid in the design of future $MC^2$ systems which will provide South Africa with much needed, low cost, high performance computing power.

dedication

To JC

To Marthinus Gerber

**Acknowledgements**

Prof. A.J. Walker
Prof. A.L. du Plessis

The $MC^2$ Project Team

especially
*Niel Viljoen*
the original Project Leader of $MC^2$

and the
*CSIR*
for providing the opportunity for an MSc on $MC^2$

Special thanks to
K.A.R. van Oudtshoorn
who did the proof reading of this dissertation

## CONTENTS

## FIGURES

# 1 INTRODUCTION

## 1.1 Overview

We live in a world which is changing so rapidly that there is no parallel in the history of the earth. Most of these changes are technological changes and the demand for computing power is growing exponentially with the rate of technological change. Computers are responsible for most of the hard work that is necessary to bring new design ideas and the technological wonders of our age into existence. Computers are the work horses doing the calculations, simulations and documentation (with the aid of the programmer!). Computers provide the necessary design and development tools for each stage of the development of a new product. If we could enhance computer efficiency and computer power results would be achieved faster. But this means that the demand for computing power is growing so rapidly (and with that all the possible solutions for this demand) that it is becoming a very complex field to work in. This dissertation endeavours to give an overview of the current status of high performance computer technology and to place the work done by the $MC^2$ project team of the CSIR into perspective, as well as to explain the design issues and the development work of the author, especially concerning network control in the implementation of a massively parallel computer.

The ever-increasing demand for microcomputing power is being satisfied at this stage by two areas of technological development: high-speed RISC technology on the one hand, and parallel processing on the other. RISC technology offers the promise of microcomputer systems that approach today's supercomputers in performance, yet it requires little change in the way in which current software is being developed. But this technology is reaching its limits, both in speed and in density. To get more performance out of RISC is also becoming expensive. Parallel processing, however, offers the promise of systems of which the computing power is limited only by the resources of the system designer and the ingenuity of the programmer! Intuitively programmers and system developers have always felt that

parallel processing is a technology that can satisfy computing power demands - costing much less, and with the explosion that has taken place in this field following the announcement of the Inmos transputer, they are being proven correct.

South Africa has also acquired the need for supercomputer performance in several areas. We have (had) the additional problem of international sanctions, as well as an increasingly unfavourable exchange rate that makes it difficult to obtain supercomputers. It would benefit this country tremendously if supercomputer power and the necessary technological expertise to develop, enhance and support these computers, could be developed locally.

The $MC^2$ project was started in April 1983 at the CSIR with two purposes in mind:

* The creation of a computer that could supply South Africa with a machine capable of performance levels equal to a CRAY type machine

* Creating a scalable computer, one that could address the computational requirements from PC level through to mainframes and supercomputers.

The only way by which the above mentioned requirements can be met, is by using modular parallelism. In the past such systems failed because the traditional microprocessors were not suitable for parallel processing implementations. Fortunately Inmos introduced the transputer, a microprocessor which was designed specifically for parallel implementations using high-speed communication channels and which is ideally suitable for a design such as $MC^2$. In the end the purpose of $MC^2$ is a multi-user parallel supercomputer using the transputer and based on the concept of configurable nodes, but with smaller spin-off products along the line which could also be used to address smaller computational needs.

## 1.2 Glossary of terms

| | |
|---|---|
| CEs/CE | Computing Element(s) |
| Configurable | The concept by which is meant that all network configurations can always be switched. |
| CPU | Central Processing Unit |
| DM | Domain Manager. The memory resident program on an IOP by which a user gets access to $MC^2/64$. |
| DMS | Domain Management Server. This is the software on $MC^2/64$ responsible for all the user control of domains. It is, in fact, the operating system of $MC^2/64$. It runs on the Helios backbone consisting of the system controller and the four cluster controllers. |
| IOP | IO Processor. For the $MC^2/64$ this will be an IBM or compatible PC. |
| Domain | A set of linked transputers and resources. |
| System Console | The computer acting as interface between the System Administrator and the Helios backbone. |
| Domain Management Services | A set of commands provided by the system software enabling the user to control his domain. |
| External Resources | Resources not included in the Helios backbone,as a high density screen. |

| | |
|---|---|
| Helios Backbone | The pipeline of transputers running Helios and the system software. These transputers control the parent domain of $MC^2/64$. |
| Parent Domain | The transputer pool and the external resources. |
| DMS | Domain Management Server. This is the operating system of $MC^2/64$ running under Helios on the system controller. |
| GFlops | Giga Floating-Point Operations Per Second. (=1000 MFlops) |
| Inmos T800 Transputer | 25 MHz processor, 12.5 MIPS and 1.5 MFlops peak performance. From 1 to 32 Mbytes of field upgradable memory. |
| Intel i860 | 40 MHz processor, 40 MIPS and 80 MFlops peak performance. From 4 to 32 Mbytes of field upgradeable memory. |
| Inter-cluster Switch | The network that connects the clusters to form the $MC^2/64$ and $MC^2/256$ systems. |
| Mbyte | Mega Byte |
| $MC^2$ | Massively Concurrent Computer |
| $MC^2/64$ | The $MC^2$ multi-user, 64 processor, configurable supercomputer. |
| $MC^2$ Clusters ($MC^2$ Maxi Clusters) | These are the 16-transputer units using two 48X48 crossbar switches to provide complete configurability and reconfigurability. |

| | |
|---|---|
| $MC^2$ Mini Clusters | An 8-transputer unit providing complete reconfigurability. |
| MFlops | Millions of Floating-Point Operations Per Second |
| MIMD | Multiple Instruction / Multiple Data Streams |
| MIPS | Millions of Instructions Per Second |
| PEs/PE | Processing Element(s) |
| Reconfigurable | The concept by which is meant that the network configuration can be changed. $MC^2/64$ is fully reconfigurable. |
| SDLC | Software Development Life Cycle |
| Sun Microsystems SPARC | 20 MHz processor, 12.5 MIPS and 3.6 MFlops peak performance. 33 MHz processor, 40 MIPS and 6.2 MFlops peak performance. |
| Worker Cards | These are the base units of all $MC^2$ machines and consist of a T800/T425 and 1 to 8 MBytes of DRAM. |

## 1.3 Network control for a multi-user transputer-based system

The $MC^2/64$ is a configurable, distributed parallel machine. Configurable means that the way in which the communication channels of the processors are connected, can be selected and changed. A distributed parallel machine is a machine consisting of processors each with its own memory which is not shared in any way. The processors communicate through high-speed communication links.

Users gain access to the machine by being allocated a certain domain of transputers. This domain of transputers can be seen as the user's personal transputer system which can be switched, controlled and used as desired by the user, without affecting another user of the system. This also means that there is no degradation in performance of the machine when more users use the system, as opposed to the traditional sequential machines. In traditional sequential machines a single CPU had to be shared amongst the different users. This was realised by allocating a time-slot to each user. The more users, the slower the system owing to both the number of users and the additional overhead the accommodation of each user generates. The systems therefore showed a considerable degradation in performance with each user logging into the system.

The network control of the $MC^2/64$ machine presented very interesting and challenging problems. The architecture of the $MC^2/64$ is unique and the control of the network is therefore also unique. $MC^2/64$ is also a distributed parallel machine. Such parallel machines are traditionally arranged in fixed configurations resulting in an immense increase in software complexity when the software fully exploits the hardware architecture. We chose to make $MC^2/64$ configurable to provide greater flexibility, especially as the hardware can be changed or configured to match the software requirements of a specific problem. Algorithms controlling the network must therefore ensure complete configurability, connecting any transputer in the system to any other transputer regardless of connections previously made. Moreover, the algorithms controlling the network must be fast and efficient, and the operating system or control software of the system must be user friendly.

This dissertation will take on the following form :

Chapter 1 : Introduction.

This chapter will also include an introduction into the state of supercomputers (especially parallel supercomputers) in the world today as well as a glossary of terms used.

Chapter 2 : A description of the $MC^2/64$ project.

This description includes the history, a description of the transputer and Occam, and a description of the $MC^2$ architecture.

Chapter 3 : $MC^2$ Cluster configuration algorithm.

This chapter will focus on the design of the $MC^2$ Cluster as a module of the $MC^2/64$. The $MC^2$ must therefore also be totally configurable. One of the design principles of $MC^2$ is also the production of smaller spin-off products for smaller computational needs, and the $MC^2$ Cluster must be designed as such.

Chapter 4 : $MC^2/64$ configuration.

This chapter will focus on the $MC^2/64$ system as a multi-user system that must be fully configurable. Any user must be able to log on at any time and switch his requested network.

Chapter 5 : $MC^2/64$ system software design.

This chapter will describe the software design techniques that were used to design the Version 1 $MC^2/64$ system software incorporating the implementation of the configuration algorithms.

Chapter 6 : Conclusions.

## 1.4 History of parallel supercomputing

The term *"supercomputer"* has been used since about 1970 to describe the most powerful machines of their generation. The term *computer power* is multidimensional but usually includes processing speed, accuracy, memory size and memory speed. A supercomputer's performance allows it to perform computing jobs that would be impractical or impossible on less powerful machines. Such problems include certain simulations in aerodynamics or fluid dynamics.

To meet computing power needs, new system architectures have evolved continuously which offer supercomputing performance or near-supercomputing performance. The overall architecture of the first generation of computers is described as serial and is usually referred to simply as the Von Neumann organisation. Such computers comprise an input and output (IO) device, a single memory for storing both data and instructions, a single control unit (CU) for interpreting the instructions and a single CPU (Central Processing Unit). Each operation of the computer had to be performed sequentially. Today supercomputer power is made accessible mainly through some form of parallelism. Parallelism refers to the ability to perform these tasks simultaneously.

Three kinds of supercomputers can be distinguished today, single-processor machines, minimally parallel machines with a few processors, and more recently, the highly parallel machines. An example of the first type is the CRAY-1. The CRAY X-MP is a minimally parallel machine with a few processors. The Connection Machine (from Thinking Machines Inc.) and the Megaframe Supercluster (from West Germany's Parsytec GmbH) are highly parallel machines. The $MC^2$ can also be distinguished as a highly parallel machine.

A new class of machine, the mini-supercomputer has also been defined recently. Mini-supers fill the gap between full-scale supercomputers such as those made by Cray Research, and super-minicomputers such as Digital Equipment Corporation's

VAX machines. Mini-supers usually process between 10 and 100 million floating-point operations per second (MFlops). They cost between $100 000 and $1.5 million, compared to the price of $2 million to $25 million for a supercomputer.[1]

The latest supercomputers and mini-supers of today have taken different paths to high performance. They have been designed with radically different architectures, usually employing some kind of parallel processing or multiprocessing. This is mainly because VLSI technology is reaching its limits and the fact that, the more powerful the processor, the more expensive it becomes. By employing a large number of cheaper and less powerful processors operating simultaneously, that which the computer power scientists and engineers have been seeking is achieved - supercomputer power without the huge expense of owning or sharing a supercomputer.

The downside of parallel processing is that it is much harder to write applications software that fully exploits the power of the machines than with traditional sequential machines. It is often very difficult for programmers to visualize the effective decomposition of an application into node-sized portions. Once a program has been written, debugging it is far more complex than for standard sequential code. The whole concept of parallel processing or parallel machines loses its advantage over sequential machines if the application software does not take full advantage of the parallel hardware.

As mentioned, the advantage of parallel machines lies in parallel applications using the parallel architectures. It does not help to port a sequential problem to a parallel machine, as the parallel machine will most probably take just as long as a sequential machine to compute the problem, if not longer. One exception here is that multiple sequential program modules can be computed simultaneously on a parallel machine.

1.($20 million for a CRAY-2 having a peak performance of 2 GFlops but a sustained performance of only 250 MFlops.)

A user who has to excute $n$ runs of the same simulation program, can execute all $n$ runs on $n$ processors at the same time, instead of running $n$ runs one after the other on a sequential machine.

A big advantage of parallel machines, however, is the issue of scalability, that is "scaling up" the power by adding more processor nodes. A user can then buy an inexpensive stripped-down machine and add processor nodes later on. Nevertheless, scalability is not a trivial task. Building easily scalable machines requires attention to issues such as memory latency and bus bandwidth. These are some of the aspects which the MC$^2$ project had to address as one of the main objectives of the project was scalability. The user should be able to buy a single transputer board and use it to develop parallel programs, and then scale up to a MC$^2$ Cluster[1] containing the number of processors satisfying his requirements. MC$^2$ Clusters can then be used to form an MC$^2$/64 system, or even an MC$^2$/128 or MC$^2$/256 depending on the user's computing power requirements.

We first review the history of parallel computers before returning to MC$^2$.

### 1.4.1 Prior to 1960

[2]The earliest reference to parallelism in computer design is thought to be in General LF Menabrea's publication of October 1842, entitled *Sketch of the Analytical Engine Invented by Charles Babbage*. He argued that the multiplication of two numbers, each consisting of twenty figures and requiring three minutes at the utmost, could be done faster if the machine could give several results at the same time. The idea of using parallelism to improve the performance of a machine had occurred to Babbage over a hundred years before technology made its implementation possible!

1.An MC$^2$ Cluster contains 1 to 16 processors.

2.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

The first general-purpose electronic digital computer, the ENIAC[1], was a highly parallel and highly decentralised machine. It had 25 independent computing units (twenty accumulators, one multiplier, one divider/square rooter and three table look-up units), each following their own sequence of operations and cooperating towards the solution of a single problem. ENIAC may also be considered as the first example of MIMD computing. When the ENIAC was completed, the first stored-program computers were already being designed. It was realized that the ENIAC could be reorganized in the centralized fashion of the new machines so that it was much easier to put problems on the machine. This first parallel machine operated as a serial centralized computer for most of its life. (The size of the ENIAC was such that it could fit into an entire street block.)

The first-generation vacuum tube computer containing several parallel features was the ACE and its commercial derivative, the DEUCE. This machine had 11 mercury delay lines, each with a capacity of 32 32-bit words and a circulation time of 1 ms. The arithmetic was serial. A card reader, card punch and multiplier could operate in parallel with the rest of the machine, and instructions existed that could perform a limited number of operations on all 32 numbers in a delay line.

Bit-parallel architecture became state of the art with the availability of static random-access memories from which all the bits of a word could be read simultaneously. The first commercial computer with parallel arithmetic was the IBM 701. This machine used electrostatic cathode ray tube storage and was followed in a few years by the first machines to use magnetic-core memory, amongst others the IBM 704. This machine had parallel arithmetic and the first hardware floating-point arithmetic unit. The IBM 704 with the other machines of its time, had all the data input or output pass through a register in the arithmetic

1.The Von Neumann computer architecture originated in the late 1940s from John von Neumann's work on ENIAC, the first electronics general purpose computer.

unit, thus preventing arithmetic from being performed at the same time as the IO. The IO units were always much slower than the rate at which the processor could manipulate the data and therefore the IO bottle-neck was always a problem in IBM 704 installations.

### 1.4.2 Scalar and pipelined vector machines

[1]*"A scalar computer is a machine that provides instructions only for manipulating data items comprising single numbers, in contrast to the vector computer that also has instructions for manipulating data items comprising an ordered set of numbers (that is to say a vector)"* (Hockney et al., 1988). The history of the first scalar computers is also the history of the introduction of increasing parallelism into the serial design of the single instruction stream/single data stream computers. *Pipelining* is the principle in which the different phases of an instruction execution are overlapped in some way.

During the 1960s, the first commercial supercomputers (for example, Control Data Corporation's 7600) employed pipelining. Control Data Corp. delivered the CDC 6600 in 1964, comprising magnetic core memory, 10 separate functional units for multiplication, division, addition, shift, Boolean, branch and increment, and 10 peripheral processors which were used to handle IO. The CDC 6600 was followed by the CDC 7600 whose increased performance was caused by a faster clock cycle and ten serially organised units that were replaced with eight pipelined units and one serial unit for division that could not be piped. The CDC 6600 and CDC 7600 were two of the most successful computers produced in the scientific market. They were renamed CYBER 70 model 74 and CYBER 70 model 76.

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

Seymor Cray left Control Data Corporation in 1972 to start a new company, Cray Research Inc. This company had the aim of building the fastest computer in the world. The CRAY-1 computer was designed and built in the short timespan of four years. The CRAY-1 provided 12 functional units, all pipelined, a fast clock and a 16-bank one-million-word bipolar memory. The CRAY-1 has eight vector registers, each capable of holding 64 floating-point numbers and a set of about 32 machine instructions for manipulating these vectors. The CRAY-1 is referred to as a pipelined vector computer. It was the first pipelined vector computer that was commercially successful and it remains the most successful of this type.

The CRAY X-MP was announced in 1982. It can best be described as two CRAY-1 computers working from a common memory of 2 or 4 Mwords. Two vector input streams and one output stream access memory simultaneously in both CPUs. The CPUs may work on different problems or different parts of the same problem. Synchronisation is achieved either via main memory or a set of synchronisation registers. In 1984 a four-CPU model CRAY X-MP was announced. The top of this range, the CRAY X-MP/48 has a peak performance of 840 MFlops.

The liquid immersion technology of the CRAY-2 involves submerging the whole machine into a bath of clear inert liquid fluorocarbon which acts as the cooling liquid. The CRAY-2 is designed to have up to four CPUs and a clock period of 4 ns. The performance of the machine is optimally 2 GFlops. The next major development for CRAY is the use of gallium arsenide chips which should allow the clock speed period to be reduced approximately by 1 ns. This machine (probably the CRAY-3) was scheduled for 1990. It is likely to have 16 processors, a shared memory of 1 Gword and a performance of approximately 15 GFlops.

### 1.4.3 The ILLIAC IV

[1]An important paper was published by Slotnick et al. in 1962, entitled "The SOLOMON Computer". This paper describes a two-dimensional array of 32X32 processing elements, each with a memory for 128 32-bit numbers and an arithmetic unit working in bit-serial fashion. Contrary to the development of the serial computer into the vector pipeline computer, the SOLOMON concept was a radical change in thinking on computer architecture. Although the SOLOMON computer was never built, it gave birth to the ILLIAC IV, the Burroughs PEPE floating-point processor arrays, the Goodyear STARAN and the ICL DAP arrays.

The ILLIAC IV was designed at the University of Illinois in 1966. The ILLIAC IV was to comprise four quadrants, each with a control unit interpreting a single stream of instructions for 64 floating-point processing elements (PEs). Each PE was to have 2000 64-bit words of thin-film memory and the PEs in each quadrant were connected as an 8X8 array. The four quadrants were to be connected by a highly parallel IO bus and supported by a large disk as secondary memory from which jobs could be retrieved. The ILLIAC IV was a machine that could be regarded as a failure in that it cost more than four times the original contract figure and it could not reach its originally proposed performance, mainly because the design could not be realised with the technology available. Its influence, however, was profound. Not only in regard to computer architectures, but also in terms of software development as quite a considerable amount of software (including computer languages) was developed. Many of the parallel architectures that appeared during the 1980s were influenced by the ILLIAC IV design.

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

The initial SOLOMON computer design was a 32X32 array of one-bit processors, each with 4096 bits of memory, conducting its arithmetic on 1024 numbers in parallel in a bit-serial fashion. This quite closely describes the ICL DAP (Distributed Array Processor) which was started in 1972. The first production model of the machine was installed in 1980. It comprises a 64X64 array of PEs, which are connected in a two-dimensional network with nearest-neighbour connections. The original DAP used small scale integration with 16 PEs and their memory per circuit board, in other words the PE logic is mounted on the same board as the memory to which it belongs. This is an interesting concept, as in the Von Neumann concept the logic and the memory are both conceptually and materially in different units which can lead to several bottle-necks. The ICL DAP introduced the concept of placing both the logic and the memory on one chip. In October 1986 AMT was founded to further develop the DAP concept. The AMT DAP 510 is a 32X32 array with 64 PEs per VLSI chip.

### 1.4.4 Array processors

[1]Another line of computer evolution involving parallelism has been the development of relatively cheap, special-purpose computers for processing arrays of data (array processors). This does not necessarily mean that it is arrays of processors, in fact most of the designs are based on the pipelined architecture. These computers normally require a host and are therefore also called attached processors. Their main application is in signal processing.

By far the most successful of this type of machine is the Floating Point Systems AP-120B. Floating Point Systems was founded in 1970, the machine was launched in 1976 and by 1985, 5 000 systems had been installed. The FPS AP-120B may be attached to either minicomputers or mainframes as host, and it performs 38-bit floating-point arithmetic in separate pipelined multiplication and

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

addition units. Three memories (for data, tables and program) and two 'scratch pads' of registers are provided with multiple paths between each memory and each arithmetic unit. Typical processing rates of 5-10 MFlops may be achieved. In 1980 FPS announced the FPS-164 Attached Processor, a 64-bit version of the AP-120B, with expandable main memory. The peak processing power is 11MFlops in this machine. The larger main memory eliminates some of the slow transfer times from the host computer to the FPS machine. A novel enhancement was announced in 1984 by FPS, the FPS 164/MAX. This machine has a standard FPS 164 as a master, and may add up to 15 MAX boards, each of which is equivalent to two additional FPS 164 CPUs.

### 1.4.5 Expert systems

[1]The Japanese challenged the computer world in October 1981 with a vision of computing in the 1990s which they called the Fifth Generation. The Japanese identified expert and knowledge-based systems (KBS) as the main application area for computing in the 1990s. These systems use a database of rules to represent the knowledge of an expert. The system is intended to behave very much like a consultation between a human expert and someone with a problem. To date the most successful applications of these fifth generation computers have been in limited areas of well-defined knowledge.

In order to provide the above capability, an expert system must comprise three parts: a knowledge database machine with 100 to 1000 Gbyte of storage capacity for the accumulated knowledge; a problem-solving and inference machine to manipulate the knowledge database and to respond to questions; and an intelligent interface machine to communicate with the human user in speech and images. The basic computer requirements of the inference machine can be expressed as the number of logical inferences made per second (LIPS), and it is

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

estimated that the system would need to perform between 100 MLIPS and 1 GLIPS. Present computers have the capability of between 0.01 and 0.1 MLIPS, therefore a much more powerful machine is needed. Such factors are only likely to be achieved by a combination of improved technology, architecture and methods. It is also necessary that computers become more parallel and use architectures such as data-flow which automatically makes use of the inherent parallelism in a problem. In contrast to computers controlled by instruction streams (control-flow computers), the only limits to the order of execution are those imposed by data dependencies. Subject to these constraints that can be identified by the hardware, a data-flow computer may perform as many operations in parallel as it has functional units.

Another important concept of the knowledge-based systems is the vast improvement in the man-machine interface (MMI). Currently available computers are weak in speech recognition, textual manipulation and graphical communication, all of which are important for an intelligent interface to the human user. All these functions can only be supplied by subtantial computing power. The Fifth Generation is therefore more dependent on very powerful machines than any of the previous computer generations.

### 1.4.6 Mini-supercomputers

[1]The mini-supercomputers are also called the 'affordable supercomputers'. The appearance of these computers is another aspect of the computer revolution introduced by VLSI chips with 10 000 or more gates per chip.

**Convex**

The mini-super Convex machine has used a CRAY-like architecture and software. Like CRAY, Convex offers pipelining and vector processing.

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

Convex can reach a speed of 60 MIPS with eight processors. The architecture of the Convex C-1 is broadly similar to the CRAY-1 in that it is based on a set of functional units working from vector registers. The detailed architecture is quite different. Each functional unit comprises two identical pipes, one for odd and the other for even elements. Each pipe performs a 64-bit operation every 200 ns or a 32-bit operation every clock period of 100 ns.

**SCS-40**

Scientific Computer Systems Corp. offers the SCS-40, a 64-bit scalar-vector mini-supercomputer with more than 40 MFlops peak vector performance and more than 18 MIPS scalar performance. The SCS-40 appeared in 1986. Like the Convex C-1, the manufacturers claim that the SCS-40 delivers 25% of the performance of the CRAY X-MP/1 at 15% of the cost. The SCS-40, unlike the C-1, uses the CRAY instruction set, and CRAY programs should run without alteration. The architecture comprises 16 main memory banks connected via multiple buses and a vector crossbar switch to eight 64-element vector registers. These in turn are connected by multiple buses to a set of pipelined functional units, corresponding to those of the CRAY series. If the floating-point add and multiply lines work simultaneously, the peak performance is 44 MFlops.

**Alliant FX/8**

Alliant Computer Systems Corp. designed the FX/8, a mini-supercomputer with between one and eight parallel processors and a peak performance of 94 MFlops. The Alliant FX/8 is the only mini-supercomputer described here which is based on MIMD architecture. Eight CEs share a common memory. Each CE is a vector computer with eight vector registers holding 32 64-bit words each, and separate pipelined functional units for floating-point addition, multiplication and division. An eight-CE machine has a theoretical peak performance of 94 MFlops (32-bit) or 47 MFlops (64-bit). The CEs are connected via a crossbar

switch to two 64 Kbyte caches. The caches in turn access the shared memory via a memory bus. The machine also contains a 'concurrency bus' that is directly connected to all the CEs. This is used to synchronise the CEs with the minimum amount of overhead. The Alliant FX/8 is also sold separately, for example, as an Apollo DOMAIN workstation. The Alliant systems are vector-parallel systems that include built-in graphics capabilities. The newest member of the series, the FX/2800 family, also excels in scalar performance by virtue of the use of Intel's RISC i860 processors.

### 1.4.7 MIMD computers

[1]*Multiple instruction stream / multiple data stream* computers are controlled by more than one stream of instructions. The term is also limited to tightly coupled systems in which the instruction streams can be programmed to cooperate in the solution of a single problem. There are many proposed MIMD architectures and only those that have been built and successfully operated are described below.

**The Connection Machine**

The Connection Machine, made by Thinking Machines Corporation of Cambridge, Mass., uses a so-called hypercube to connect 65 536 custom made processors. Its massively parallel approach yields up to 2.5 billion instructions per second for general computing purposes and 2.5 GFlops in the CM-2 model introduced in April 1986. Externally the Connection Machine (designed by Danny Hillis) is a black cube consisting of eight smaller black cubes covered with flashing red lights that are used for trouble-shooting. Each subcube contains 16 boards arranged vertically.

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

Each board contains 32 chips. Each chip consists of 16 individual processors, each with its own memory. Data storage consists of 42 Winchester hard disk drives connected to operate in parallel.

**Carnegie-Mellon C.mmp and Cm***

One of the most ambitious early examples of MIMD computing was the C.mmp computer at Carnegie-Mellon University. This comprised of 16 DEC PDP-11 minicomputers connected to 16 memory modules by a 16X16 crossbar switch. This machine was completed in 1975 and remained operational till 1980.

The Cm* was an entirely different concept using microprocessors and memory subdivided amongst the microprocessors and made local to them. The Cm* used a hierarchical packet-switching network to provide communications between the processors. The primary unit was the 'computer module' which comprised a DEC LSI-11 microprocessor with 64 Kbytes of dynamic memory, and perhaps peripherals attached to its LSI-11 bus. This module could be used as an independent computer but up to 14 could be linked to a common intra-cluster bus to form a tightly coupled cluster. Within the cluster data transfer is by direct memory access. The system is built up by linking clusters together via two inter-cluster buses to form a loosely coupled network exchanging data by packet-switching techniques.

**Erlangen EGPA**

Perhaps the most original and imaginative MIMD architecture was developed at the University of Erlangen, West Germany, and is called the Erlangen General Purpose Array (EGPA). The connections between the computers are topologically similar to that of a pyramid, with the computers at the corners and connections along the edges. The control C-computer, at the top of the pyramid, controls four B-computers at the corners of its base. The four B-computers also have direct connections along the edges of the base. The idea is expandable to further levels by

making each B-computer itself the top of a further pyramid with four A-computers at its base. The advantage of the EGPA pyramidal topology is that short-range communications can take place most effectively along the nearest-neighbour connections, whereas long-range communication can take place most effectively by sending the data higher up in the pyramid. The five-computer system has been used for contour and picture processing, and has proved to be about three times faster than a single computer of the same type.

**Livermore S-1**

The largest MIMD project is the Livermore S-1 computer of the US Navy and Department of Energy, which comprises 16 CRAY-1-class pipelined vector computers connected to 16 memory banks by a full crossbar switch which provides a direct logical connection between each computer and each memory bank. Each of the component computers is provided with a data cache (64 Kbytes) and an instruction cache (16 Kbytes) in order to limit the traffic through the switch. All the computers are also directly connected to a common bus to facilitate the transfer of small amounts of data and the synchronisation of messages between them.

**Hypercubes**

Binary cubes were studied extensively as possible interconnection networks for MIMD computers, but they were not realised in a practical way until the Cosmic Cube was built. The point, line, square and cube with nodes at their corners and connections along their edges are the zeroth-, first-, second-, and third-order hypercube networks. Each order is made by taking two copies of the next lower-order hypercube and joining corresponding corners. In this sequence the $d$th-order hypercube has $n = 2^d$ m\nodes and $d = \log_2 n$ connections to each node. One attraction of this connection scheme is therefore that the number of connections grows relatively slowly with the size of the hypercube.

**Intel iPSC and iPSC-VX**

In 1985 Intel announced the iPSC hypercubes in sizes $2^5$, $2^6$ and $2^7$ rated at 2, 4 and 8 MFlops. Each processing node contains an Intel 80286 microprocessor with an 80287 arithmetic coprocessor. In 1986 Intel announced the iPSC-VX in which a pipelined vector processor board is attached to each node, increasing the peak theoretical performance to 20 MFlops per node in 32-bit mode. Two boards are now required per node and thus the maximum number of nodes are reduced to 64. As there has been no accompanying increase in the rate of internode communication, the performance of this machine is likely to be determined primarily by the time spent in communication rather than arithmetic. Careful problem formulation and programming are therefore required to realise performances anywhere near the peak rates.

**Floating Point Systems T series**

The Floating Point Systems T series is a massively parallel design that combines parallelism, pipelined vector arithmetic, VLSI and RISC architecture. The basic unit, a T/20, consists of two system nodes each with eight processors and is capable of 192 MFlops peak performance. Floating Point states that a speed of 65 GFlops is attainable in the T/10000 by combining 256 T/20 cabinets for a total of 4 096 processors. Each node has an Inmos transputer to handle communication and 1 Mbyte of dual port video DRAM as memory. The DRAM can feed a 256-element contiguous vector of 32-bit numbers in parallel to one of four vector registers which themselves feed two WEITEK 64-bit pipelined floating-point arithmetic chips for multiplication and addition respectively. The theoretical peak performance is 16 MFlops per node. In this architecture we again find that the communication between nodes is at least two orders of magnitude slower than the arithmetic which complicates programming and problem formulation immensely if one is to achieve peak performance rates. These machines use the Trollius operating system.

**Bus-connected systems**

Bus-connected systems are by far the most common commercial MIMD computers. They are based on attaching multiple computing elements (CEs) and multiple memory modules to a common bus. Some examples are discussed below.

**ELXSI System 6400**

The Elxsi System 6400 can consist of one to twelve processors and uses parallel multiprocessing and vector capability to reach a performance of 100 million Whetstone instructions per second. The ELXSI 6400 is an example of a MIMD system in which the processors and memory are connected by a fast shared bus. In ELXSI there may be from one to ten CPUs and one to four IO processors accessing from one to six memory systems (4 to 192 Mbytes) via a common high-speed databus known as the Gigabus. Also taking into account the bus cycles required, the total time to complete a floating-point multiplication in one CPU is about 800ns, giving a performance of 1.2 MFlops.

**Sequent Computer Systems : Sequent Balance**

Another proponent of parallel processing is Sequent Computer Systems with its Sequent Balance product family. It can configure up to 12 PEs (model 8000) or 24 PEs (model 21000) and up to 28 Mbyte of shared common memory attached to a 52-bit wide pipelined packet bus. The PEs are National Semiconductor 32032 microprocessors and provide performance from 1.4 to 21 MIPS. The Sequent Balance is a 32-bit bus-connected shared-memory architecture.

**BBN Butterfly**

The BBN Butterfly was produced by BBN Advanced Computers Inc., Cambridge, Mass. Using a parallel architecture based on the Motorola MC68020 microprocessors, the Butterfly is able to produce 250 MIPS of processing power with a combination of 256 processor nodes. A proprietary high-performance switch called the Butterfly switch directly

interconnects all the processor nodes This computer differs from other MIMD systems in that it is a distributed-memory switched system. All the memory of the system is distributed as local memory to the multiple CEs which are then connected to each other by a multistage packet switch. The distributed nature of the memory is hidden from the programmer who may write his program as though all the memory were shared by the processors. The only difference between accessing data in the CE's local memory or the local memory of another CE, is the time for the messages to pass through the switch. The topology of the switch is a Banyan network which is similar to that required to bring data together in the $log_2 n$ stages of a fast Fourier transform on $n$ data, hence the name Butterfly switch.

**Specific MIMD designs compared to MC$^2$**

In this subsection a few MIMD machines comparable to the design of MC$^2$/64 are considered. It is beyond the scope of this dissertation to give a detailed description of each machine and how it compares with the MC$^2$. It can be mentioned, however, that the Parsytec and the Meiko systems are two systems that have a very similar approach to the MC$^2$ system.

**Meiko Machine**

The Meiko Computing Surface is a MIMD machine with a distributed memory architecture. The system is highly modular, and as more nodes are added, the processing power, memory bandwidth and communication bandwidth all increase linearly. The Meiko Computing Surface at Edinburgh University consists of 200 T800 transputers, each with 4 Mbyte of memory. (Meiko also produces smaller systems which do not fall within the scope of this discussion.) The Meiko machines are based on a domain type of system where a number of processors represent a domain of processors. These units are all linked by means of large crossbar switches

designed by Meiko. IO of the specific system at Edinburgh is through a MicroVAX computer. Some fast hard disk subsystems based on the Meiko MK021 disk controller board are included in the system.

The operating system used on the Computing Surface is a dual operating system called MEIKOS, which is a Unix-based disk filing system. The domain switching network of the MEIKO is controlled by an operating environment called M2VCS. Programming in the domain is done using OPS (Occam Programming System) based on the Inmos TDS[1]. All aspects of parallelising code is left to the user. Any configuration can be switched within a domain. Simulated annealing is used to minimise the length of connections between processors in the network requested by a user. MEIKO also supports the Helios operating system. MEIKO also supports CSTools, a program development toolset for multiprocessor computer systems. CSTools consists of cross-development tools such as compilers, configuration systems and runtime facilities such as high-level communications services and symbolic debuggers.

The latest release of MEIKO is processor nodes which use the Intel i860 and the Sun Microsystems SPARC processors. The reason why they moved away from the transputer seems to be that they need more powerfull processor nodes, and Inmos has been very slow to provide a more powerfull transputer. In the new processor nodes the i860 (for instance) provides the computing power while a transputer is used as a link engine.

**Parsytec Megaframe Supercluster**

The Parsytec Supercluster is a second generation transputer-based machine and incorporates a number of interesting features. The Supercluster series consists of two models, a 64-processor model and a

1.TDS is an acronym for Transputer Development System, an environment developed by Inmos for the development of Occam programs.

256-processor model. The 64-processor Supercluster in turn consists of 4 logical subclusters. Each one of these subclusters provbides complete configurability through two 96X96 crossbar switches. These switches are built using a number of C004 switches from Inmos. Each cluster provides 32 links for connecting the subclusters to form the 64-processor Supercluster. The 256-processor Supercluster comprises 4 of these 64-processor Superclusters.

The IO unit of the Parsytec Supercluster is located in a system services cluster which is connected through links to the 96X96 crossbar switches of each one of the subclusters. It seems that this aspect of the Supercluster design is probably one of its best features as high speed IO is lacking in most of the other transputer-based computers. The system services cluster contains up to 4 SCSI units which can attain a maximum transfer rate of 6 Mbytes per second. The disk system supports the Unix 4.3 BSD standard file system. Other IO features of the system services cluster include interfaces to non-host units such as terminals.

The Parsytec Supercluster supports the Helios operating system as the main operating system.

The architecture of the Parsytec is in many aspects similar to that of the $MC^2$ system, but the architectures evolved entirely independent from each other, although over a similar period of time.

**The Supernode (Esprit P1085)**

The Supernode consists of units containing 16 transputers which are connected by means of a 72X72 crossbar switch. These crossbar switches are gate arrays made by Thorn (NEC ASIC's). The switches gave problems with resynchronisation of the information passing through them (unlike the Inmos C004 made specifically to switch transputer links). The project was industrialised by TELMAT Informatique in 1988.

**The Telmat T.Node**

TELMAT Informatique was born in 1985 out of the successful ESPRIT Supernode. TELMAT industrialised the Supernode to bring the T.Node to the market in 1988.

The T.Node is a massively parallel computer based on the Inmos transputer. Telmat has created a family of parallel computers, the T.Node (8-32 transputers), the T.Node tandem (32-64 transputers) and the Mega.Node (128-1024 transputers). The T.Node family is a fully reconfigurable, modular and expandable transputer network. Dynamic switching devices, integrated in all systems, allow the full reconfiguration of all the configurations, from 8 to 1 024 transputers. The graph of the network can be dynamically modified by the program, according to the nature of the calculation to be performed. Results can be stored on disks, transferred in memory for graphic systems or used by the host computer. The T.Node tandem is the basic building block of the Mega.Nodes.

The T.Node Worker Modules is the basic computation element of the T.Node system. Each board contains 8 T800 transputers. Each of the T800's has 3 Mbytes of memory. The links of all the worker processors in the T.Node tandem are connected in a switching device, which can modify the network topology of the T.Node. The switch is a specific VLSI circuit. The T.Node tandem can again be seen as the "node" of the larger configurable network of the Mega.Node. Another switch is used to connect T.Node tandems together, allowing full reconfigurability of up to 1 024 transputers.

The T.Node supports Helios as operating system. Telmat also provides T.Node Tools, a software toolbox integrated in the implementation of Helios on the T.Node systems. The T.Node Toolbox allows a user to switch the T.Node, and provides support for selective reset/analyse and support for multiple users of the T.Node system.

**Transtech Devices' Multi-Computing Platform**

The Multi-Computing Platform is a parallel processing system from Sun-3 and Sun-4 workstations. The MCP1000 has a transputer-based motherboard that supports up to 32 transputer modules (TRAMs). The motherboard connects a 32-bit VME interface into a processor farm consisting of 4 sites, each having eight TRAM slots. The processor farm feeds into a transputer link-switching matrix and then into a bank of link outputs. When fully loaded the MCP1000 is capable of 72 MFlops with up to 12 platforms per Sun chassis. This offers a computing facility of 864 MFlops. Up to four users can have access to a Multi-Computing Platform from any point in a Sun network. Multiple platforms can be configured to run within the same Sun workstations as large multi-user computational engines, or across the Sun Network as a distributed computing facility.

The operating system supported is GENESYS. GENESYS resides alongside SunOS.

**Chorus Supercomputer ComputeServer**

The ComputeServer is a high-performance parallel processing system designed specifically for access by Sun, Macintosh and PC-compatible systems over industry standard networks such as Ethernet. It is designed to accelerate compute-intensive applications remotely via a multiprocessing system based on 1 (CS1) to 16 (CS16) processors. Current systems are based on the Inmos T800, but the announced systems will be based on the Intel i860. Cost-effective hardware upgrades are available, and source-code portability between successive generations of ComputeServer hardware is provided by adherence to the Linda C and Fortran 77 open specifications.

Linda is rapidly emerging as an industry standard syntax for parallel programming. Chorus' Linda C implementation, the most robust Linda

for any platform, also transparently handles all networking and RPC functions between the desktop environment and the ComputeServer system.

**The XTM of Cogent Research INC.**

The XTM is a desktop parallel supercomputer. Cogent Research also provides QIX, a operating system that makes it easy to write programs that are independent of architecture and topology. QIX is based on Unix and uses Linda extensions to provide a distributed parallel operating system. Languages available include C, C++ and Fortran 77, with a graphical user interface based on X-Windows.

Present configurations support from 2 to 32 processors, currently the T800 transputer, and provide between 8 to 128 Mbytes of memory. A propriety dual bus architecture uses both a parallel bus and point-to-point links with an intelligent switch that reconfigures the communication topology as programs run. This creates a distributed-memory machine with close to shared-memory performance. Present versions deliver up to 160 MIPS and 48 MFlops.

**Computer System Architectures (CSA)**

CSA offers a broad family of transputer-based products, including add-in systems and host interfaces for Sun, Apollo and PC-based workstations; processor boards; peripheral interfaces; different compilers; operating systems and other software tools for the transputer. The SuperSet series of parallel computing products includes the SuperSet.16 and SuperSet.64 parallel processors with optional switched node topology and parallel disk and data acquisition subsystems. The SuperSet.16 provides 25 MFlops, and up to 25 Mbytes/sec of sustainable IO bandwidth, and contains 16 T800 transputers. The SuperSet.64 provides 100 MFlops and up to 100 Mbytes/sec of sustainable IO bandwidth, and contains 64 switched T800 transputers. CSA also provides individual boards, interfaces and software.

All CSA products are compatible and all systems are expandable. A SuperSet.16 can be upgraded to a SuperSet.64 with the addition of nodes and a chassis.

## 1.5 Classification of parallel architectures

There is no useful and accepted classification scheme for parallel architectures at present, mainly because there are so many different architectures. This can be seen from the brief overview of the best known systems in the previous section.

### 1.5.1 Levels of parallelism

[1]Parallelism has been used to improve the effectiveness of computers since the earliest computer designs. Parallelism has been applied to several distinct levels which may be classified as follows :

*Job Level / Task Level*
* between jobs
* between phases of a job

*Program Level*
* between parts of a program/ between procedures
* within DO loops

*Instruction Level*
* between phases of instruction execution

*Arithmetic and Bit Level*
* between elements of a vector operation
* within arithmetic logic circuits

Parallelism at the job or task level is mainly concerned with the organising of jobs such as IO jobs or computing jobs so that bottle-necks are minimised. Parallelism on this level is mainly handled by the operating system.

1.The information in this section is primarily obtained from the book "Parallel Computers 2", written by RW Hockney and CR Jesshope.

Program level parallelism is concerned with different sections (or procedures) of code of the same program that are so independent that they can be executed in parallel on different processors in a multi-processor environment. Some sections of independent code can be recognised from a logical analysis of the source code, others are data-dependent and are therefore not known until the program is executed. A data-flow analysis of a program can aide the programmer a lot in identifying parts of the program that can execute in parallel.

At an even lower level, the processing of any instruction may be divided into several suboperations, and pipelining may for instance be used to overlap the different suboperations on different instructions.

At the lowest level, one has the choice in the arithmetic logic itself, whether to perform this arithmetic in a bit-serial fashion, or on all bits of a number in parallel. This is parallelism at the arithmetic bit-level - an active area of development in the first generation of computers during the 1950s.

Owing to the formal CSP[1] methodology that was used for their design, transputers are geared specifically towards *Job Level* and *Program Level* parallelism. The rate of computing speed versus communications speed of the transputer is 5:1, making it ideally suitable for parallelism where the computing power versus communications power ratio is also 5:1. These problems, therefore, must feature the need for much computing power compared to little communication power.

Transputers are therefore mainly used in distributed-memory machines. A distributed-memory machine is a machine in which memory is local to the processors and is not shared in any way. A distributed-memory machine with local memory associated with each processor, but with a 1:1 computing to

1.CSP - Communicating Sequential Processes, a formal methodology especially developed for arguing about parallel processes and the way in which they interact with each other. CSP was developed by C.A.R. Hoare.

communication ratio, approaches the shared-memory system as the time spent on communication is the same as that spent on computing. The time spent on communication thus becomes irrelevant.

Distributed-memory machines, where the computing to communication ratio is >1:1 (such as $MC^2$ with 5:1), must place more emphasis on computing rather than on communication since communication will most probably, in most problems, be the limiting factor in determining the solution speed. Problems must be analysed to minimize the communications between processors while maximizing the computing per processor. This will ensure efficient use of the distributed-memory architecture when solving problems. A way to analyse the communication versus computing aspect of any problem is with a data flow analysis since such an analysis determines the flow of data through a software problem. With the results of a data flow analysis, the software problem can be mapped onto the hardware in such a way that flow of data between modules (the communication between processors) is limited.

#### 1.5.1.1 Levels of parallelism on the transputer

For the transputer the different levels of parallelism are provided through Occam, parallel compilers and available operating systems. Although the transputer is ideally suitable for parallelism at job or program level (parallel compilers and operating systems), parallelism can also be provided on instruction level through Occam.

##### 1.5.1.1.1 Occam

Occam, which is the basic language of the transputer (see section 2.4), provides parallelism at levels as low as the instruction level. The language is a subset of CSP and therefore provides parallelism at instruction level although, if running on one transputer, the transputer would most probably simulate the parallelism using time-slicing which is not real parallelism at all. The code below will result in *instruction* being executed 4 times in parallel.

```
PAR i = 0 FOR 4
  Instruction
```

But Occam can also provide parallelism at program level, for instance allowing different subroutines to be executed concurrently:

```
PAR
  Routine1
  Routine2
  Routine3
```

This code will result in Routine1, Routine2 and Routine3 to be executed in parallel.

Occam is specifically geared towards low-level parallelism, burdening the user with all the nitty and gritty of actually parallelising his code. Because Occam was specifically designed with the transputer in mind, it allows a transputer applications programmer to utilise all the power of a transputer, but burdens the user with all the effort necessary to actually make use of that power.

#### 1.5.1.1.2 Parallel compilers

Parallel compilers for the transputer are available for all main languages such as C, Pascal, Modula-2, Ada, Fortran, etc. These languages adhere to the different language specifications but with extensions, allowing a user

parallelism at program level, in other words, running different parts of the program in parallel. The user si still burdened with the actual parallelising effort, but at least it is in a language he is familiar with.

#### 1.5.1.1.3 Operating systems

Different levels of parallelism are provided by an operating system such as Helios (see Appendix C, section 7.3). Compilers for the different popular languages are also available and can be used as in the naked environment, but the real advantage of the operating system is that the parallelism is made transparent to the user. It is for instance, possible to let the operating system do all the load balancing for different tasks. This implies a degradation in speed as the overheads increase.

### 1.5.2 Flynn's taxonomy

"Flynn does not base his macroscopic classification of parallel architecture on the structure of the machines, but rather on how the machine relates its instructions to the data being processed. A *stream* is defined as a sequence of items (instructions or data) as executed or operated on by a processor. Four broad classifications emerge, according to whether the instruction or data streams are single or multiple:" (Hockney et al., 1988, p.56)

*SISD - Single Instruction Stream / Single Data Stream*

This is the conventional serial Von Neumann computer in which there is one stream of instructions and each arithmetic instruction initiates one arithmetic operation, leading to a single data stream of logically related arguments and results. (Examples: CDC 6600, CDC 7600)

*SIMD - Single Instruction Stream / Multiple Data Stream*

This is a computer that retains a single stream of instructions but has vector instructions that initiate many operations. Each element of the vector is regarded as a member of a separate data stream. This classification therefore includes all machines with vector instructions. (Examples: CRAY-1, ILLIAC IV, ICL DAP)

*MISD - Multiple Instruction Stream / Single Data Stream*

This class seems to be void as it implies that several instructions are operating on a data item simultaneously. (No Examples)[1]

1."Pipelining - the application of assembly-line techniques to improve the performance of an arithmetic or control unit" (Hockney et al., 1988, p.5). A pipelined architecture does therefore not fall into this category as instructions are not performed on the data item simultaneously.

*MIMD - Multiple Instruction Stream / Multiple Data Stream*

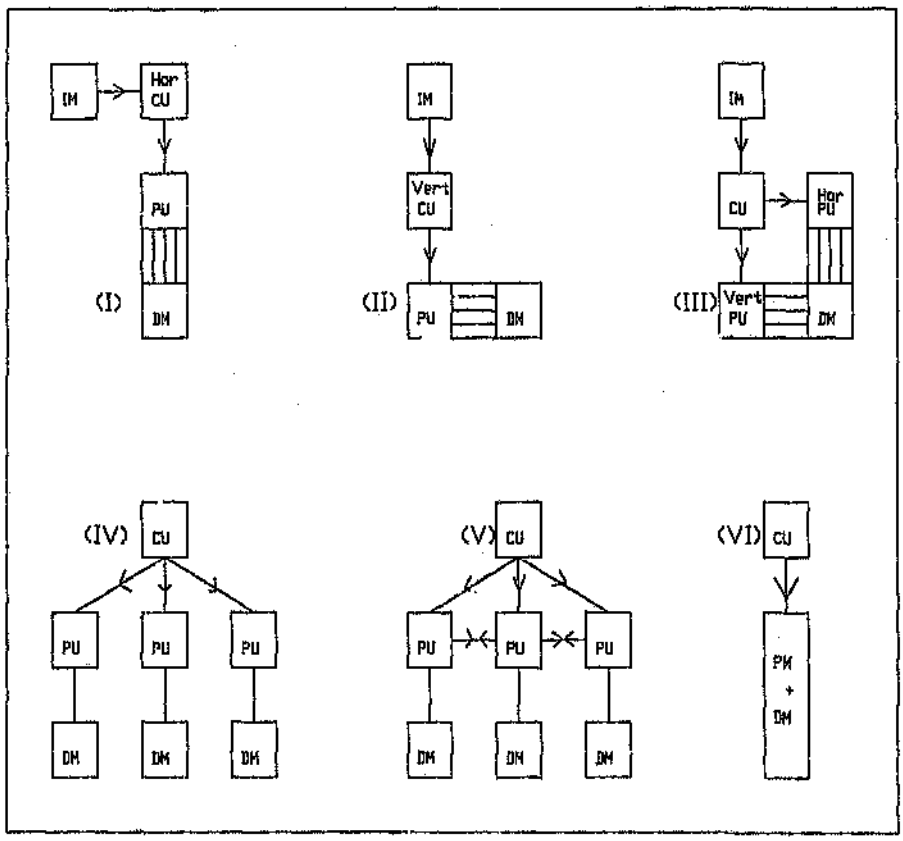
Multiple instructions streams imply the existence of several instruction processing units, and therefore necessarily several data streams. This class contains all forms of multi-processor configurations, from linked main-frame computers to large arrays of microprocessors.

Hockney and Jesshope evaluate the above taxonomy as being too broad, since it lumps all parallel computers, except the multi-processor, into the SIMD class and draws no distinction between the pipelined computer and the processor array, or between the different architectures. Flynn's taxonomy can be used for a broad distinction between different machines however (Hockney et al., 1988, p.57).

**MC$^2$/64** can be classified as a MIMD system as we can identify multiple instruction streams working on multiple data streams. This classification can also be subdivided to distinguish between the different architectures of MIMD machines and a further classification will therefore also be helpful in the MIMD class.

### 1.5.3 Shore's taxonomy

"Unlike Flynn, Shore (1973) based his classification on how the computer is organised from its constituent parts. Six different types of machines were recognised and distinguished by a numerical designator." (Hockney et al., 1988, p.58)




Schematic representation of Shore's taxonomy: (I) word-serial, bit-parallel; (II) word-parallel, bit-serial; (III) = (I+II), orthogonal computer; (IV) unconnected array; (V) connected array; (VI) logic-in-memory array (Hockney et al., 1988).

*Machine I* - The conventional Von Neumann architecture with a single control unit (CU), processing unit (PU), instruction memory (IM) and data memory (DM). A single DM read produces all bits of any word for processing in parallel by the PU. The PU may contain multiple functional units which may or may not be pipelined. (Examples : CRAY-1 = pipelined vector computer, CDC 7600 = pipelined scalar computer)

*Machine II* - The same as machine I, except that a DM read fetches a bit slice from all words in memory instead of the bits of one word, and the PU is organised to perform its operations in a bit-serial fashion. (Example: ICL DAP)

*Machine III* - Combination of machines I and II. It comprises a two-dimensional memory from which may be read either words or bit slices, a horizontal PU to process words and a vertical PU to process bit slices; in short, it is an orthogonal computer.

*Machine IV* - This machine is obtained by replicating the PU and DM of machine I (defined as the processing element, PE) and issuing instructions to this ensemble of PEs from a single control unit. There is no communication between the PEs except through the CU. The absence of connections between the PEs limits the applicability of the machine, but makes the addition of further PEs relatively straightforward. (Example: PEPE machine)

*Machine V* - This machine is the same as IV but with the added facility that the PEs are arranged in a line and nearest-neighbour connections are provided. This means that any PE can address words in its own memory and those of its immediate neighbours. (Example: ILLIAC IV)

*Machine VI* - Machines I to V all maintain the concept of separate data memory and processing units, with some databus or switching element between them, although some implementations of one-bit machine II

processors include the PU and DM on the same IC board. Machine VI, called a logic-in-memory array (LIMA), is the alternative approach of distributing the processor logic throughout the memory. (Example: ICL DAP).

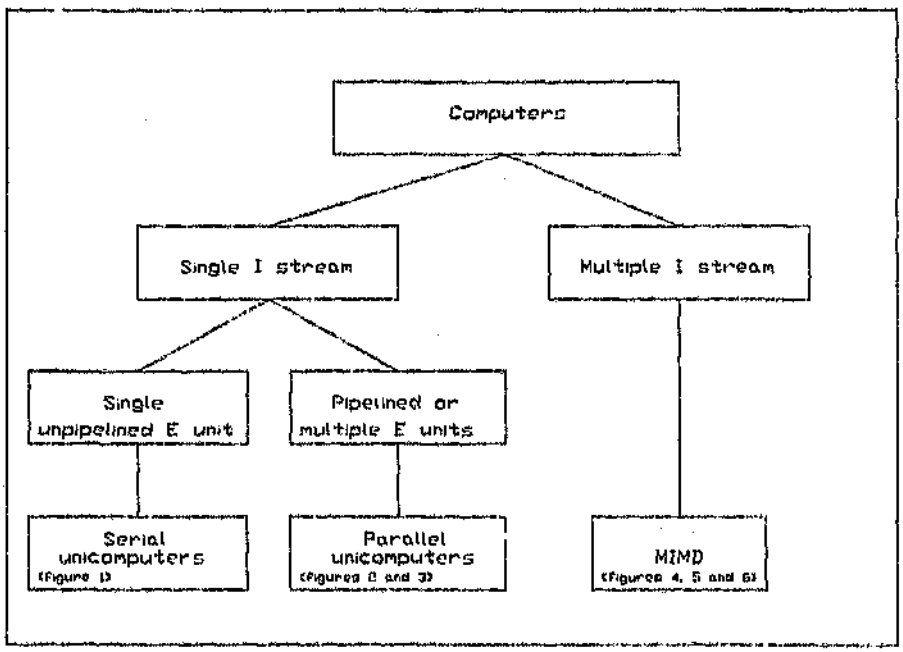
Hockney and Jesshope evaluated Shore's taxonomy: ".. we can see that Shore's machines II to V are useful subdivisions of Flynn's SIMD class, and that machine I corresponds to the SISD class. Again the pipelined vector computer, which clearly needs a category of its own, is not satisfactorily covered by the classification..." (Hockney et al., 1988, p.59).

$MC^2$ does not fit one of these classifications. A $MC^2$ transputer node is, in effect, a node containing the CU, PU, IM and DM since such a node is a separate entity running a separate task. Nodes communicate with each other through communication links. $MC^2$ can be arranged (or switched) to simulate machines IV, V and VI. In such a case one of the nodes will take over the role of the CU, and the other nodes can be arranged as PUs but with their DM attached to them, as memory is associated with a node in the $MC^2$ system.

### 1.5.4 Structural taxonomy

Hockney and Jesshope formulated a structural taxonomy based on a structural notation. This is the best taxonomy so far describing parallel machines. A summary of this taxonomy is given here (Hockney et al., 1988, p.73).

The structural classification taxonomy of parallel machines is based on tree structures and there is therefore only one route from the top of the diagram to any of the classes of computers that are defined on the bottom line. Certain large computers have properties belonging to more than one class and Hockney and Jesshope chose the dominant property to classify these machines.




The broad subdivisions in computer architecture (Hockney et al., 1988, p.73)

At the highest level the computers are classified according to the functional classification of Flynn. Computers are divided into those with single instruction streams (SI), and multiple instruction streams (MIMD).

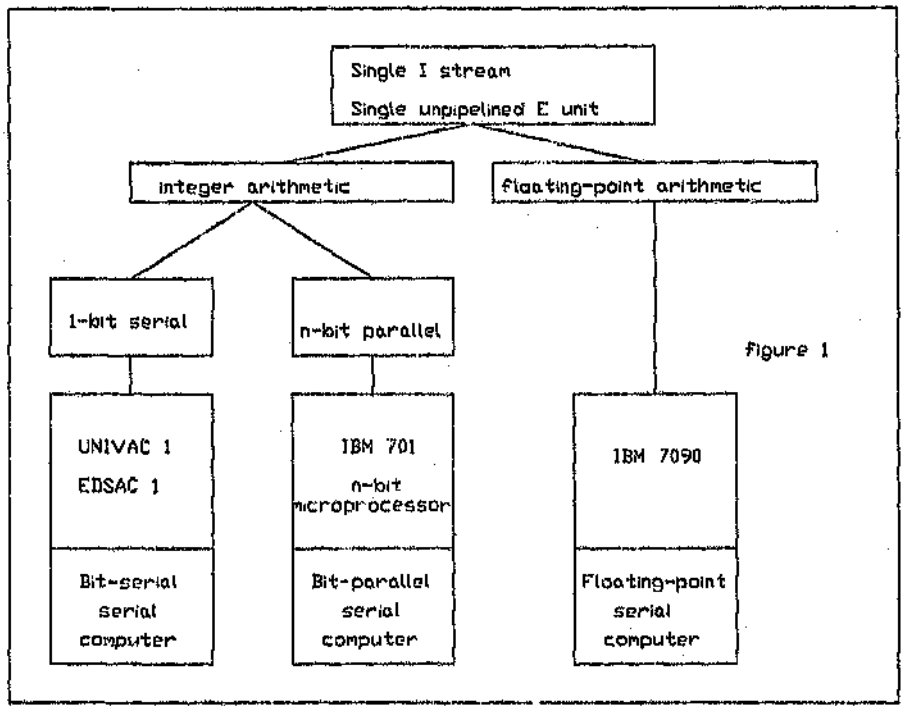



Figure 1 : Classification of serial unicomputers (Hockney et al., 1988, p.74)

SI machines are further subdivided into those with a single unpipelined E unit and those with multiple and/or pipelined E units[1]. Remembering that an E unit may only execute one function at a time (even though it may be able to compute many functions), the single unpipelined E unit subdivision leads to sequential operation and includes all serial computers. The multiple E units or pipelined subdivision allows different types of overlapping in the family of parallel unicomputers.

The next level of subdivision is based on the type of arithmetic that is performed by the E units. The difference in complexity between a one-bit arithmetic unit (less than 10 logic gates) and a floating-point arithmetic unit (thousands of logic

1.E unit : Executional Unit

gates) is sufficiently large to constitute a qualitative difference which should be recognised. (The extra space required for floating-point circuitry also places divergent constraints on assembling large arrays of processors from those associated with one-bit processors.) In order to include the historic evolution of serial computers, this class has been divided into integer arithmetic and floating-point arithmetic. The integer arithmetic class is divided into serial and parallel arithmetic.

Figure 2 covers the introduction of functional parallelism and pipelining into the traditional serial computer concept, and Hockney and Jesshope first differentiate into separate classes computers with, and without pipelined E units. On the left is the unpipelined multi-unit scalar computer, such as the CDC 6600, which obtains its performance entirely by functional parallelism. On the right the pipelined computers are first divided into those with or without explicit vector instructions. This division is necessary to separate the high-performance scalar computers (CDC 7600) from the pipelined vector computers (CRAY-1). The pipelined computers with vector instructions are further subdivided into those with separate special-purpose pipelines for each type of arithmetic and those with one or more general purpose pipelines capable of performing more than one type of operation. The pipelined computers with only scalar instructions are subdivided into those in which one instruction controls all units at each cycle and those in which instructions are issued to units individually when they are ready to carry out an operation.

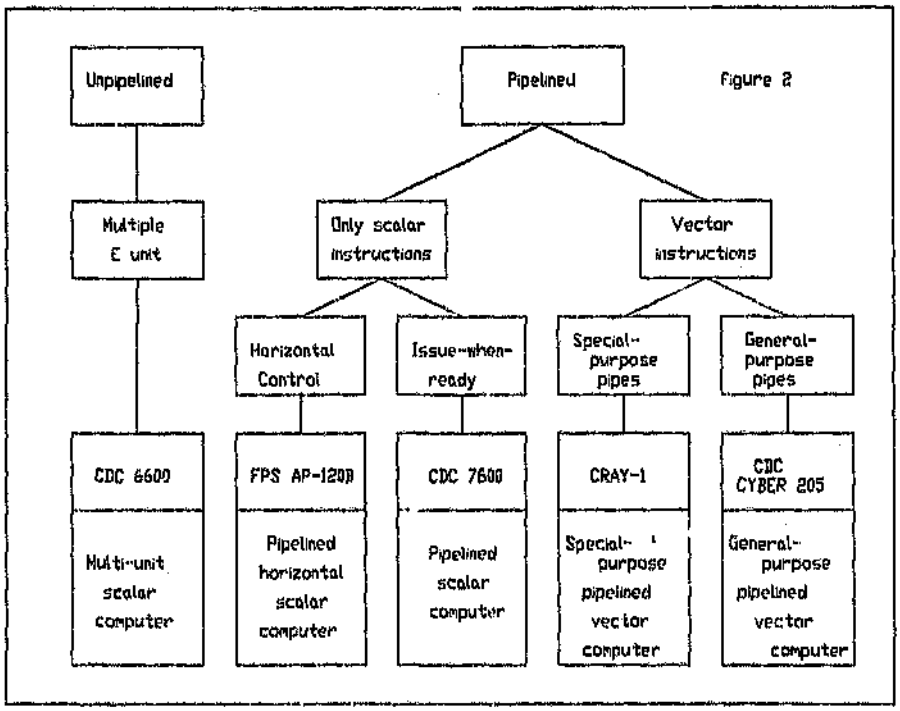



Figure 2 : Parallel unicomputers based on functional parallelism and pipelining. (Hockney et al., 1988, p.75).

The alternative of obtaining parallelism by the replication of processors under lockstep control is considered in figure 3. This is again divided into the floating-point class and the few-bit class. The next subdivision concerns the connections between the processors, whether these be unconnected or connected to neighbours. Other forms of connections are lumped loosely under the class of cross-connected processors and memory.

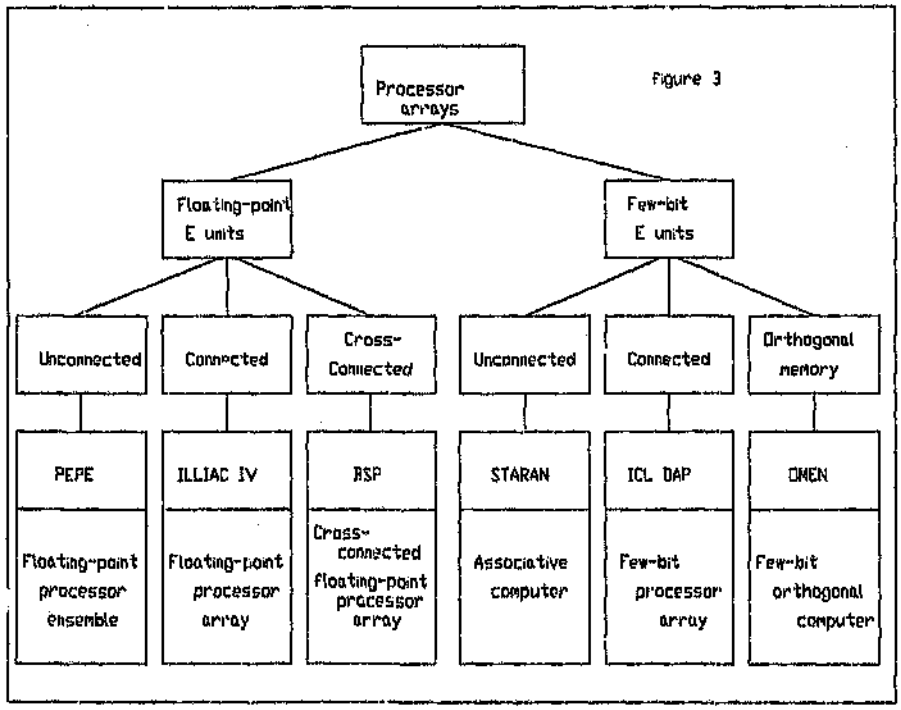



Figure 3: A classification of processor arrays (Hockney et al., 1988, p.76)

In figures 4, 5 and 6 Hockney and Jesshope shows a possible taxonomy for MIMD computers. In this taxonomy they have only included computers controlled by multiple streams of conventional instructions, in other words, control-flow computers.

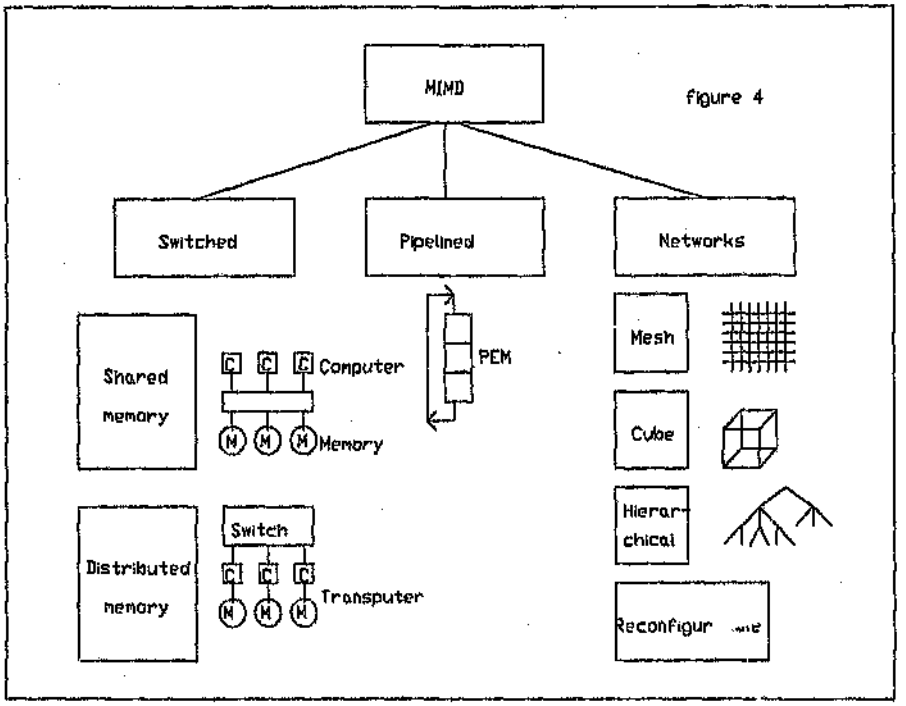



Figure 4 : A structural taxonomy of MIMD computer systems (Hockney et al., 1988, p.78)

Figure 4 shows the broad division into pipelined, switched and network systems. Multiple instruction streams may be processed either by time-sharing a single sophisticated pipelined instruction processing unit (pipelined MIMD), or by providing separate and simpler instruction processing hardware for each stream. MIMD systems using the second alternative naturally divide into those with a separate and identifiable switch (switched MIMD) and those in which computing elements are connected in a recognisable and often extensive network (MIMD networks).

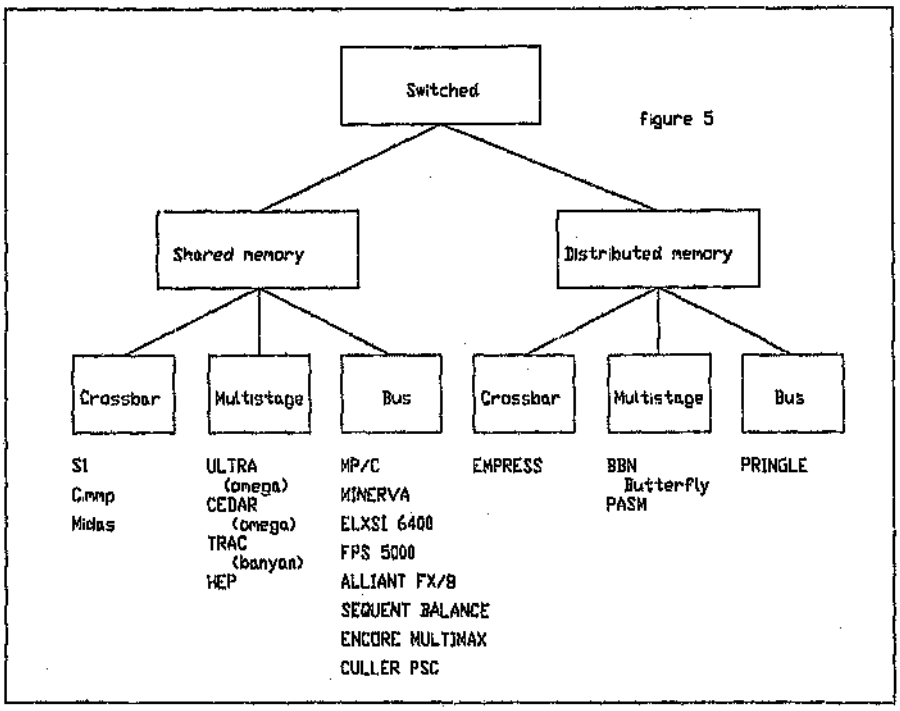



Figure 5: A subdivision of the switched class of MIMD computer systems (Hockney et al., 1988, p.79)

In the *switched MIMD systems* all connections between computers are made via the switch, which is usually quite complex and a major part of the design. In the *MIMD networks* individual computing elements (CEs) may only communicate directly with their neighbours in the network, and long-range communication across the network requires the routing of information via, possibly, a large number of intermediate CEs. The CE must therefore provide a small computer (microprocessors), a portion of the system memory, and a number of links for connection to neighbouring elements in the network. Early network systems provided these facilities on a board. The Inmos transputer now provides a CE in a chip and is therefore an ideal building bock for MIMD networks (but also for switched MIMD systems).

In networked systems the CEs are the nodes of the network and may also be called nodal computers or processors, or processing elements. Within this definition they must be called CEs in order to indicate that they are complete computers with an instruction processing unit. The term PE is reserved for the combination of arithmetic unit and memory without an instruction processing unit.

Switched systems are further subdivided into those in which all the memory is distributed amongst the computers as local memory and the computers communicate via the switch (distributed memory MIMD), and those in which the memory is a shared resource that is accessed by all computers through the switch (shared-memory MIMD). A further subdivision is then possible according to the nature of the switch, for example, crossbar, multistage and bus connections in both shared- and distributed-memory systems. Many larger systems have both shared common memory and distributed local memory. Such systems should be considered as hybrids or simply switched MIMD systems. Hockney and Jesshope prefers to classify them as variations with the shared-memory section and reserve the distributed-memory section for systems with no separate shared memory. The classification is based on the location of memory that is intended for permanent storage of the main data of the problem. Local cache memory that is present in almost all systems for temporary storage during calculation is not relevant for this classification.

All MIMD networks appear to be distributed-memory systems, but they may be further subdivided according to the topology of the network as in Figure 6. The simplest network is the star, in which several computers are connected to a common host. Single and multi-dimensional meshes are also used. Binary hypercube networks in which there are only two computers along each dimension form an interesting class which is receiving a lot of attention (Intel iPSC). There are also examples of hierarchical networks based on trees, pyramids and bus-connected clusters of computers. The most suitable computer network certainly depends on the nature of the problem to be solved, hence it is attractive

to have a MIMD network which may be reconfigured under program control. The CHiP computer and the Southampton ESPRIT Supernode computer are designed to satisfy this requirement (as is MC$^2$/64).

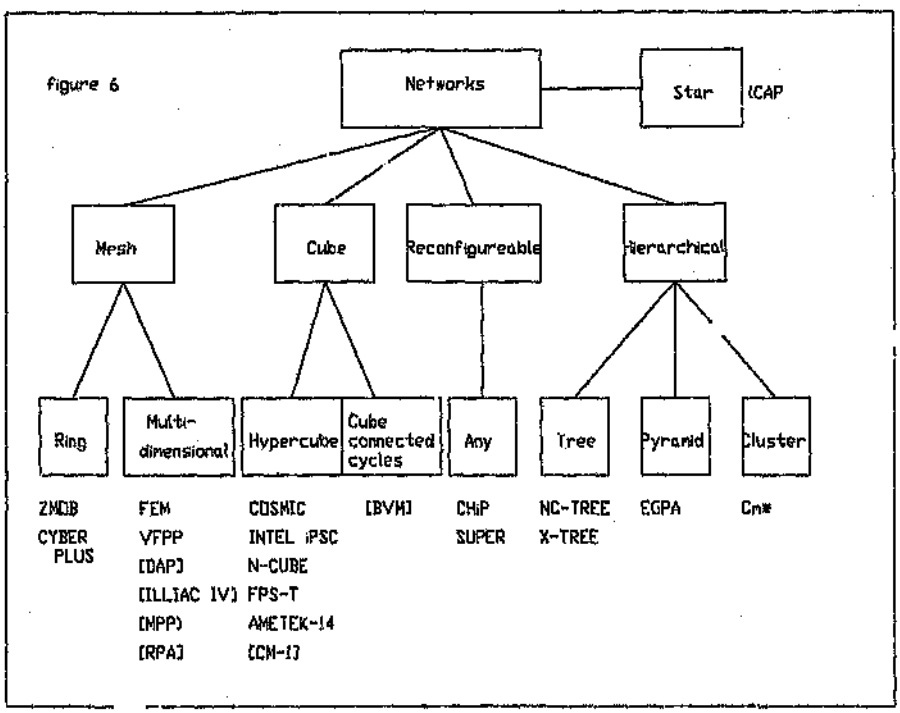



Figure 6 : A subdivision within the networked class of MIMD computer systems (Hockney et al., 1988, p.80)

The author would classify MC$^2$/64 as a switched network, distributed memory MIMD machine according to the classification scheme of Hockney and Jesshope. MC$^2$/64 falls somewhere between the switched and the networked MIMD machines. Any network can be switched in the MC$^2$ machine, but for program duration this network stays the same (implying a networked MIMD system). For the problem being executed, MC$^2$ is a networked MIMD machine. Although the nodes communicate with each other through a switch, the switch is not changed during execution time. This is a restriction imposed by Version 1 of the MC$^2$/64

control software, mainly because the transputer has no memory protection or communication guarantee if a user decides to change the network while the transputers are still engaged. If a user takes responsibility for handling the synchronisation of transputers while they are active on a problem, there is no reason why a network cannot be changed during certain phases of execution. An example could be a problem with different phases. Let us say phase 1 needs a nearest neighbour connection. The $MC^2$ network is switched into a mesh. Phase 2 needs to compute a Fourier transform, and the $MC^2$ network is switched into a butterfly network for the calculation of a Fourier transform. Such a system approaches the switched MIMD machine. $MC^2/64$ could therefore be classified as a switched network MIMD machine, but with the emphasis more on a networked MIMD machine.

## 1.6 Supercomputing in South Africa

The international tendency today is definitely towards the use of parallel machines to attain very high levels of performance. The Supercomputer market is expected to grow by 300% internationally between 1988 and 1991 while the traditional Mainframe market will only grow about 22% over the same period.

Sequential processors, such as the Intel processors, have dominated the computer market totally in the last decade. Even some of the large supercomputers have been based on such processors. These processors were not designed with parallelism in mind and Inmos saw the need to develop a microprocessor based on a new concept offered by the CSP model.

Since the transputer was developed specifically as a parallel processing processor, the interest in all transputer related topics is now expanding at a rapid rate. The T800 (with floating point capability) is a very high performance microprocessor with unique communications-based architecture. There is also a tremendous growth in the availability of transputer software. The transputer is the first microprocessor to be developed with parallelism in mind, and the interesting design concepts have definitely influenced the world of computers, providing new possibilities and challenges.

In South Africa we need to look at this new emerging field to satisfy our computing demands. At this stage the South African supercomputer user is totally dependent on the outside world which is (was) quite hostile. South Africa does not have the resources to compete in the supercomputer markets of the world, but perhaps a specific field might be identified in which we could develop local expertise. We could become leaders in this field, and thus also address our country's needs. The field in which we can most effectively address supercomputer performance, is the massively parallel environment, mainly because the building blocks of such a system are less expensive, and such a system is relatively easy to realise. The primary drawback of such a system is that the user applications which are available do not use the parallel architecture optimally.

The University of Stellenbosch developed the first local transputer card, to be plugged into a PC as host, in 1988. The CSIR produced its own card at nearly the same time, but the purpose of the CSIR project was different from the start. The purpose was defined as the building of parallel single-user workstations and in the end, a parallel multi-user supercomputer.

Considerable interest in the transputer has developed in South Africa, but most of the time the processor is used in embedded systems as it is a high-performance processor at a relatively low cost. The communications based architecture makes it suitable for such an environment.

The products that were developed by the $MC^2$ project team at the CSIR include transputer Worker Cards, $MC^2$ Clusters and $MC^2$ Mini Clusters, a 64 processor $MC^2/64$ multi-user machine, as well as peripherals such as A/D cards, graphics boards, LAN connections etc. At this stage the $MC^2$ products are being produced and marketed by a company that was launched for that purpose, CTS (Concurrent Technology Systems).

# 2 MC²

## 2.1 The MC² project

The MC² project was initiated to exploit new international technological developments and build a multi-user, low-cost, high-performance supercomputer for South Africa, centred around the transputer and based on the concept of reconfigurable transputer nodes. The international tendency is definitely towards the use of parallel machines to attain very high levels of performance.

The MC² design approach had to be modular with spin-off products along the way which could be used to satisfy smaller computational needs in South Africa. In the end, MC² must provide solutions to computational problems experienced in South Africa.

The main drawback of MC² in South Africa is the availability of reliable software that can exploit the architecture to provide better solutions than do other systems addressing the same needs. It is, for instance, not worthwhile to just port existing sequential software to the transputer as this will in most instances not provide the necessary performance increase. The existing software needs to be analysed to see which parts can execute in parallel, and the software must be rewritten to use the MC² architecture effectively. Parallel Programming is also much more complex and it is difficult to provide reliable and maintainable software.

Compatibility with existing international transputer systems is extremely important and the system was kept as compatible as possible with architectures used internationally so that existing international software could be used. It is very important to adhere to international standards or even emerging international standards, especially since our resources are restricted. By following these standards, we ensure compatibility regarding interfaces, software and hardware modules with international products.

## 2.2 History of MC$^2$

The MC$^2$ project was started in April 1988. The original project leader was Niel Viljoen. The project was to take place in three distinct phases :

Phase 1 : Initial hardware design [Completion Aug. 1988]

Phase 2 : Creation of a single user parallel processing workstation [Completion March 1989]

Phase 3 : Creation of a parallel processing distributed computer [Completion March 1990]

These three phases have been completed on schedule and the first MC$^2$/64 was delivered to the University of Stellenbosch. The second MC$^2$/64 is at this point in time (December 1990) operational at MIKOMTEK[1], CSIR. It is used specifically for research regarding upgrades on MC$^2$, and for the local development of transputer-based software.

Before we enter a detailed hardware description of the MC$^2$, it is first necessary to examine the transputer and Occam.

1.Division for Micro-electronics and Communications technology.

## 2.3 The transputer

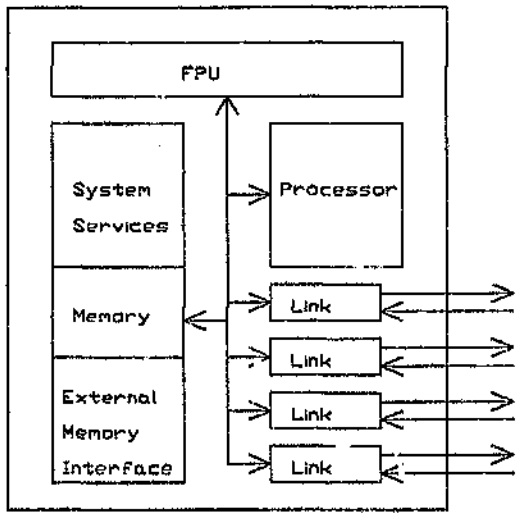
The Inmos[1] transputer[2] is a high-performance 32-bit processor with a unique communications-based architecture designed for parallel processing systems. For the purpose of this document, only the T800[3] transputer will be discussed. The core of the T800 is the 32-bit 10 (RISC) MIPS processor to which a 64-bit floating-point unit capable of sustained performance of 1.5 MFlops at 20 MHz (and 2.25 MFlops at 30MHz) has been added.

The highly integrated chip also carries a configurable memory controller, 4 Kbytes of high-speed RAM, four Inmos serial links that can operate up to 30Mb/sec, 32-bit wide 26MB/sec memory interface that can address up to 4 GBytes of external RAM linearly, and high-performance graphics support. The chip is packaged compact and fewer design chips are needed than for the state of the art machines. To provide maximum speed with minimum wiring, the transputer uses point-to-point serial communication links for direct communication to other transputers.

1.Inmos was originally a British semiconductor company, but is now part of the Franco-Italian company SGS-Thompson, Europe's second largest semiconductor firm.

2.The word "transputer" derives from "transistor-computer".

3.The previous generation of transputers, the T414, was capable of performing integer operations only and this was a real limitation in the high-performance computer market where floating point operations are essential. The T800 became the first chip with built-in floating-point capabilities.




Block diagram of the T800

A small core of about 32 instructions is used to implement simple sequential programs. In addition there are more specialized groups of instructions which provide facilities such as long arithmetic and process scheduling.

The transputer supports multitasking in hardware, but also virtually (multitasking on one transputer). In practice, both physically concurrent and virtually concurrent processes can run on a transputer network. The transputer's multitasking capability makes it possible for a programmer to write a concurrent program for a network of transputers even when the number of computing nodes in the network is unknown, or may vary. A programmer can develop and debug a program on one transputer that will eventually run on a network of transputers.

Transputers communicate only through links, allowing the memory to be dedicated entirely for use by the processor[1]. The transputer implements no memory protection, and cannot share memory with another transputer. Integration of the

1.If the transputer is used in large systems such as MC2, the system memory will be localised and associated with each processor making it a distributed memory machine.

processor and memory in the same device results in considerable performance gain as memory access does not require communication outside the device. The choice of point-to-point communication links also means that a system can be constructed from an arbitrary number of transputers. As the number of transputers in a system increases, the total processing power, memory bandwidth and communication bandwidth of the system grows proportionally.

The transputer architecture is designed for parallel processing on the distributed memory, message-passing model. Inmos designed the programming language Occam to take advantage of the computing model of the transputer. Occam has interesting features for synchronizing processes and for inter-processor communication. Both the transputer and Occam were derived from the CSP[1] model. This is a feature of the transputer which makes it revolutionary, as a total new design methodology was used to design the chip, enabling it to be used specifically for parallel processing.

1.Communicating Sequential Processes.

Some of the published benchmarks for a transputer compared with other state of the art systems are shown below. These are projected figures but still provides comparative data and are adequate for illustrative purposes.

| Processor | Mips | Dhrystones | MFlops | Whetstones | Chips |
|---|---|---|---|---|---|
| T414-20 | 10 | 11,345 | 0.1 | 660,000 | 1 |
| T800-20 | 10 | 11,345 | 1.5 | 4,000,000 | 1 |
| T800-30 | 15 | 17,017 | 2.25 | 6,000,000 | 1 |
| 386-16/387 | 4 | 6133 | ? | 1,800,000 | 2 |
| 386-25 | 5 | 10,000 | -- | -- | 1 |
| 68020-25/68881 | 5 | 7119 | ? | 1,500,000 | 2 |
| 68030-25/68882 | 6 | 9,500 | ? | 1,800,000 | 2 |

As mentioned, the transputer architecture simplifies system design by using point-to-point communication links. Point-to-point communication links have advantages over multiprocessor buses, for example, the fact that there is no contention for the communication channels, regardless of the number of processors in the system. There is no capacitive load penalty as transputers are added to the system, and the communications bandwidth does not saturate as the size of the system increases. Rather, the larger the number of transputers in the system, the higher the total communications bandwidth of the system.

To summarise, the transputer is a component designed to exploit the potential of VLSI. This technology allows large numbers of identical devices to be manufactured cheaply. For this reason, it is attractive to implement a concurrent system using a number of identical components, each of which is customised by an appropriate program. The transputer is, therefore, a VLSI device with a processor, memory to store the program executed by the processor, and communication links for direct

connection to other transputers. Transputer systems can be designed and programmed using Occam which allows an application to be described as a collection of processes which operate concurrently and communicate through channels. The transputer can therefore be used as a building block for concurrent processing systems, with Occam as the associated design formalism.

## 2.4 Occam

William of Occam was a fourteenth century philosopher who lived in Oxford and was best known for a Latin quotation known as Occam's razor: *Entia non sunt multiplicanda praeter necessitatem*, in other words KEEP IT SIMPLE.

Occam is known as the language of the transputer. Occam is not only a programming language for the transputer implementing concurrent processing systems but, also a subset of a software design methodology (CSP) which completely specifies the transputer. In fact, Occam predates the transputer in that it was used to design the chip; conversely, the transputer is the best component for implementing the programming language Occam. Occam is a deriviation of CSP, the work of Tony Hoare on Communicating Sequential Processes (CSP), which gives a mathematically based notation for specifying the behaviour of parallel processes. Occam is based on the CSP model of computation with features chosen to ensure efficiency of implementation. At the heart of Occam is its facility for expressing concurrency.

Occam is not an assembly-level language, but a high-level language. From the beginning Occam was geared towards non-Von Neumann architectures. The primary requirement was that it should support concurrency and communication, especially the point-to-point inter-processor communication of the transputer.

Occam enables a system to be described as a collection of concurrent processes which communicate with each other and with peripheral devices through channels. Occam programs are built from three primitive processes:

| | |
|---|---|
| v := e | assign expression e to variable v |
| c ! e | output expression e to channel c |
| c ? v | input from channel c to variable v |

The primitive processes are combined to form constructs:

| | |
|---|---|
| **SEQuential** | components executed one after another |
| **PARallel** | components executed together |
| **ALTernative** | component first ready is executed |

A construct itself is a process, and may be used as a component of another construct. Conventional sequential programs can be expressed with variables and assignments, combined in sequential constructs. Concurrent programs can be expressed with channels, inputs and outputs, which are combined in parallel and alternative constructs.

Each Occam channel provides a communication path between two concurrent processes. Communication is synchronised and takes place when both the inputting and the outputting processes are ready. The data to be output is then copied from the outputting to the inputting process, and both processes continue.

Occam differs in many respects from other programming languages such as Pascal, Modula-2 and Ada. These languages have a form of concurrency which assumes that the aim is to have a set of processes sharing a single computer. Occam comes from a different direction which lies in the conception of the transputer as a single-chip microprocessor intended for connection into collections of processing elements working co-operatively on a task. Occam is a language intended for programming such multi-transputer systems and the choice of features in the language has been motivated by the need for a distributed implementation.

An important objective in the design of Occam was to use the same concurrent programming techniques both for a single computer and for a network of computers. This means that the decision about how to distribute the system over multiple processors may be made fairly late in the development cycle. This enhances the fact

that the transputer supports multitasking in hardware and virtually. This is a very important concept, supporting the modularity of transputer systems. It means that a user can buy one transputer and do all the development of a system using only this transputer. This transputer can be used to simulate all the concurrency of the user's implementation. When debugged, the implementation can be loaded onto a network of transputers to get the necessary improvement in computer power. This supports the modularity design principle of the MC$^2$ systems.

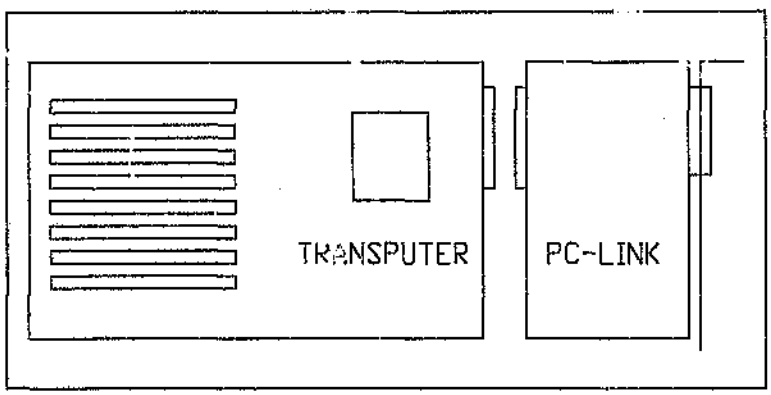
The use of Occam in the design and development of software leads to significant differences in approach when compared with the use of sequential languages. An Occam program is designed as a collection of communicating processes, with data flowing across channels between the processes; the format of the data flow is defined by the channel protocols. Novel challenges in the design of software are caused by the distributed nature of the system, such as the trade-offs between computation and communication, and the avoidance of deadlock.

## 2.5 Hardware of MC²

To understand the problem statement for this dissertation it is necessary to have a knowledge of the different hardware components developed for the MC²/64 system.

### 2.5.1 MC² worker card

The design approach of the MC² project is modular. The basic building block of MC² must therefore also be a module on its own.

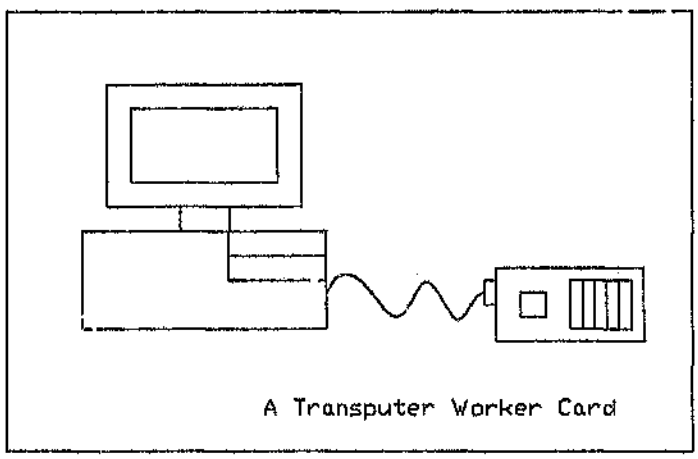



MC² Worker Card and the PC-Link Card used to interface the Worker Card with an IBM compatible PC

The basic building block of the MC² systems is the MC² Worker Card. Because the transputer was designed for systems on the distributed-memory message-passing model, each transputer has its own memory. The MC² Worker Card contains a T800 transputer and between 1 and 8 Mbyte of fast DRAM. The Worker Card is the module containing the transputer, its memory and the rest of the necessary logic.

The Worker Card contains all the logic and memory necessary to ensure a individual module which can be used on its own. IO is provided by an external resource such as the IBM PC. The PC-Link Card provides the module that

interfaces the transputer Worker Card, or the other MC$^2$ systems, with an IBM or compatible PC. The PC-Link Card is a very flexible module in that it provides two connectors which can connect to the transputer to provide the necessary interface. By connecting it as in the preceding diagram, the Worker Card - PC-Link Card combination can be plugged into the PC-bus as one card. The PC-link Card also provides a connector outside so that the Worker Card can be used outside a PC. In this case only the PC-Link Card is plugged into the PC and the Worker Card is externally connected to the PC-Link Card using a cable.

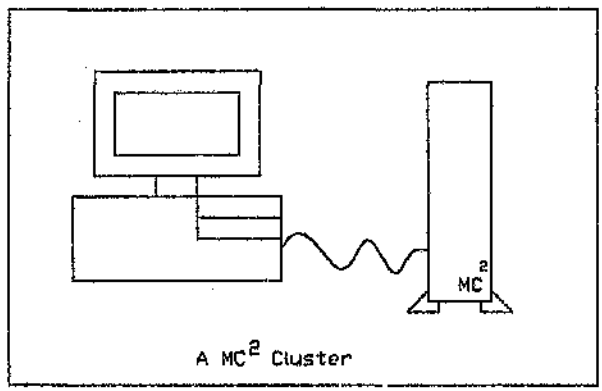



A PC-Link Card can also interface a Transputer Worker Card outside a PC

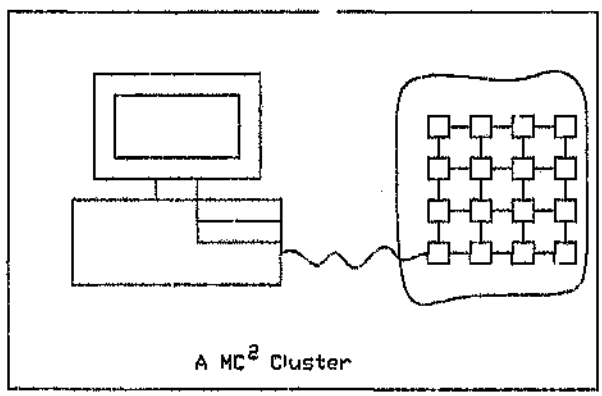
Normally, software is developed on the PC using a PC-based editor. The compiler and linker is downloaded with the software to the transputer to compile and link the software. Finally the software is run using an *afserver* which provides the interface between the software and the PC for IO.

### 2.5.2 MC$^2$ Cluster

The MC$^2$ Cluster provides the framework and motnerboard to house up to sixteen transputers (consisting of Worker Card modules) which can be connected to form any requested configuration.

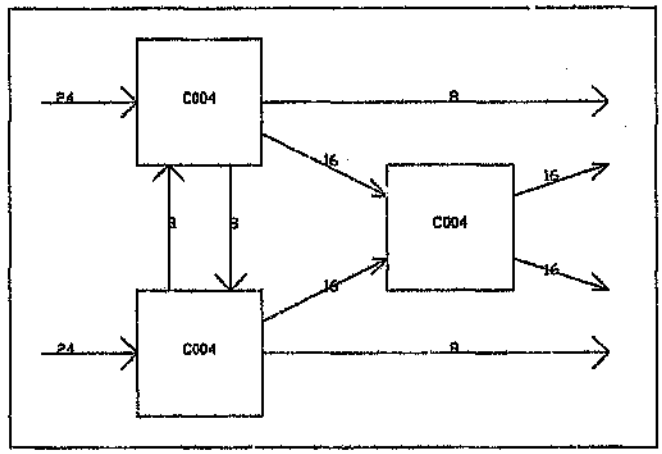



An MC$^2$ Cluster used as a single user workstation




A transputer network (as embodied in an MC$^2$ Cluster) connected to a PC as host

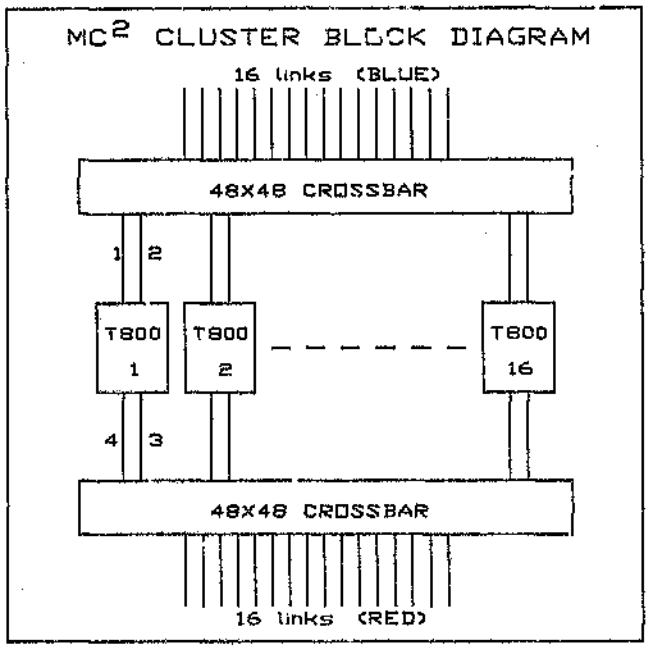
The MC$^2$ Cluster switch consists of two 48X48 crossbar switches built from three Inmos C004's.




The 48X48 Crossbar Switch

"The IMS C004 provides full switching capabilities between 32 Inmos Links. The IMS C004 can also be used as a component of a larger switches, such as the 48X48 Crossbar switch. The C004 operates at 10 or 20 Mbits/sec. The switch is programmable over a separate Inmos configuration link." (Inmos, IMC C004 Programmable Link Switch, 1987).

The two crossbar switches are located on both sides of the central row of transputers as indicated in the block diagram below. Two links from each transputer go into each switch. Configuration of the network is done using the unique multi-graph colouring scheme that is explained in the section on cluster configuration. Each $MC^2$ Cluster is completely configurable meaning that any requested network configuration can be mapped onto the hardware.



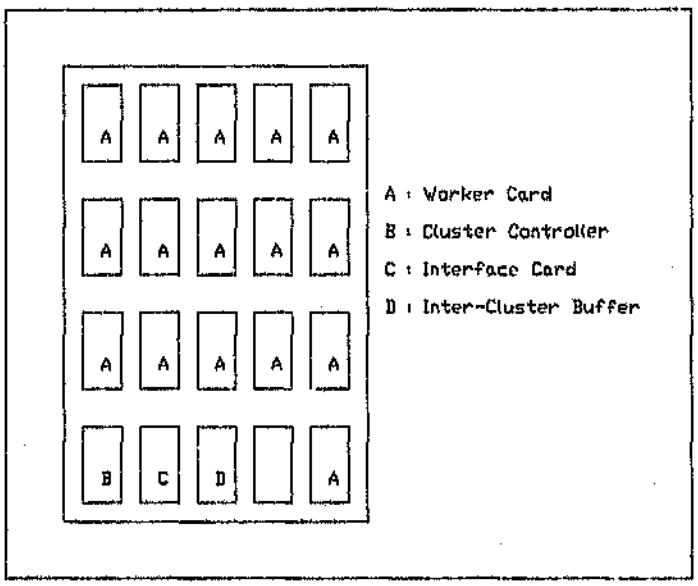

MC2 Cluster block diagram

Each MC2 Cluster has 32 links emanating from it, 16 from each one of the two 48X48 crossbar switches. These are the links that are used to connect Clusters together to form the larger machines. They also connect peripheral devices to a cluster.

The IO is again provided by a PC using the PC-Link Card. Special IO cards can also be plugged into the system, providing their interfaces are the same as the MC2 Worker Card (described in the previous section)[1]. The PC however, provides cheap IO.

1.An example is the high speed, high resolution Graphics Worker Card that was also developed by the CSIR. Such a card can be plugged into one of the slots of the MC2 Cluster providing high speed output to a high resolution graphics screen.

The physical construction of a MC$^2$ Cluster consists of a cabinet housing the motherboard as well as the power supply. The motherboard contains the 16 slots for each one of the transputer Worker Card modules that can plug into the system. The cards slide into the slots and are plugged into the motherboard. Two other slots are provided for the cluster controller[1] and the interface card connecting a cluster to the PC-Link Card in the PC. Another slot provides access to the 32 inter-cluster links released by an MC$^2$ Cluster. This is the slot which is used when clusters are connected together to form the larger MC$^2$ systems.




The MC$^2$ Cluster Cabinet

Software is written, compiled and linked as described in section 2.5.1. A user uses the cluster in single workstation mode by first switching his requested

1.The cluster controller is a MC$^2$ Worker Card which is transparent to the user, but which is necessary to control the MC$^2$ Cluster switches. In a single user workstation, it is possible for the user to use this transputer as a 17th transputer, but this transputer cannot be fully switched as certain links are used to control the cluster switches.

transputer network in the cluster using configuration software provided with the MC$^2$ Cluster. The users' software is then downloaded onto the transputer network using the *afserver* to provide IO. Another configuration can be switched by just downloading the new configuration.

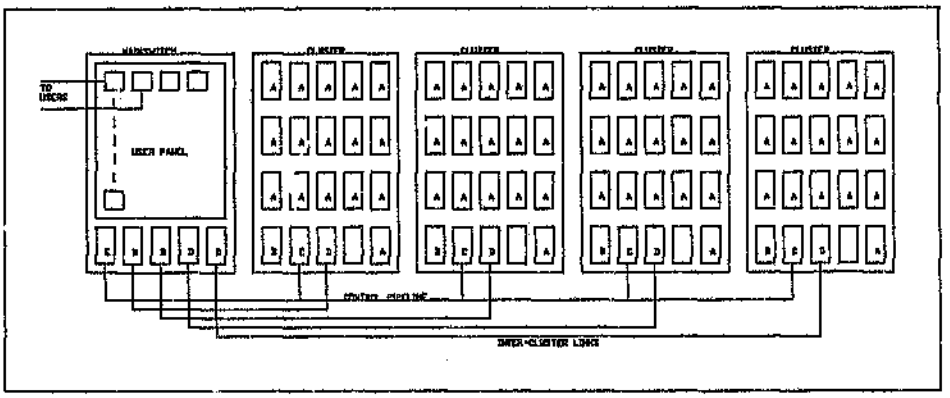
### 2.5.3 MC$^2$/64

The MC$^2$/64 is a 64-processor, distributed memory, multi-user machine. Multiple users can log into the machine each requesting a domain of transputers to work with. Once a user has been allocated his domain, he can switch and control his domain, and execute his software without influencing other users. (The control software allowing users to switch and control their domains, is discussed in chapter 5.)

The MC$^2$/64 system consists of MC$^2$ Clusters connected together to form a 64 processor multi-user system. To keep the system configurable, the different MC$^2$ Clusters need to be connected together in such a way that any two transputers in the system can be connected without influencing other connections made already, and without blocking the system in such a way that no further connections can be made. Furthermore, the performance of supercomputers depends very much on the allowance of high-speed communication between processors or clusters of processors. This implies that the design of the central switching unit is of the utmost importance as this will determine the configurability of the system, as well as influence the speed of communication between transputers in different clusters. (The central switch and configuration algorithms of MC$^2$/64 is discussed in chapter 4.) The current MC$^2$/64 central switch or main switch consists of a modified Clos network. Another two 48X48 crossbar switches are added to provide access to users.

As mentioned, the main building blocks of the multi-user systems are the MC$^2$ Clusters with up to 16 transputers each. This implies that the MC$^2$/64 model will

consist of four MC$^2$ Clusters. The MC$^2$/128 will have 8 MC$^2$ Clusters and the MC$^2$/256 16 MC$^2$ Clusters. The rated maximum performance of these units will be 96, 192 and 384 MFlops respectively[1].



The physical construction of MC$^2$/64. This diagram shows the four MC$^2$ Clusters as well as the Main Switch Cluster. A : Worker Cards; B : Cluster controllers; C : Interface cards; D : Inter-Cluster buffers; E : System controller.

The central switch is housed in a cabinet which is the same as those of the MC$^2$ Clusters. The central switch motherboard[2] contains the central switch as well as all the socket connectors for the inter-cluster buffers bringing the inter-cluster links from each cluster to the central switch. These are the links used to connect clusters together, as well as to connect users to the system. The two external or user 48X48 crossbar switches allowing users access to the machine, are also housed in the main switch cluster. Furthermore, a user panel with the user connectors is provided into which a user may physically plug his cables allowing his PC to be used as a IOP[3] to MC$^2$/64.

1.To put these figures in perspective, the CRAY 1 supercomputer has a performance of approximately 100 MFlops. These figures will increase by approximately 50% when (if) the newer transputers become available. (Inmos has been very slow with the release of the newer transputer versions.)

2.The Central Switch motherboard is a ten layer board containing eight signal layers and an earth and VCC plane.

3.IO Processor

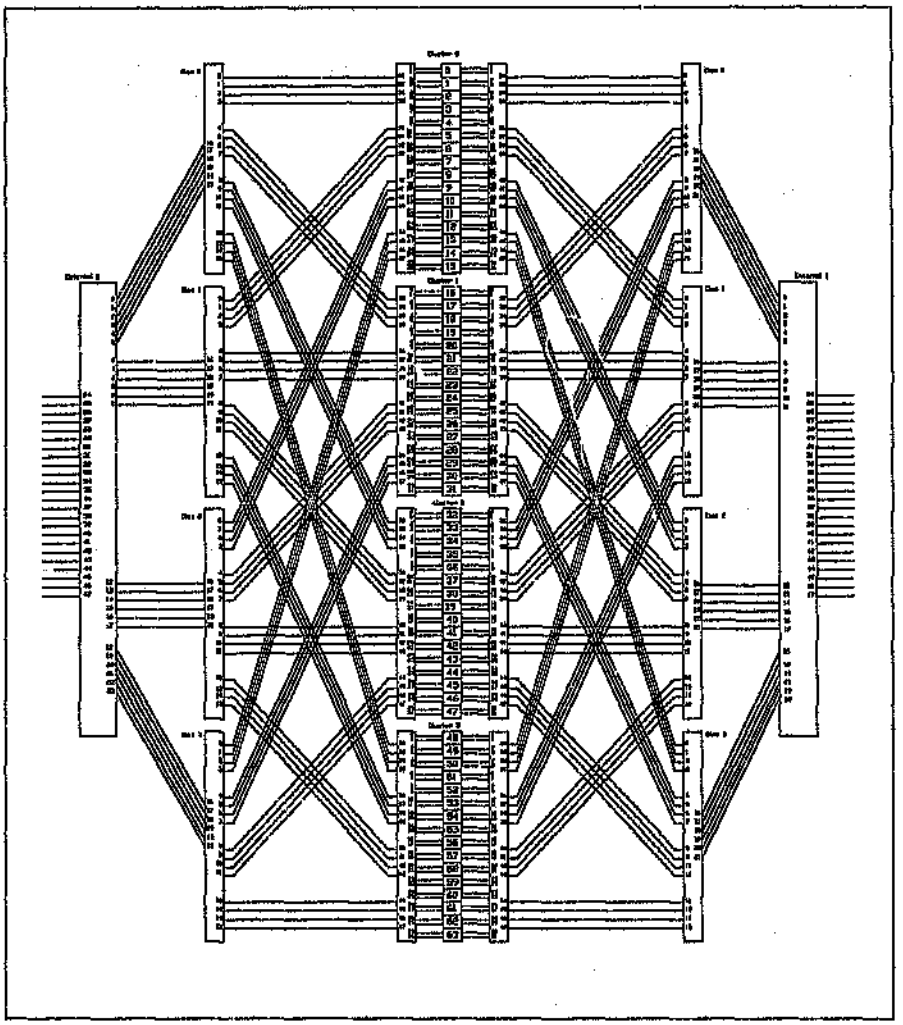
The MC$^2$/64 system is controlled by the control pipeline. The control pipeline consists of five transputers : the system controller and four cluster controllers. The system controller is housed in the Main Switch Cluster while the cluster controllers are housed in the different MC$^2$ Clusters.

The Main Switch Cluster also contains the two external switches (or 48X48 user buffer switches). These are the switches which allow users access to MC$^2$/64. One switch exist per Euler colour. A user panel provides users with the physical connectors for access to MC$^2$/64. User PCs used as IOPs to MC$^2$/64 have to be fitted with the standard PC-Link Card. Three links on the PC-Link Card are used to provide access to MC$^2$/64. Links 0 and 1 are used to allow a user access to his user domain (one link per Euler colour) and link 2 is used to send domain control commands to the DMS[1] running on the control pipeline[2].

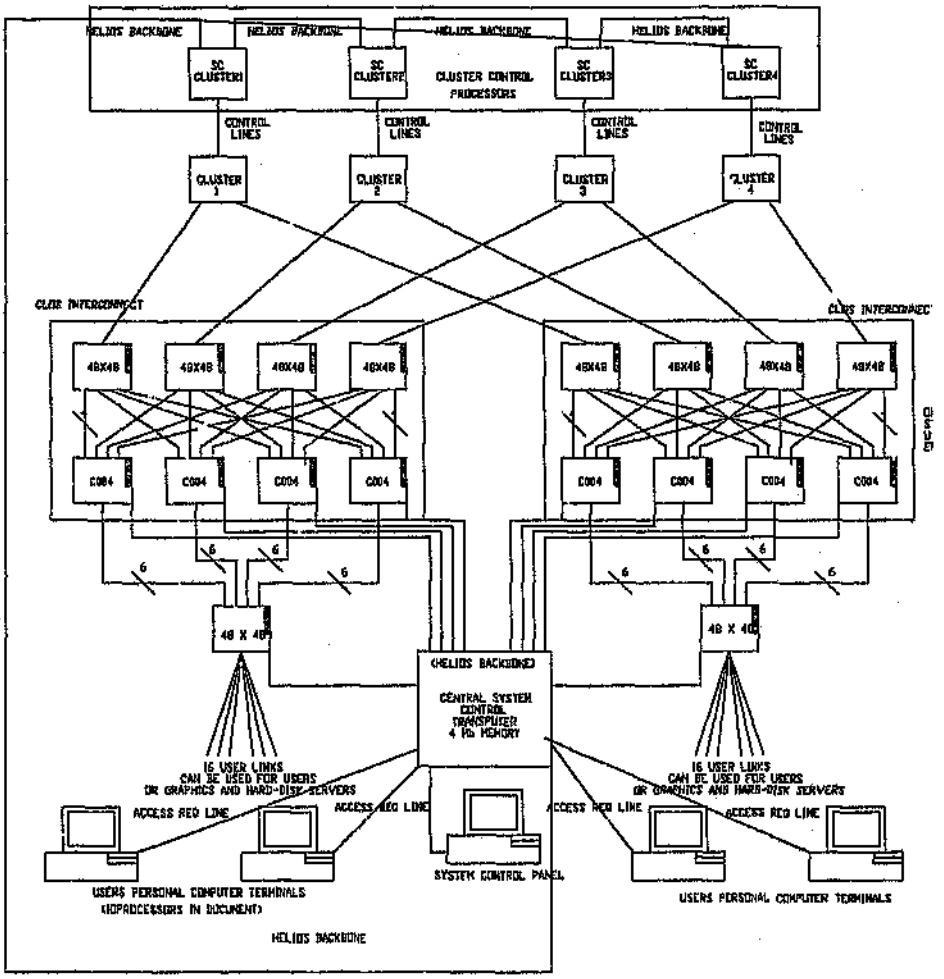
All links emerging from MC$^2$ Clusters are buffered using RS422 buffers and twisted pair cables. This includes the control pipeline, all the inter-cluster links and all the user links. This maintains reliable communications, even over longer distances.

1.Domain Management System

2.The MC$^2$ Lan card will allow users access to MC$^2$/64 through a LAN. This option will replace the current user-panel with the physical access of cables from the user IOPs.



This diagram is the link interconnection diagram of MC$^2$/64, showing all the transputer links and how they can be switched through the system.




A block diagram of MC²/64. This diagram shows the two identical switching networks of the MC²/64 system, as well as the control pipeline from the system controller to each MC² Cluster. Each user IOP uses a link to communicate control commands to the system controller enabling a user to switch and control his domain of transputers.

The preceding diagram also shows how the system console is connected to the system controller (or central switch control transputer). The system console consists of the console PC running the DMS[1] under Helios on the control pipeline. The DMS is the control software of MC$^2$/64.

1.Domain Management System

# 3 MC² CLUSTER CONFIGURATION

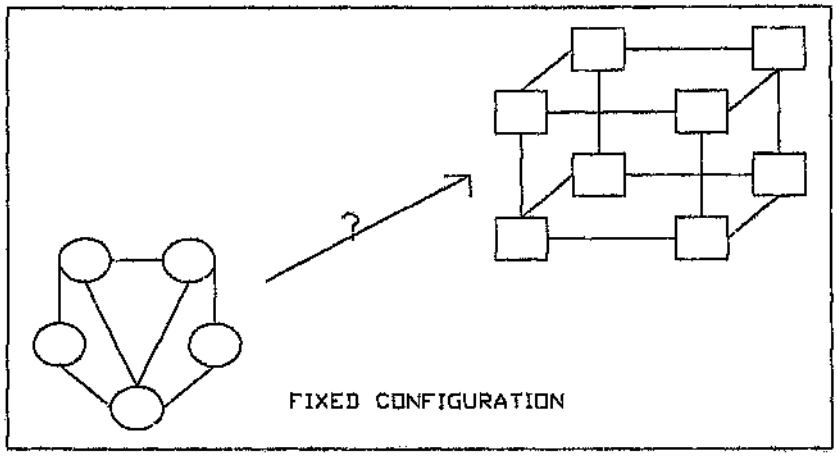
## 3.1 MC² cluster design

The MC² Cluster must adhere to the design concepts of MC² namely modularity and configurability. (Configurability means that a user must be able to switch any network configuration in a cluster.)

The trend in massively parallel systems is to use a fixed network of processors and to customise the software to get the maximum performance out of the system. If a fixed network system such as is incorporated in the hypercube is used, the *mapping problem* immediately raises its head. The problem of matching a user map to a system map is defined as the traditional mapping problem, and the solution to this problem always comes back to simulated annealing and related solutions which are computationally very expensive, and which we want to avoid.

The mapping problem is generally stipulated as follows: "*Is there a mapping of a system of communicating processes onto a processor network such that the neighbouring processes are assigned to the neighbouring processors?*" This problem is equivalent to the graph-isomorphism problem which is NP-complete, i.e. there is no better method of finding the minimum solution than to calculate all the possible solutions. However, the number of possible solutions is an exponential function of the number of tasks. Two graphs are said to be isomorphic to each other if there is a one-to-one correspondence between their vertices and edges such that the incidence relationships are preserved.

But the mapping problem is summed up the best by Sahid Bokhari : "It appears unlikely that an efficient exact algorithm for the general mapping problem will ever be found. Research in this area must concentrate on efficient heuristics that find good solutions in most cases." (Bokhari, 1981)

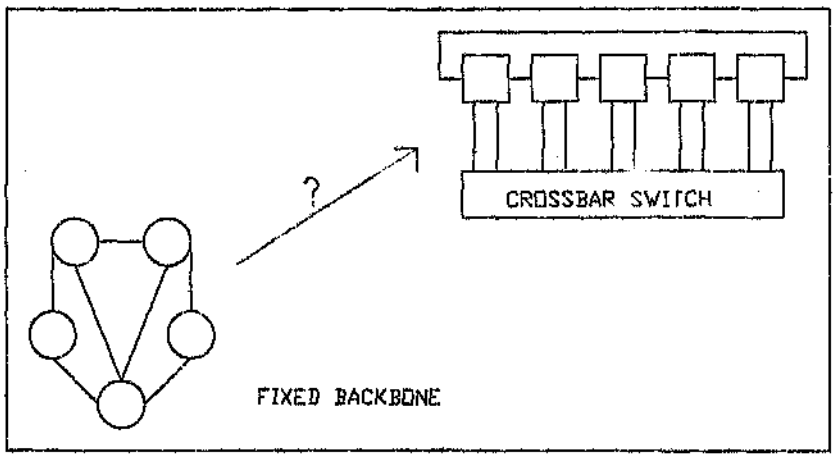



The mapping of a requested user network onto a fixed configuration

After Inmos announced the C004 link switching device, many different designs and implementations of link switching networks were published (Hill, Inmos technical note 19). The Inmos C004 provides switching of 32 input wires to 32 output wires with full resynchronisation of the signals at rates up to 20 Mbaud. It is itself programmable over a link. Link signals may be propagated through any number of C004s with some delay but no degradation. None of the commercial machines such as the FPS T-series(USA), the Meiko Computing Surface or the Esprit P1085 use the C004. This is largely because of the late announcement of the C004 after these companies had already committed themselves to other devices.

Several existing switching network designs (for example the UK Alvey Parsifal machine) implement a fixed backbone using two links on each transputer and switch only the remaining two links. It has been claimed incorrectly (Inmos technical note 19) that such networks provide complete connectivity. This is not the case as a fixed

backbone network will require a Hamiltonian cycle[1] through the graph in order to be able to switch the network. All connected graphs do not necessarily contain Hamiltonian circuits. To extract a Hamiltonian circuit from a arbitrary transputer network is also computationally very expensive and the problem increases with the number of transputers.




The mapping of a requested user network onto a fixed backbone

The Esprit P1085 project team developed a switching network which, using a novel

1.A Hamiltonian circuit is a cycle in a graph that passes through each vertex exactly once. There is no simple rule for determining the existence of Hamiltonian circuits as there is for Eulerian circuits. The *travelling salesperson problem* is the problem of finding the Hamiltonian circuit with the least possible weight when the edges are assigned positive weights (which could represent distance for instance). For large sets of vertices, the problem is sufficiently complex so that no known algorithm computes the best solution in reasonable time.

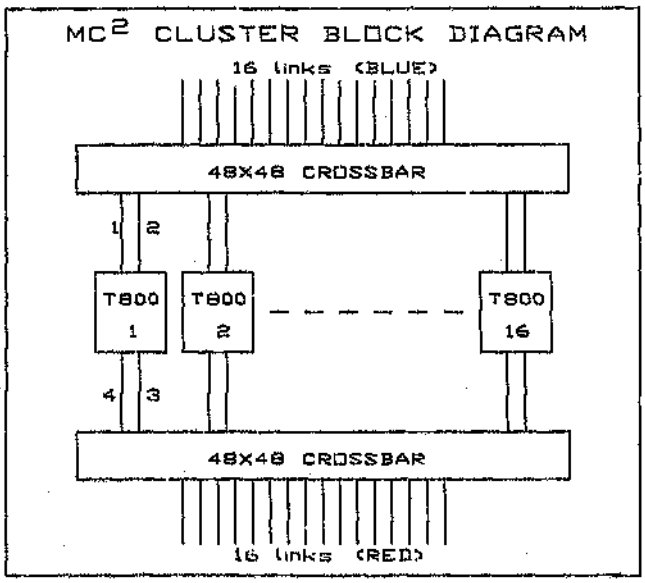
algorithm based on a Eulerian cycle[1] through the graph of transputer link connections, provides complete universal routing of all transputer graphs and multigraphs (Nicole et al., 1988). Multigraphs have more than one link between some pairs of transputers. For the Esprit project, a special switching element was developed to provide complete configurability of networks of up to 72 transputers using just eight switching components.

The $MC^2$ project team decided to design a cluster that will not only switch two links as in the fixed backbone structure, but will exchange the fixed structure with another crossbar switch. What finally emerged was the $MC^2$ Cluster as shown in the following. Such a system is completely configurable when using the Esprit algorithm as will be shown in the next section (see section 3.2). It must be noted here that although the architectures of the $MC^2$ Cluster and the Esprit architecture are very similar, the architecture of $MC^2$ evolved separately from, and almost simultaneously as that of the Esprit system.

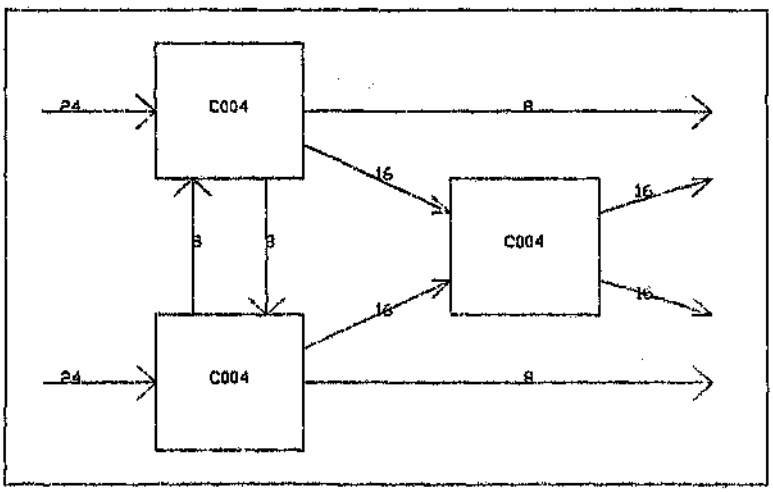
The block diagram of the $MC^2$ Cluster shows the two 48X48 crossbar switches comprising the switching network of each cluster. Up to 16 transputers can be accommodated, each having two links switched by each one of the two crossbar switches (refer to section 2.5.2).

1.The Konigsberg bridges problem is the problem of seven bridges connecting two islands in a river and the banks of the river. Euler tried to find a path that crosses each bridge exactly once and he could not find one. Euler formulated his rule as follows : " A connected graph with at least two vertices has an Eulerian path if and only if there are 0 or 2 vertices of odd degree. The path is a cycle if and only if each vertex has even degree."

A graph is said to be connected if there is a path between every pair of vertices in the graph.




MC2 Cluster

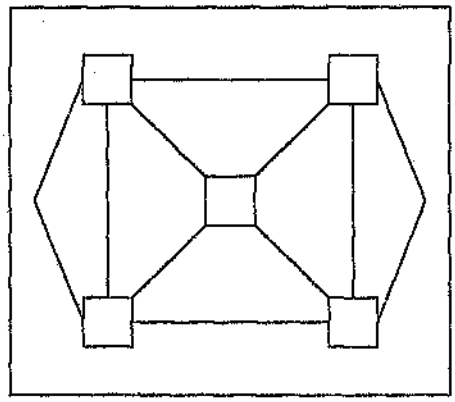



The 48X48 Crossbar Switch. This switch has been built using three C004 Link Switch Devices from Inmos.

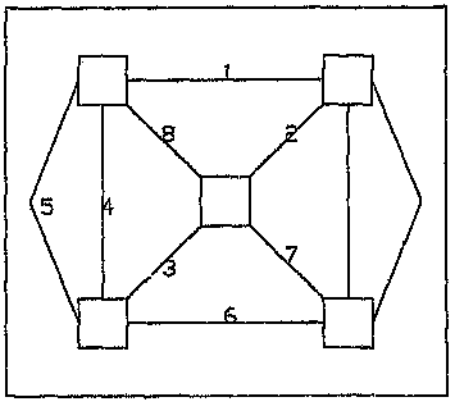
## 3.2 Two-colour Euler algorithm

Graph theory provides a mathematical framework in which to incorporate transputer networks to investigate their properties. We can regard a transputer network as a graph in which the transputers are vertices and the links are edges. We will have a regular (all vertices have same number of edges) 4-valent (a vertex has 4 edges as a transputer has 4 links) graph. We allow multiple edges between vertices (multiple links between transputers) and so we have a multigraph.

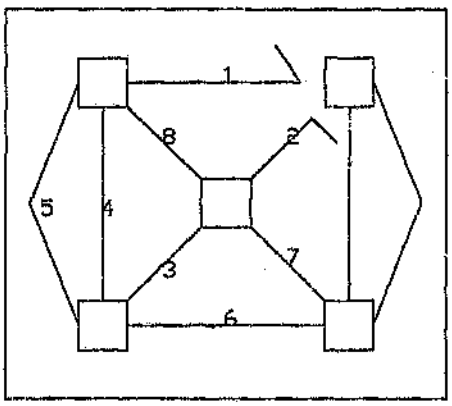
Eulerian cycles are cycles that visit every edge exactly once. All connected graphs and multigraphs with an even number of edges per vertex have Eulerian cycles. It is very easy to construct a Eulerian cycle out of a graph or multigraph. The proposed algorithm of the Esprit team is : *Start at any vertex, and simply walk around the graph. As each vertex has an even number of edges, this walk will never stop at a dead end, but will continue until we have eventually visited our starting point sufficiently many times for the cycle to close there. If all edges have been visited, all is well, if not, simply open the cycle at the vertex with unvisited edges and set off along one of them. Repeat until all edges have been visited. After having built our Eulerian cycle, we can use it to perform a two-colouring of the edges of the graph. Simply follow along the cycle, colouring edges alternately with the two colours as they are encountered. There will be exactly two edges of each colour at each vertex* (Nicole et al., 1988).
These two colours provide the links that are to be switched by each of the 48X48 crossbar switches of the MC$^2$ Cluster. The algorithm can be displayed graphically as shown in the following figures :



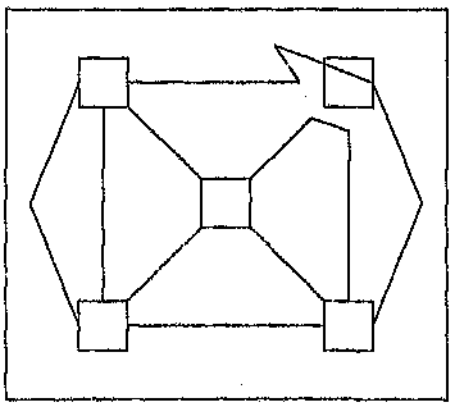
The requested network



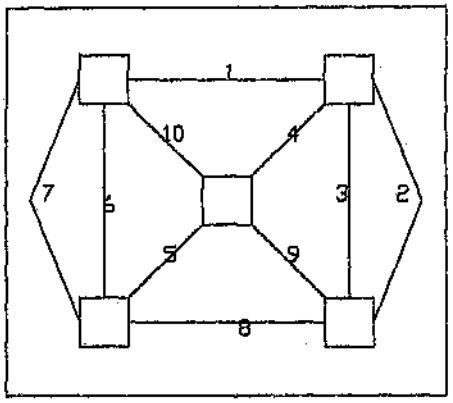

The first cycle has been constructed. We have closed the cycle at the starting transputer but all edges of the graph have not been visited.



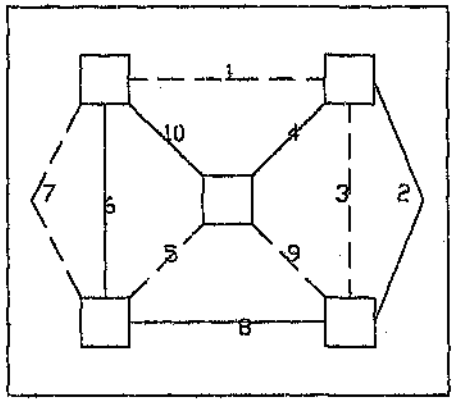

Break the cycle at a vertex with unvisited edges.



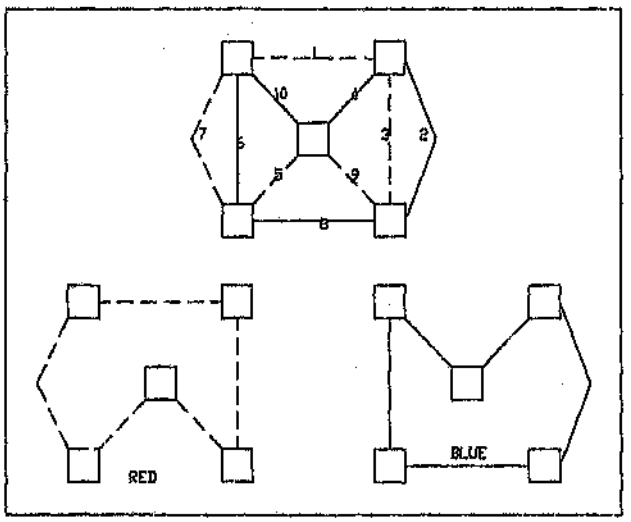
Set off along the unvisited edges and complete the Euler cycle.




The completed Euler cycle

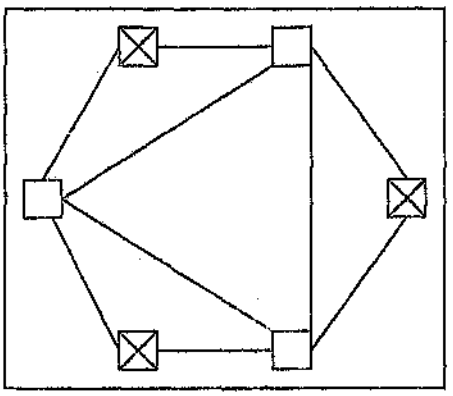



Colouring of the Euler cycle. Colour all even edges one colour and all uneven edges another colour.

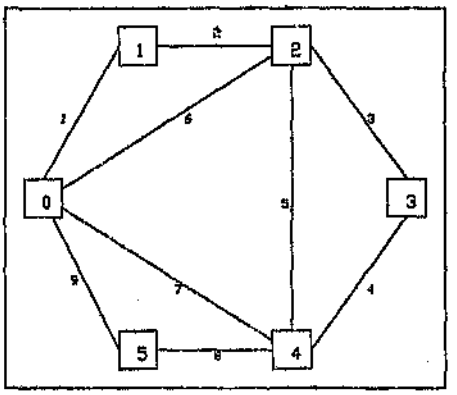



The two Euler cycles extracted from the coloured graph for switching by each one of the two 48X48 crossbar switches of the **MC²** Cluster

During the implementation of this algorithm the author encountered some interesting problems and possibilities, the first of which is that which occurs when all the vertices of a requested switching graph do not have four edges connected. An example of this, and the Euler cycle extracted, is shown in the diagram:

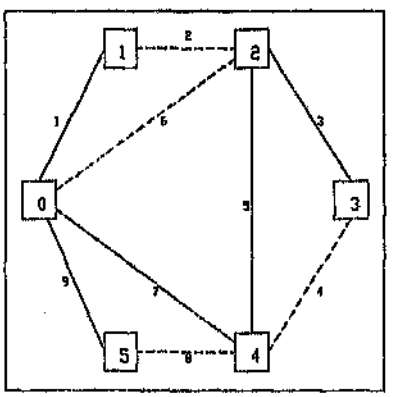


The requested switching network



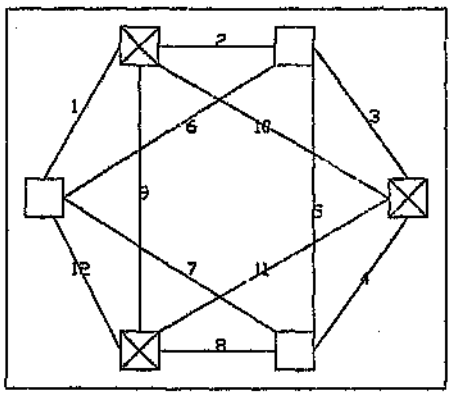

The Euler cycle extracted from the graph

If we colour the Euler cycle starting at transputer 0, we get a vertex with three edges of the same colour connected to it. This, of course, is not switchable by the chosen $MC^2$ Cluster configuration.

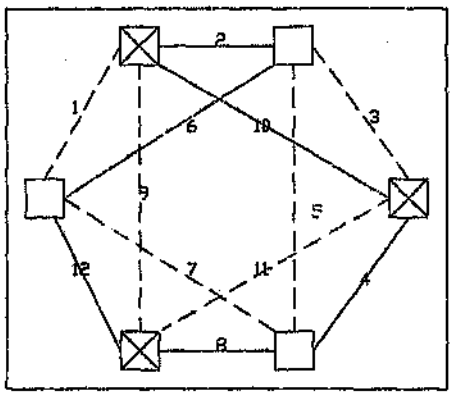



The coloured Euler cycle with three edges
of the same colour connected to transputer 0

A solution to this dilemma is to add links so that each vertex has four edges as shown in the next graph. It is in any case necessary to add links if there is a vertex in the graph without an even number of edges, because a graph without an even number of edges per vertex does not contain a Eulerian cycle. In the first graph the links have been added and the Euler cycle constructed. In the second graph the colouring and therefore the extraction of the two separate Euler cycles has been done. After the two Euler cycles have been extracted from the graph, the added links can be removed so that only the original graph requested by the user, is switched.

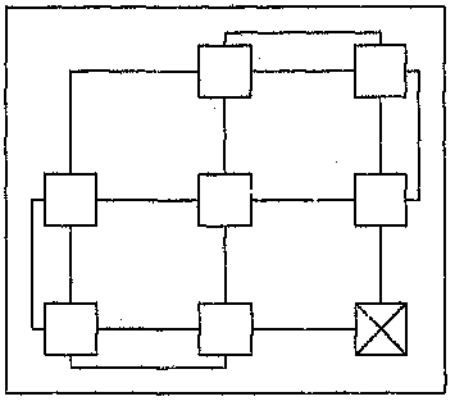



The original graph, but with links added so that each vertex has four edges

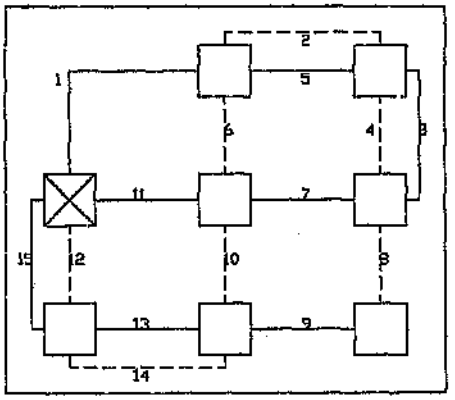



The two Euler cycles have been extracted from the original graph. The added links can be removed at this stage so that only the graph requested by the user is switched.

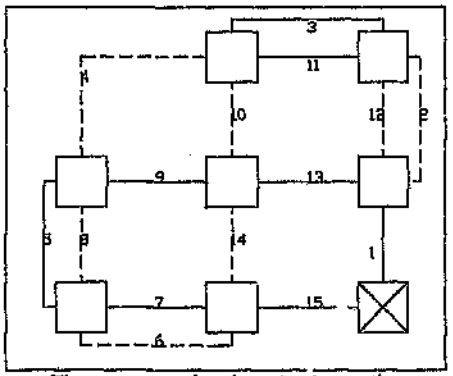
It is sometimes possible to get a graph where no links can be added (except to connect two links of the same transputer) as in the graph in the next diagram. The colouring of such a graph will also always give the wrong result if one does not start at the vertex with only two links connected. This is because the graph has an uneven number of edges, and, because the purpose of the exercise is to extract two Euler cycles out of the given graph, one must ensure that the start and end edge of a Euler cycle falls on a specific vertex. The solution is therefore to start at the vertex with only two edges connected as both the edges of this vertex must fall in the same Euler cycle.



A graph where no links can be added to the vertex with only two edges

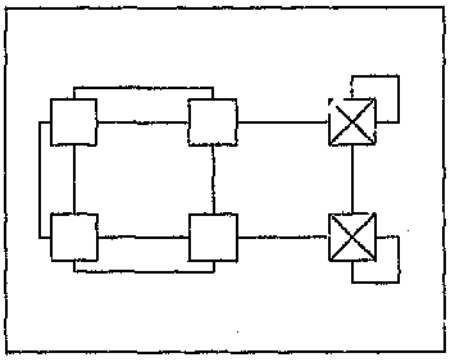



Colouring of such a graph starting at the transputer marked with a cross

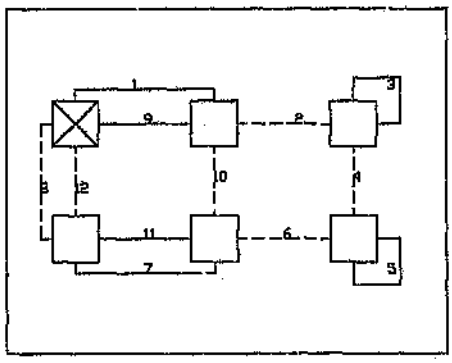



The correct colouring starting at the transputer marked with a cross

Another interesting example is networks where two links of the same transputer are connected (this is also a solution to the case mentioned above). The Esprit algorithm and the architecture of the MC$^2$ system supports such a network without any problems.



A network with two transputers that have links that are connected to each other (Marked with an X.)




The graph with the two Euler cycles extracted, starting at the transputer marked with an X.

## 3.3 Choice of links

A user does not have freedom of choice connecting the links in his requested network, as it is impossible to connect, say link 0 to link 1 of transputers in the $MC^2$ Cluster. This is because only links 0 and 2 of all transputers in a $MC^2$ Cluster are switched by a 48X48 crossbar switch, and links 1 and 3 by the other 48X48 crossbar switch (refer to the block diagram of a $MC^2$ Cluster in the previous section).

A rule of thumb for users is to connect even links to even links and odd links to odd links. This is the same principle as used in the two-colour Euler algorithm where even links are coloured one colour and odd links another colour.

If a $MC^2$ Cluster is used as a single-user workstation, a user can have the problem partially solved by connecting the links emerging from the cluster crossbar switches, or the inter-cluster links. By connecting the inter-cluster links, a user connects the two cluster crossbar switches and he can therefore connect even links to odd links. This is only a partial solution as only 16 links are released from a cluster crossbar switch, and 32 transputer links are switched.

So far most users did not find this restriction a severe limitation. Users of large parallel systems usually want to map a software graph onto a hardware graph, and the low-level details of the hardware graph, such as link numbers, are not important. Especially if tools exist that make these processes (mapping onto the hardware and link number selections) transparent to the user.

This restriction can, however, be a disadvantage. It can mean that software ported from other machines will not run. This can be a great disadvantage as one of the main features of MC$^2$ should be its compatibility with international systems so that their software can be used on this system as well (without alteration). Fortunately, it seems that so many possible architectures were developed that most software vendors chose to leave low-level configuration either to the user, or write software that automatically does network detection and in this way configure the application software. So far we have not found any application software that does not run on MC$^2$.

## 3.4 Conclusion

The algorithm that was developed by the Esprit project team aided the MC$^2$ project team with the design of a totally configurable MC$^2$ Cluster that can be used as a single user workstation, but also a component for larger MC$^2$ systems. The switching algorithm of the Esprit project team was enhanced so that it can be used to configure MC$^2$ Clusters. This algorithm provides complete configurability of MC$^2$ Clusters enabling a user to switch any requested network configuration onto a cluster. A user, however, does not have freedom of choice of links, as even links may only be connected to even links, and odd links to odd links.

The advantage of being able to use the Esprit algorithm lies in a much less complicated MC$^2$ Cluster design. To be able to switch all the links in a cluster containing 16 transputers, one needs a bigger switch than a 64X64 crossbar switch, as a cluster must release links for external connections. A 64X64 crossbar switch will only provide complete connectivity of all transputer links. Such a switch is much

more complex to design and realise than the 48X48 crossbar switches of the current MC$^2$ Clusters. The number of C004's used will increase tremendously, and this will also cause the additional problem of a substantial delay through the C004's (see Appendix A, chapter 7.1). If such a switch is built without using C004's, the switch must be able to regenerate and synchronise the link signal as the transputer links are sensitive to the signal quality, and most custom-made switches used in other machines internationally, showed problems regarding the quality of the link signal.

# 4 MC²/64 CONFIGURATION

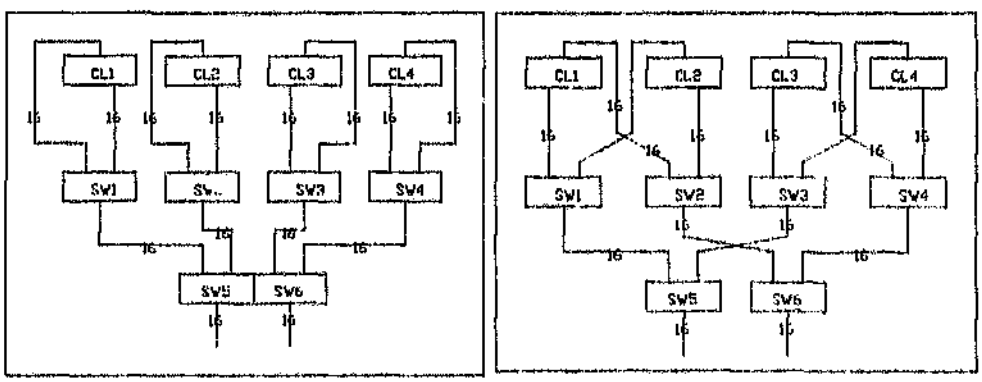
This section will give an overview on the MC²/64 inter-cluster switch, the algorithms used for the configuration of MC²/64, and the simulation of the configurability of MC²/64.

For this dissertation, *Configurability* is defined as the possibility to always connect any transputer with unconnected links to any other transputer in the system with unconnected links. *User Configurability* is defined as the configurability a user experience when using MC²/64.

## 4.1 The MC² Inter-cluster switch

To build any of the multi-user MC² machines, such as the MC²/64 machine, a central coupling unit is needed. This central coupling unit is necessary to provide an integrated system using the individual MC² Clusters. The central coupling unit must also handle the IO resources and other peripherals, and give users access to the machine.

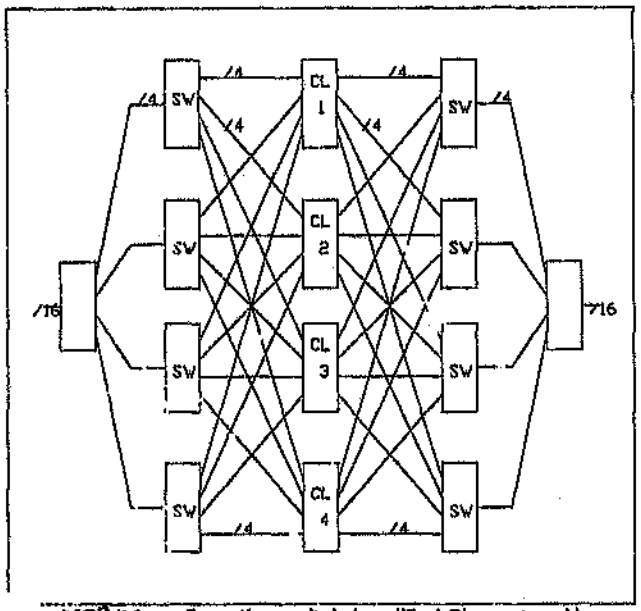
Possible interconnection networks that were studied for implementation are shown in the next two diagrams :



The two networks are completely blocking after a few connections. "Blocking" means that after a few connections have been made, it is not possible to add connections as all the available paths through a specific switch for example, have been used. This is not desirable as a user of $MC^2/64$ must be able to connect any transputer to any other transputer without connections being influenced or the influencing of other network configurations.

The next network considered was a modified Clos network shown in the next figure. This network is not blocking. There will always be a path between two transputers in this network as there are many different paths for routing a connection between two transputers. The only limiting factor is the number of inter-cluster links in the system, and this concers the number of links emerging from a $MC^2$ Cluster, and is not a factor relating to the network. As mentioned in chapter 2, section 2.5.2, a $MC^2$ Cluster releases 32 links (16 from each of the 48X48 crossbar switches) for inter-cluster connections.

One big advantage of the modified Clos network is the limited number of hardware components that are necessary to realise such a network. Each Clos crossbar switch for the $MC^2/64$ need only switch 16 inter-cluster links and provide the external IO. An Inmos C004 can therefore be used, and the only components needed for a complete Clos network are four C004 switches (per Euler colour), which is less than the number required for any of the other networks that were investigated.




$MC^2/64$ configuration switch (modified Clos network) incorporated in the $MC^2/64$ design. The Clos network provides several paths between any two transputers in the system, providing maximum switchability with the minimum number of hardware components. This diagram also shows the external switches used to connect an external link to the system.

From this diagram it can be seen that two identical Clos networks exist, one for each Euler cycle half. Another important switch in each Euler half is the external switch. This is the 48X48 crossbar switch allowing external connections to be made to transputers in the clusters. This is the way a user's domain will be connected to his IOP. These switches are constructed in the same way as the 48X48 crossbar switches, from three Inmos C004s.

Six links from each one of the Clos crossbar switches are routed to the external switches in the current $MC^2/64$ systems, giving a total of 24 links. A total of 24 users can be allowed to the system, as this number of users would take up all the available external connections. A user is allocated one link from each one of the Euler halfs (or from each one of the two external switches).

The external switches are a very important component in the system allowing flexibility for user connections, but more important, the external switches allow the Clos switch to be used optimally for inter-cluster connections (which is its most important function). If users were to be connected directly onto the Clos crossbar switches, the functionality of the Clos switch would be severely hampered as the Clos network would then have to take into account the location of the users when switching inter-cluster connections. If the user request could be switched before the user IOP links are connected, this would solve the problem. This means that the user request must firstbe switched, and then afterwards, when all connections have been made, one specific Clos crossbar switch will be the switch through which the external connection is to be made. The specific user can then be connected to this Clos crossbar switch. It is not practical to run around with the user IOP connectors every time a user wants to switch his configuration, and therefore the external switches has been introduced to implement the necessary connection of a user IOP to the specific Clos crossbar switch.

For the rest of this section, only one Clos network (in other words, one half of the total $MC^2/64$ switch) will be discussed as the two networks for each one of the Euler halfs are identical.

## 4.2 Number of Clos crossbar switches

The number of crossbar switches used in a Clos network is determined by the number of links released by a cluster, or the number of links routed to a switch from the cluster. If a system is built from clusters releasing 16 inter-cluster links (per Euler cycle), 16 Clos crossbar switches are necessary. If the Clos crossbar switches are still C004s, a fully connected system will consist of 32 clusters each releasing 16 inter-cluster links, and 16 Clos crossbar switches, each routing one link to each one of the 32 clusters. Clusters releasing 32 links per Euler half will need a 32 Clos crossbar switch to ensure a fully connected system.

The current MC$^2$/64 uses only four Clos crossbar switches (per Euler half) as four of the 16 inter-cluster links are routed to a Clos crossbar switch. The number of Clos crossbar switches is therefore in this case determined by the number of inter-cluster links routed to such a switch.

The number of Clos crossbar switches = (number of inter-cluster links released by a cluster per Euler half) / (number of links routed to the Clos crossbar switch per cluster).

For the current MC$^2$/64 system: number of Clos crossbar switches = 16/4 = 4.

## 4.3 Expandability

By expandability is meant the size to which the system can be expanded. The expandability of the system is determined by the size of the Clos crossbar switches as the Clos crossbar switches must connect to every cluster. The number of clusters in the system is therefore determined by the size of the Clos crossbar switch.

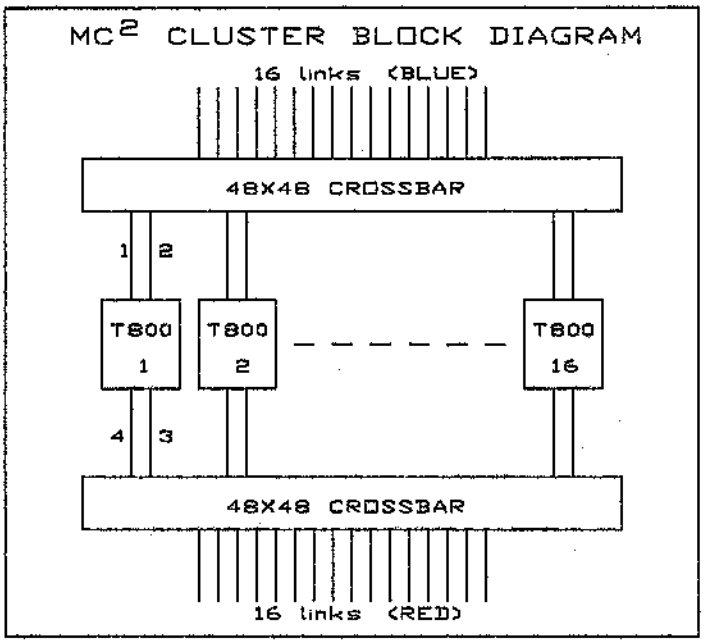
The Clos switches in MC$^2$/64 are Inmos C004 switches and are therefore 32X32 crossbar switches. The maximum number of clusters in a system is therefore 32, each cluster routing one link to each one of the Clos switches. This will give a fully connected system. Such an interconnection network however, leaves no links

unoccupied at the Clos crossbar switches for external connections providing the IO, and less than 32 clusters will have to be used to allow users access to their transputer domains. Some links must be available at each Clos crossbar switch for external input and output.

## 4.4 $MC^2$ Clusters and $MC^2/64$ configurability

The results of the simulation which are presented in this section had extremely far reaching effects as the $MC^2$ Clusters had been redesigned for the new $MC^2/64$.

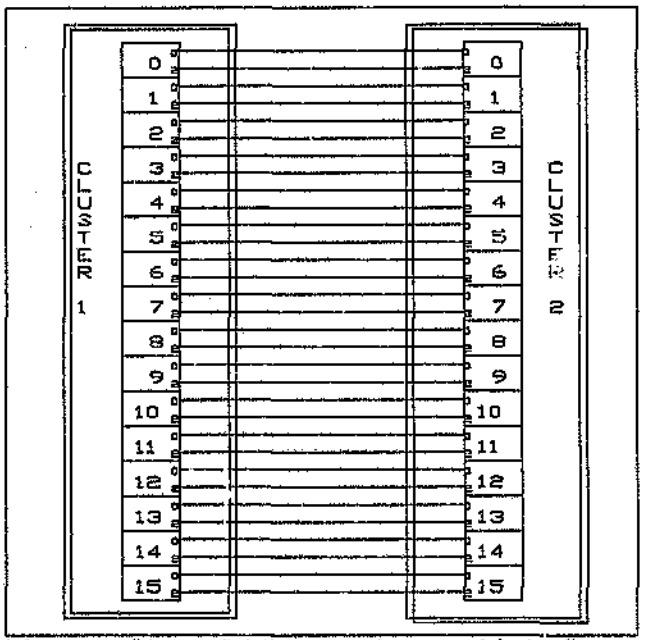
The $MC^2/64$ system consists of four $MC^2$ Clusters connected together via two modified Clos networks. Four of the sixteen links per Euler colour emerging from a cluster are routed to each one of the four crossbar switches. These links are referred to as the inter-cluster links, and two identical networks of inter-cluster links exists, one for each Euler colour.




MC² Cluster

Since the MC²/64 Clusters only release 16 links per cluster crossbar switch (or per Euler-colour) for inter-cluster connections, *the* MC²/64 *system is not completely configurable.* The system can run out of inter-cluster links when trying to connect transputers in different clusters using certain network configurations.

If the 16 transputers in a cluster need 17 inter-cluster links in a specific Euler network, MC²/64 won't be able to switch the configuration as the current MC² Clusters releases only 16 links for inter-cluster connections. As can be seen in the MC² Cluster diagram, 32 links (from the 16 transputers per cluster) enter each one of the two 48X48 crossbar switches, and only 16 links emerge. These 16 links are the key to the configurability of the MC²/64 system. As the number of links emerging from a cluster increases, more links become available for inter-cluster switching of the transputers in a cluster, and the more switchable/configurable the system becomes. The optimum number of links emerging from a cluster is 32 as this enables complete switchability of the MC²/64 system.




It is not possible to switch the connection network in the diagram because of the limited number of inter-cluster links emerging from a cluster. The transputers must be placed in such a way that use of inter-cluster links is limited. The network in the diagram is actually a double pipeline, and it is possible to place the transputers in such a way that only two inter-cluster links are used.

The ratio
*Links emerging from a Cluster per Euler Cycle* to *Transputer Links per Cluster)*
= the ratio
*Links emerging from a Cluster per Euler Cycle* to *(4XTransputers per Cluster)*
gives a direct indication of the configurability of the system. This ratio will be referred to as the configurability ratio of $MC^2$ systems. For the current $MC^2$ systems, this ratio = 1/2. This ratio stays the same for larger systems if the same clusters are used to assemble the larger machines such as the $MC^2/256$. The switchability/configurability of these larger systems is the same as the $MC^2/64$ as it is determined by the $MC^2$ Clusters alone. A configurability ratio of 1 indicates full configurability.

An $MC^2$ Cluster releasing 32 links ensures complete configurability of the $MC^2$ systems. Such a cluster needs two 64X64 crossbar switches instead of the current 48X48 crossbar switches. The disadvantage of such a system is that complexity of the hardware increases immensely, not only because of the larger crossbar switch but because of a doubling in the number of links that needs to be routed through the system. The number of crossbar switches necessary for the Clos switch also doubles (the number of Clos crossbar switches = 32/1 = 32 for a fully connected system).

These results aided the $MC^2$ project team in designing new $MC^2$ Clusters. It was decided that all the future $MC^2$ Clusters would have a configurability ratio of 1, which would enable complete configurability of the $MC^2$ systems. This can be realised by releasing 32 links from a 16 transputer cluster, or 16 links from a 8 transputer cluster [1]. The only reason why the placing algorithm (see chapter 4, section 4.6) would still be used, would be to ensure that most of the links of a user

1.A 8 transputer cluster (or Mini Cluster) also has the advantage that Inmos C004s can be used for the cluster crossbar switches. A C004 can switch 16 links from the 8 transputers, as well as release 16 links for inter-cluster connections.

network fall within one cluster as the link speeds on these links could be faster. The links would not have to be routed through the switching components of th Clos switch either which would result in a delay (see appendix A, section 7.1).

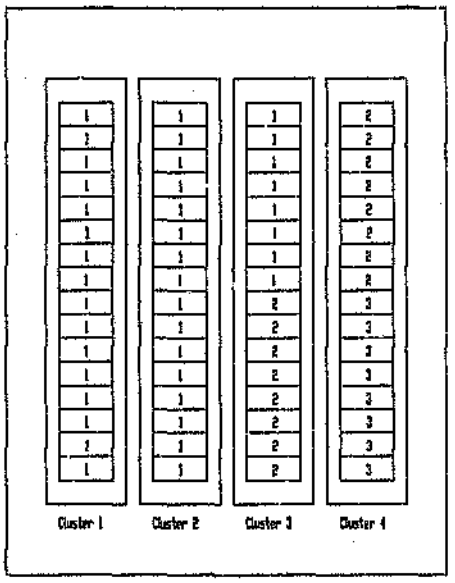
The conclusion that must be drawn is that in theory, the MC$^2$ systems are not completely configurable. However, a user need not know this as there are ways in which the usage of system can be optimised. There can almost always ensure configurability for the user. The rest of this section on MC$^2$/64 configurability will deal with the ways in which the usage of the system can be optimised.

## 4.5 Population of a MC$^2$/64 system

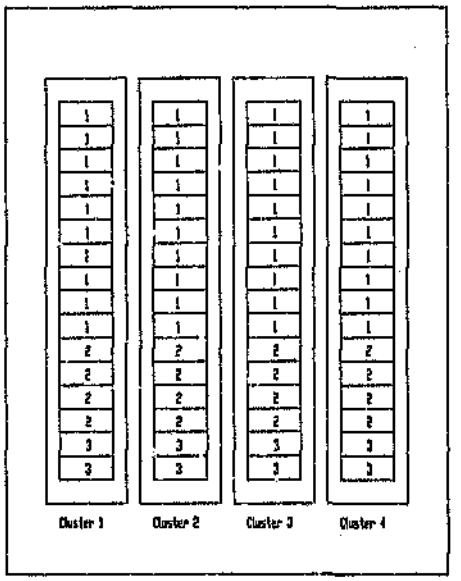
A factor that is important in the user configurability of the MC$^2$/64 system is homogeneousness. *User configuerability* is defined as the configurability a user experiences when trying to switch domains. A *homogeneous* system is defined as a system where all transputers in the system are exactly the same - of the same type and having the same amount of memory. As the users of MC$^2$/64 are allowed to plug different types of transputers into the system, we do not necessarily have a homogeneous system. A MC$^2$/64 system can therefore be homogeneous or non-homogeneous. In a non-homogeneous system the t f transputers and transputer memory vary. The transputer type can be a T800 or a T414, an e amount of memory per transputer can be 1, 2, 4 or 8 MBytes. Special types of transputers, such as a transputer dedicated to graphics and connected to a high resolution screen, can also be used in the system. The population of such a non-homogeneous MC$^2$/64 system is one of the factors that influences the configurability of the system tremendously.

The way in which transputer cards are plugged into the system, is called the population of the MC$^2$/64 system. The population of a homogeneous MC$^2$/64 system is trivial since all the cards are the same. A non-homogeneous MC$^2$/64 system can be populated *vertically*, *horizontally* or *randomly*. *Vertical population*

means that the different types of transputers are plugged into the system starting at cluster 1 transputer 0, filling cluster 1 and then going on to fill cluster 2 until all transputers have been used.




A MC²/64 system filled vertically. The system contains 40 transputers of type 1, 16 transputers of type 2 and 8 transputers of type 3.




The same MC²/64 system filled horizontally

*Horizontal population* means that a system is filled horizontally starting at cluster 1 transputer 0, then cluster 2 transputer 0 up to cluster 4 transputer 0, on to cluster 1 transputer 1, and so forth until all transputers have been used.

An MC²/64 system populated randomly is just what it says : the different types of transputers are just plugged into the system at random.

The influence of the population of the MC²/64 system on user configurability will be discussed further in the section on the MC²/64 configuration simulation.

## 4.6 MC²/64 configuration algorithms

The modified Closs network provides several paths between any two transputers with unconnected links in the current MC²/64 system, but nevertheless the Clos network does not eliminate all problems. As mentioned, there are a limited number of inter-cluster links available due to the fact that a MC² Cluster releases only 16 links for inter-cluster connections (per Euler-colour). The current MC²/64 system has a configurability ratio of 1/2 (see section 4.4).

As there is a limited number of inter-cluster links the configuration of the current MC²/64 is reduced to a placing/locating problem, implying that the use of the restricted resources (in MC², the inter-cluster links) must be limited. This is done by *placing* a user's transputers as far as possible within a MC² Cluster as the MC² Cluster is completely configurable (see chapter 3, section 3.2).

If it is not possible to place the transputers of a user all within one cluster, for example because of the location in the system of specific types of transputers a user is requesting, the transputers must be placed in such a way that the minimum number of inter-cluster links is used. This can be done by placing the transputers as far as possible in one cluster, but again avoiding a nearest-neighbour analysis of the requested user configuration as this approximates the *mapping problem* for which there is no easy solution (see chapter 3, section 3.1).

A possible *placing* algorithm :

```
Find a cluster that can accommodate the user request.
IF a cluster is found, Place the transputers
ELSE UNTIL all transputers placed DO
     Find the cluster that can best accommodate the unplaced transputers
     Place the transputers which can be placed
END
```

The algorithm that was however implemented in version 1 of the DMS system software of MC$^2$/64 is as follow :

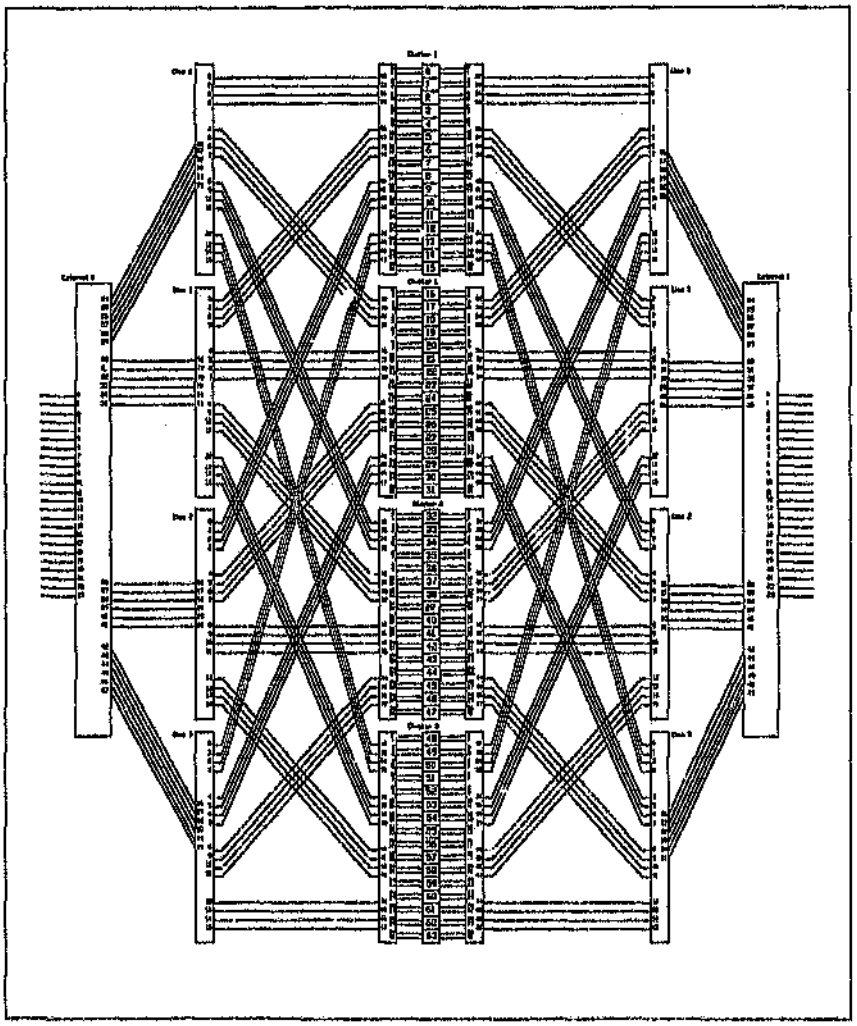
```
Find a cluster that can accommodate the user request.
IF a cluster was found, Place the transputers
ELSE UNTIL all transputers placed DO
     Find the first transputer that matches
     Place the transputer
END
```

This algorithm tries to place the user request in one cluster, and if there is no cluster that can accommodate the user request, a floodfill is done whereby the first transputer in the system that meets the requested transputer type, is allocated. A simulation of this algorithm and the total MC$^2$/64 configurability is discussed in section 4.7.

A simulation of the MC$^2$/64 configurability and the way in which user requests are accommodated by the system, is discussed in the next section.

### 4.6.1 Accommodating user requests

This sections explains how a user request is accommodated in the MC$^2$/64, implementing the *placing* algorithm. The two identical Euler switch cycles can be distinguished in the link diagram of MC$^2$/64.



The link diagram of MC$^2$/64

According to the *placing* algorithm, a user request will be placed as far as possible in one cluster, and the two crossbar switches of the clusters will be switched to represent the requested network in that cluster. To connect transputer 16 link 0 to transputer 30 link 2 for instance, one must connect 0 and

29 in the number 0 crossbar switch of MC$^2$/64 Cluster 1. If one connects the 0 and 29 in the same cluster but using crossbar switch 1, the effect will be to connect transputer 16 link 1 to transputer 30 link 3.

An inter-cluster connection is connected as follows: let us say that the user request requires transputer 16 link 0 to be connected to transputer 44 link 2. This will require a connection in the number 0 crossbar switch of MC$^2$/64 Cluster 1, connecting 0 to for instance 32 and it will require a connection in the number 0 crossbar switch of MC$^2$/64 Cluster 2, connecting 25 to a link connected to the same Clos crossbar switch than Cluster 1 number 32, in this case number 35. In Clos switch 0, crossbar switch number 0 needs to connect 11 and 4 to connect the two inter-cluster links from Clusters 1 and 2. An inter-cluster connection therefore uses two inter-cluster links.

An external connection, connecting transputer 0 link 1 to the user's PC will require a connection in crossbar switch 1 of Cluster 0 connecting 0 to 32. In Clos 1 crossbar switch number 0, a connection from 0 to 16 is needed, and in External switch 1 we need to switch 0 to 24 provided the user's PC is connected to link 0 of External switch 1. An external connection therefore requires one inter-cluster link.

## 4.7 Simulation of MC²/64 configurability

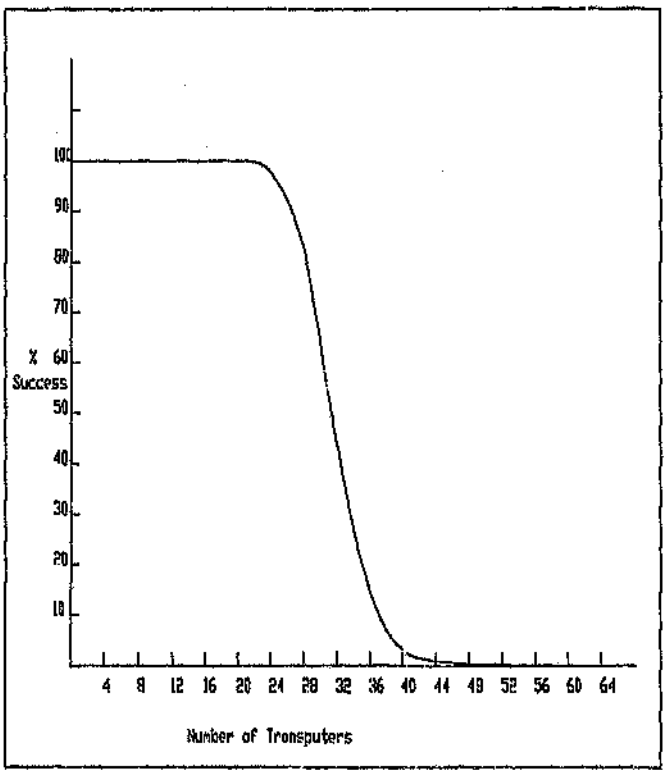
The simulation program, **MAPSIM**, was written to simulate the current MC²/64 system with the placing algorithm mentioned in section 4.6 and the way in which users would use it. **MAPSIM** was written to specifically simulate the configurability and user configurability of the MC²/64 system and not the larger MC² systems. The results obtained during the simulation were used to make predictions about the configurability of the larger systems.

As mentioned, *configurability* is defined as the possibility to always connect an transputer with unconnected links to any other transputer in the system with unconnected links. The MC²/64 is not completely configurable as the MC² Clusters release only a limited number of links that can be user for inter-cluster connections.

*User configurability* is defined as the configurability a user experience when using MC²/64. The user configurability of the system is enhanced by using the *placing* algorithm, and by keeping the system as homogeneous as possible.

Different aspects determining the configurability of MC²/64 were examined. These included the effect of a MC² Cluster releasing only 16 links, the homogeneousness of the system, and user requests. These topics will be discussed in detail.

The configurability of the MC$^2$/64 system as obtained from MAPSIM is depicted in the following graph.




This graph gives an indication of the configurability of the MC$^2$/64 system. The results were obtained by trying to switch 500 random user configurations of each number-of-transputers and the mean number of successes were calculated and plotted. (Important to note here is that the links of the transputers were connected *randomly*.)

The results obtained in the MAPSIM simulation are not very promising, but this does demonstrate the *worst* possible case. The graph shows that, when using random link connections, the success rate of switching 32 transputers is about 44% and the success rate for switching a configuration of 64 transputers is zero.

It is, however, possible to have a much higher success rate if the user adheres to some switching and system configuration rules, of which some are instinctive rules such as not numbering or connecting a network of transputers randomly.

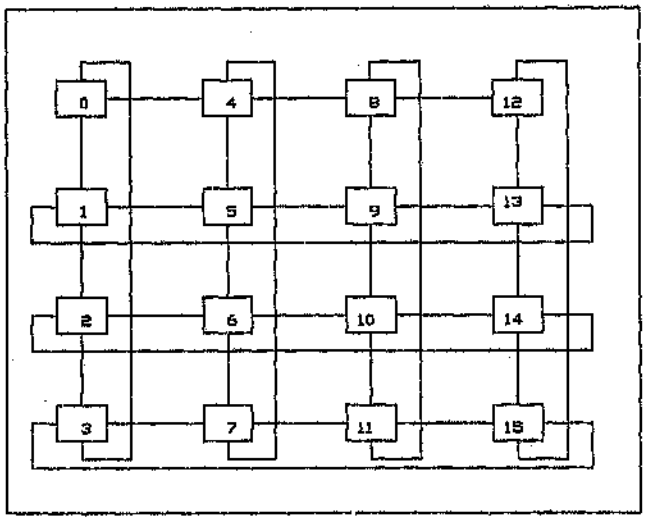
### 4.7.1 Population of MC²/64

As mentioned, a *non-homogeneous* MC²/64 system can be populated *vertically*, *horizontally* or *randomly*.

The following example shows how a user configuration consisting of transputers of different types is influenced by the population of the system. (This example was obtained from MAPSIM.)

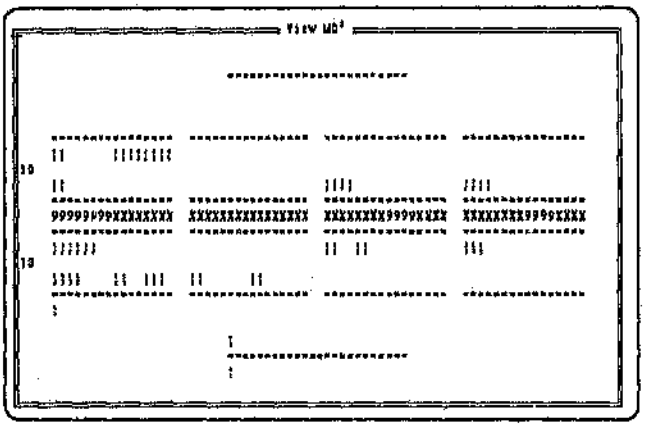
The MC²/64 system contains 40 1Mbyte T800 transputers, 16 2Mbyte transputers and 8 4Mbyte transputers and is populated *vertically*.

```
! The configuration information : (T8 = T800)
Number = 16
Domain = 9
(T8,1024,0; (( 3,0);     E1; ( 4,2); ( 1,3))
T8,1024,1;  (( 2,0); (13,1); ( 5,2); ( 0,3))
T8,1024,2;  (( 1,0); ( 3,1); (14,2); ( 6,3))
T8,1024,3;  (( 0,0); ( 2,1); ( 7,2); (15,3))
T8,1024,4;  (( 5,0); ( 7,3); ( 0,2); ( 8,3))
T8,1024,5;  (( 4,0); ( 6,1); ( 1,2); ( 9,3))
T8,1024,6;  (( 7,0); ( 5,1); (10,2); ( 2,3))
T8,1024,7;  (( 6,0); (11,1); ( 3,2); ( 4,1))
T8,2048,8;  ((11,0); ( 9,1); (12,2); ( 4,3))
T8,2048,9;  ((10,0); ( 8,1); (13,2); ( 5,3))
T8,2048,10; (( 9,0); (14,1); ( 6,2); (11,3))
T8,2048,11; (( 8,0); ( 7,1); (15,2); (10,3))
T8,4096,12; ((13,0); (15,1); ( 8,2);      X)
T8,4096,13; ((12,0); ( 1,1); ( 9,2); (14,3))
T8,4096,14; ((15,0); (10,1); ( 2,2); (13,3))
T8,4096,15; ((14,0); (12,1); (11,2); ( 3,3))
```

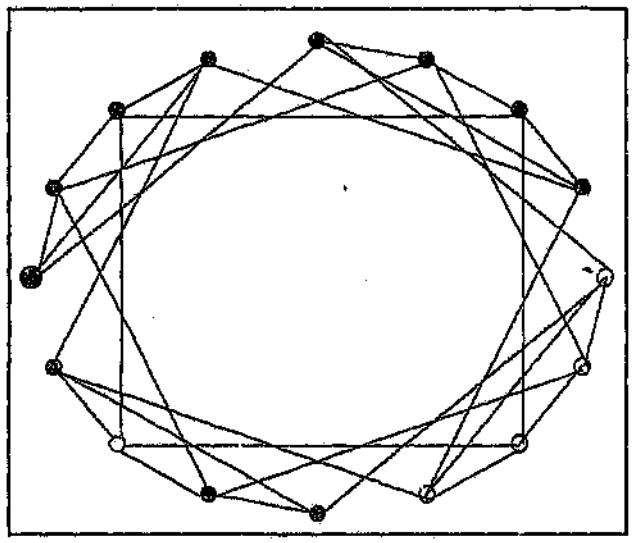



The user request: A mesh of sixteen transputers
of which 8 transputers have 1Mbyte,
4 transputers have 2Mbyte and 4 4M[illegible]e

The user request is accommodated by the system as shown below:

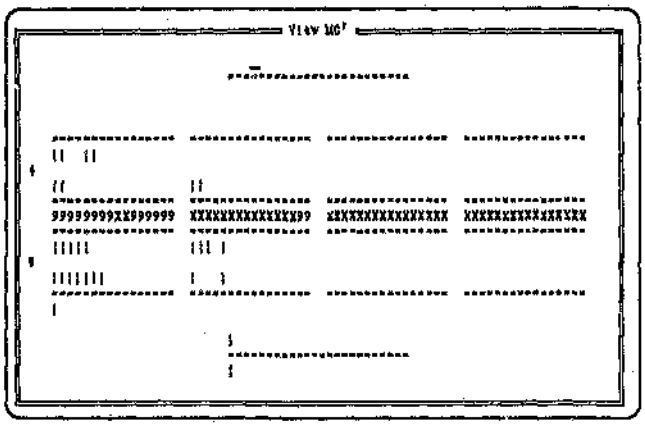


The transputers (indicated by the domain number 9) of the user are distributed over three clusters and a lot of inter-cluster links (the numbers on the left) are necessary to switch the user configuration.

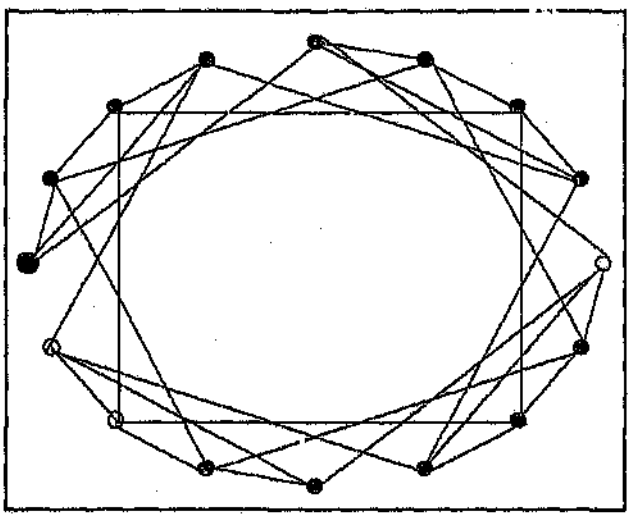


The inter-cluster links necessary to realise the user request. Going clockwise around the graph starting at the fat transputers, the first group of filled transputers are in the same cluster, and the open transputers in another cluster, and the last group of filled transputers in yet another cluster. Lines connecting transputers that are shown to be in different clusters are inter-cluster connections, consisting of two inter-cluster links.

In a horizontally populated system, the user request is accommodated as shown below, resulting in a configuration that needs much fewer inter-cluster links:



Fewer inter-cluster links are used when the system is configured horizontally



The transputers are located in two clusters only. For this user request, only 6 inter-cluster connections and therefore 12 inter-cluster links are needed. Another inter-cluster link is used for IO.

The above results obtained from MAPSIM show that a non-homogeneous system influences the configurability of MC$^2$/64, since a user request will most probably be distributed through the clusters. If most user requests are homogeneous, a vertically populated system will provide the best results concerning the usage of inter-cluster links. Non-homogeneous user requests will be accommodated better in a vertically populated MC$^2$/64 system as can be seen from the above example.

The ideal for the current MC$^2$/64 is still to use it as an homogeneous system, as the MC$^2$/64 was originally designed to be used as an homogeneous system. Although MC$^2$/64 can be used as a non-homogeneous system, this usage must be minimised as far as possible.

### 4.7.2 User requests

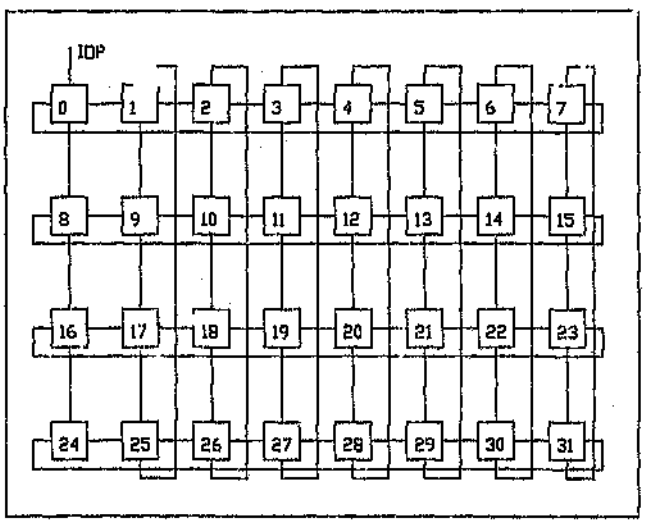
The way in which the system is used is also a factor determining the user configurability of the current MC$^2$/64 system. This includes the number of transputers the user requests, the network configuration requested, and how the network is requested from the system.

According to data collected from users, the number of transputers a user allocates will generally be 1 (for program development), or an fixed number of processors (to run a specific program). This number of processors will most probably fall in the set {1, 2, 3, 4, 5, 8, 10, 16, 32, 64}, while other requests are possible but not very likely. This is because the common network configurations all use a certain number of transputers. As most of the users will generally need a number of transputers less than 16, a user request could be accommodated in one cluster and the system configurability would be high. Whether a user request could be accommodated in one cluster is again dependent on the homogeneousness and the population of the MC$^2$/64 system.

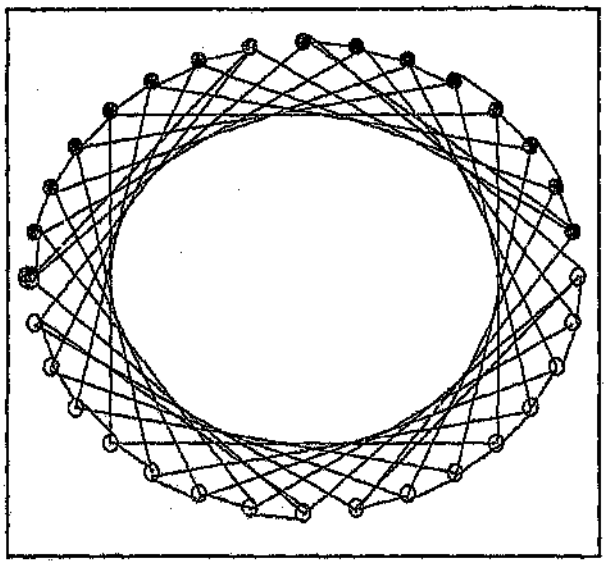
The network configuration requested by a user also plays an important part in the configurability of MC$^2$/64 as the system will not perform nearest-neighbour placing or searching. Such a method will again approach the "*mapping problem*". The system places transputers in the sequence they appear in the user request. The first transputer will be placed first, the second will be placed second etc. The responsibility to request and number transputers in such a way that the system will place them optimally, rests on the user. If the user request can be accommodated in one cluster according to the placing algorithm, the user network will not influence the configurability of the system.

### 4.7.2.1 Numbering

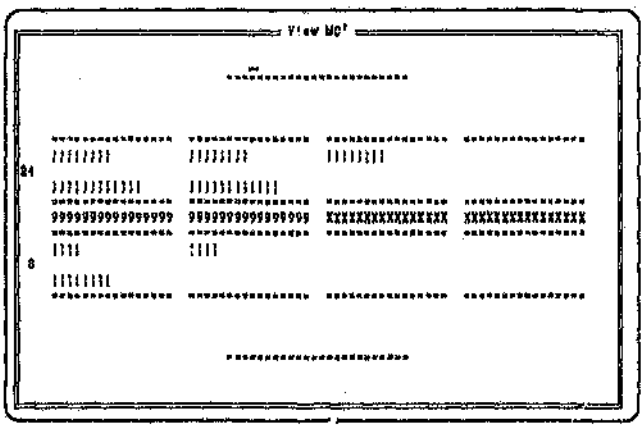
The way in which the numbering of a user request influences the configurability, is presented in the next example obtained from MAPSIM.



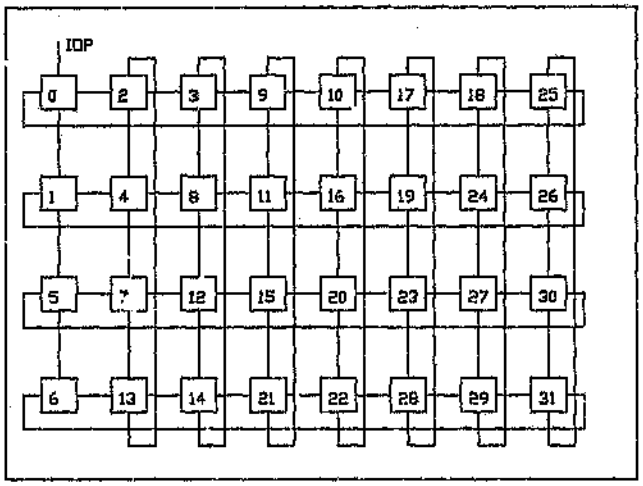

An example of numbering for a mesh.
The transputers must be numbered in such a way that
the nearest neighbours have the nearest numbers.



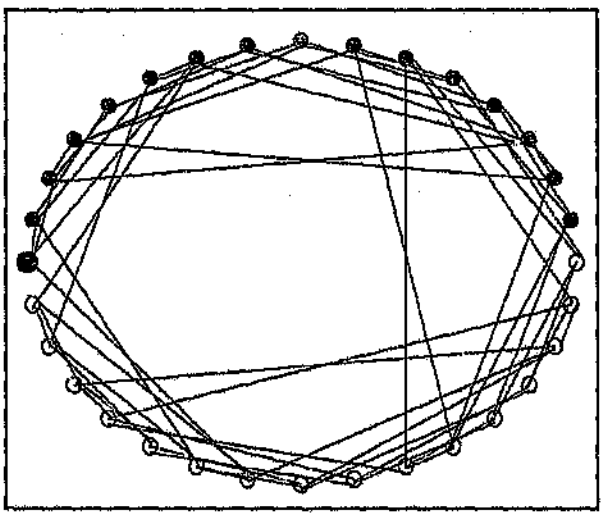
A diagram of the switching network. Sixteen inter-cluster links are used to realise the user-configuration.



The allocation of the transputers of the mesh. In a homogeneous system the transputers will be allocated in the first two clusters.

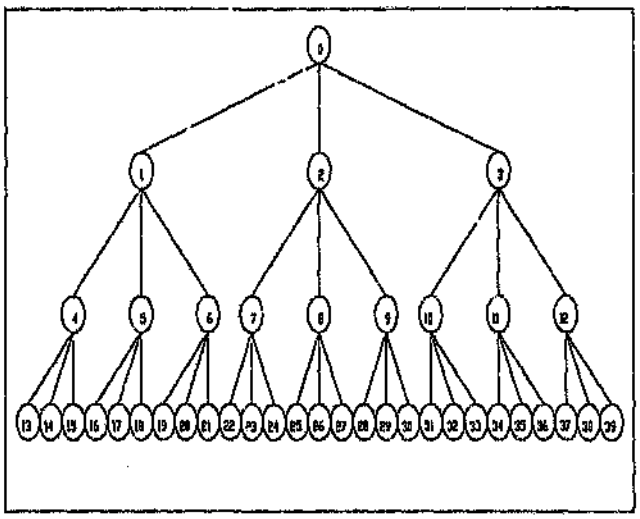



The same mesh with a different numbering sequence

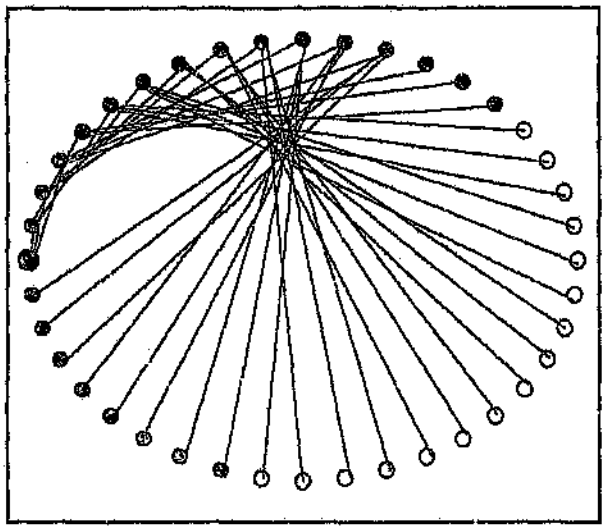


The diagram of the switching network. This diagram indicates that only 12 inter-cluster links are used. The way in which this mesh was numbered is therefore more effective than the numbering of the previous network request.

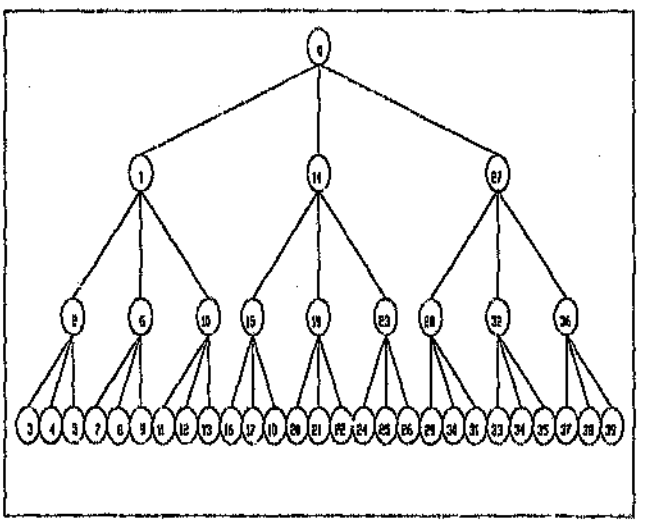
What follows is another example of the influence that numbering has on the inter-cluster links used in the system. In the first figure, a tree was numbered horizontally, and in the second vertically, resulting in an substantial reduction in the number of inter-cluster links used.



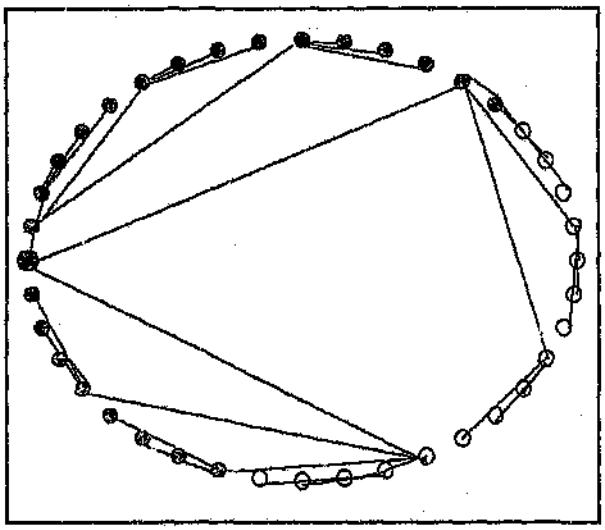
A tree numbered horizontally



The inter-cluster links used for the horizontal numbering



A tree numbered vertically, with a better nearest neighbour placement

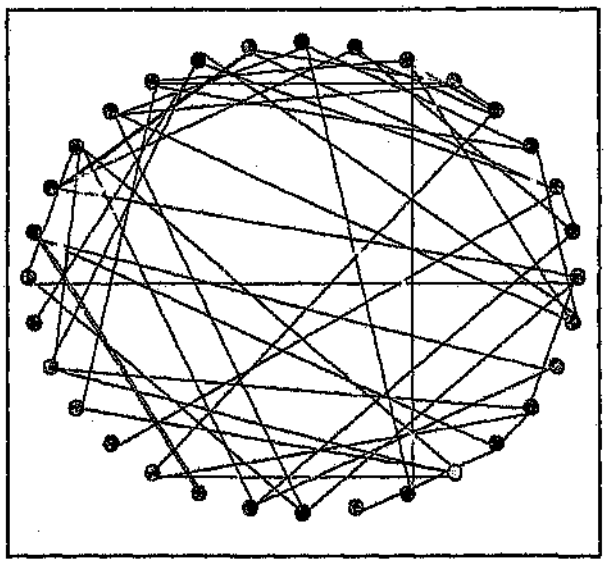


The inter-cluster links used for the vertical numbering is much less than in the previous example.

### 4.7.2.2 Neighbouring

The concept of neighbouring was introduced in the simulation program as a method with which to simulate the way in which users would request their configurations ("neighbouring" meaning that a certain transputer may only be connected to his numbered neighbours). A neighbouring of 4 will therefore result in a transputer only being connected to its four nearest neighbours on both sides concerning their *numbers*. A neighbouring of 4 will result in transputer 10 only being connected to transputer numbers 6, 7, 8, 9, 11, 12, 13 or 14.

If a user connects his transputers randomly (or numbers his transputers randomly), a typical graph displaying the user connections would look like this:

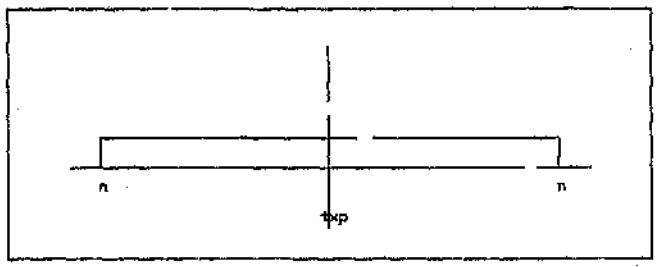


A random connection of 32 transputers

The graph shows that random user configurations use a large number of inter-cluster links. Fortunately, most users of MC$^2$/64 do not use random configurations, and we could assume that a user would not just number or connect transputers randomly, but would start, for instance, at the top left corner with numbering and number all his transputers in an ordered way. We can therefore assume that any user will use some form of neighbouring, especially if it is important for the configuration of the machine. This leads to the concept of neighbouring.

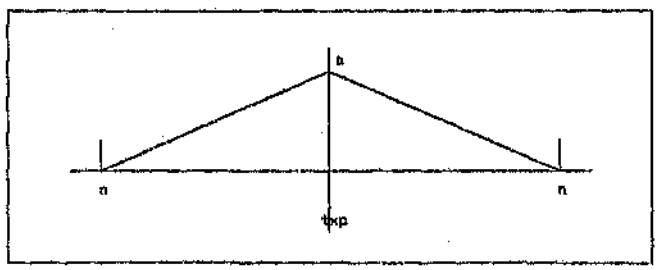
Neighbouring of transputer connections (n=neighbours) means that a specific transputer may only be connected to its neighbours on both sides. If the number of the transputer is 8 and the neighbouring is set to 4, transputer 8 may be connected to transputers in the set {4,5,6,7,9,10,11,12} (if the number

of transputers in the domain is more than or equal to 12). A random transputer number within this set is selected to which the transputer is then connected.

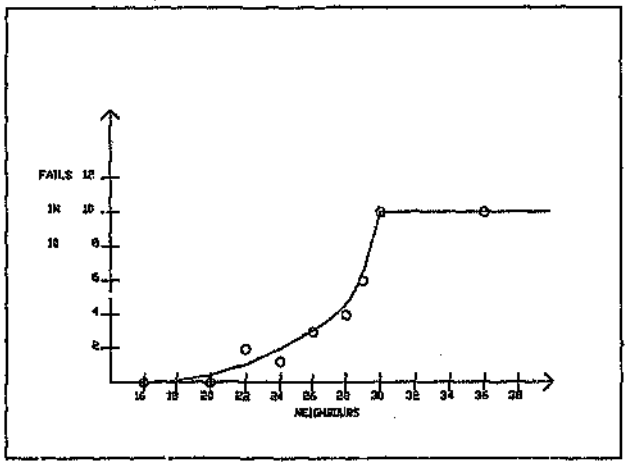



Neighbouring

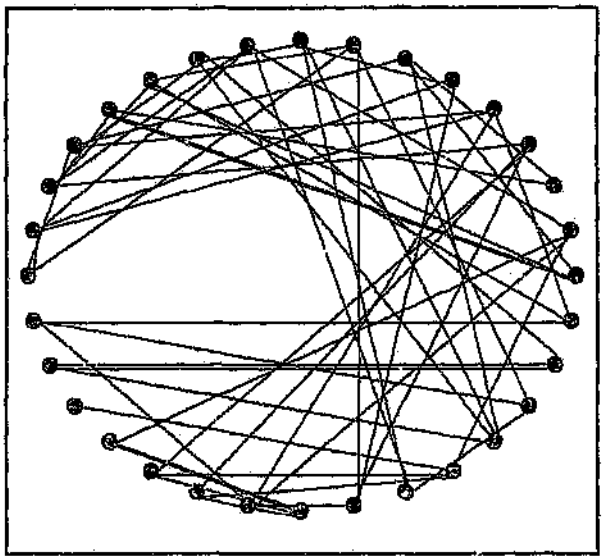
*Spesneighbouring* or *special neighbouring* of transputer connections means that the transputers in the neighbouring set are weighted, and then selected. If the current transputer is again 8 and the spesneighbouring is set to 4, the possibility of transputers 7 and 9 being selected will be most likely, followed by transputers 6 and 10, and lastly transputers 4 and 12. The gradient depicts the amount at which certain transputer numbers are weighted, and this gradient can be changed.




Spesneighbouring (n=neighbours, a=amplitude)

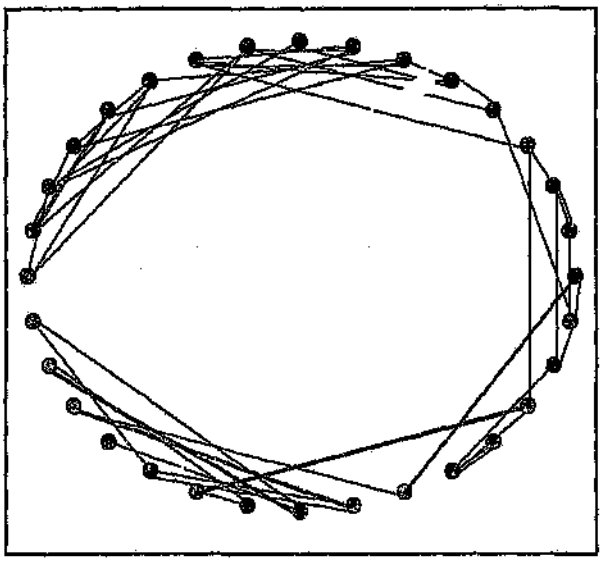



This graph gives an indication of the effect of neighbouring on the configurability of the MC2/64 system. A neighbouring of 30 has 10 fails out of 10 tries, but a neighbouring of 20 still has a 100% success rate. This was tested in a homogeneous system, but with random user connections.



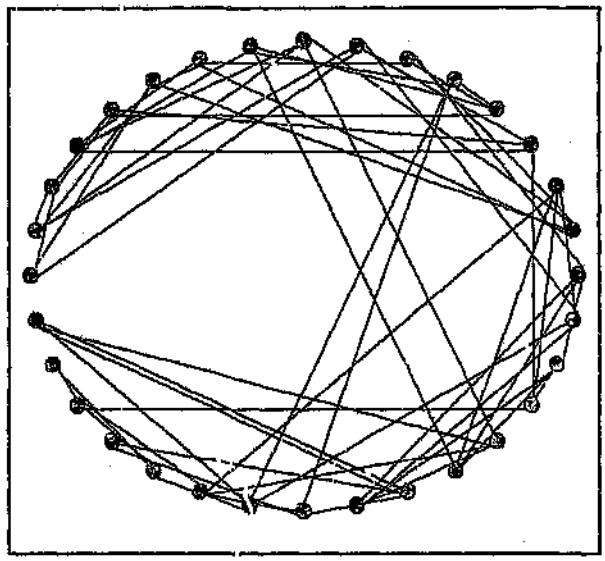
A neighboured configuration of 32 transputers with a neighbouring of 16.

The graph indicated a normal neighbouring (without weighting) of 16, in other words, a current transputer can only be connected to its 16 nearest neighbouring transputers (in number).



A neighboured configuration of 32 transputers with a neighbouring of 8

This diagram differs from the previous one as the neighbouring has been reduced. A current transputer can now only be connected to its 8 nearest neighbours (in number, not in location). The numbering must again be performed in such a way that the nearest numbers to a current transputer are also its nearest neighbours.



A spesneighboured configuration of 32 transputers with a neighbouring of 16 and a gradient of 5

This diagram differs from the first diagram as it illustrates the weighted neighbouring. The gradient of the weighting is 5. There are fewer long lines connecting transputers in this diagram than is the case in the first diagram.

The concept of neighbouring provides us with a more realistic view of the usability of the $MC^2/64$ system, although the model is not without its flaws. If a user adheres to some form of neighbouring, the possibility that he will be able to switch his network configuration is quite high, even if he uses random link configurations. The above exercise has enabled us to provide the user with directions for the setup of a user configuration, where the transputers must be connected where possible to their nearest numbers.

## 4.8 Conclusion

This section discussed the interconnection network needed when connecting the $MC^2$ Clusters to form the integrated $MC^2/64$ systems. Two modified Clos networks are used in the $MC^2/64$ systems, one for each Euler half. This network provides several paths between any two $MC^2$ Clusters in the system, and is non-blocking.

This section also showed that the current $MC^2/64$ systems are not completely configurable, in fact, the configurability of the $MC^2/64$ systems = 1/2 because of the limited number of inter-cluster links released by the current $MC^2$ Clusters for inter-cluster connections. Configurability is not a factor determined by the interconnection network, but by the $MC^2$ Clusters.

The *placing algorithm* was implemented in version 1 of the DMS (Domain Management Server) software. This algorithm tries to minimise the usage of the limited inter-cluster links when accommodating a user request in the system.

The MAPSIM simulation program was also discussed in this section. This program provided valuable insights into the ways in which $MC^2/64$ can be used to optimise the user configurability[1]. The main factors influencing *user configurability* are :

* the population of the $MC^2/64$ system, and
* user requests.

The results obtained from the two $MC^2/64$ systems currently operational showed that users very seldom ran into problems with configurability. It seems that most users use the system in such a way that they do not run into configuration problems.

1. *User configurability* is the configurability a user sees when trying to switch his user requests. The *configurability* of the system influences *user configurability*. The system *configurability* can only be changed by changing cluster designs. *User configurability* can however be enhanced by implementing algorithms (such as the *placing algorithm*) which optimise the use of the restricted resources. A user will then have a more configurable system.

# 5 MC²/64 SOFTWARE

## 5.1 Introduction

This section will introduce the MC²/64 system software or the DMS[1], and the way in which this software was designed and developed. This is the software incorporating the configuration algorithms of chapters 3 and 4.

## 5.2 Software development life cycle (SDLC)

Software is usually developed according to a life cycle model. Different life cycle models exist and the emphasis of the different software development life cycle models usually vary according to their applications. Boehm, in his article 'Software Engineering' presented one of the most known software life cycles (Boehm, 1976). Boehm's life cycle starts with the *system requirements* phase. During this phase a complete, consistent and unambiguous specification must be specified, describing *what* the software must do, but not *how* it must be done. The *how* must be specified during the design phases.

Boehm identifies a few problems during the *design* phase, one being that the software specification usually cannot be finished without undertaking some design and even implementation of the system to verify certain aspects of the specification. Software designers also have a lot of freedom in their designs as no specific design standards exist which can help designers to choose between different alternatives during this phase. Software designers also tend to design from the bottom up. They develop software components before addressing the serious issues of interfaces and integration (Boehm, 1976).

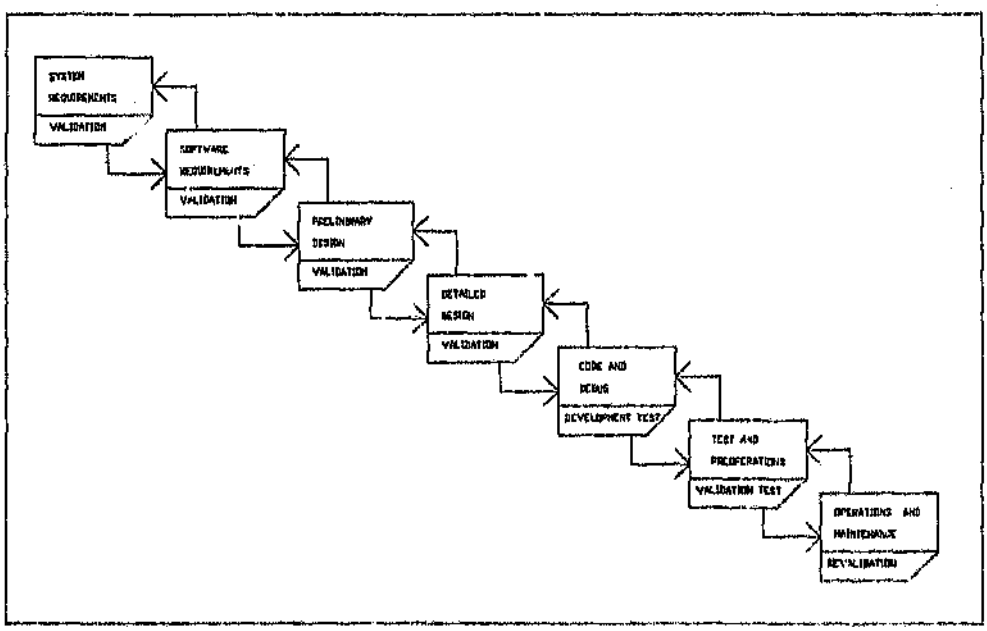
1.Domain Management Server

During the *coding* phase, the software is coded in a specific language. Here programmers also have a lot of freedom. Boehm mentions that programmers tend to limit control structures and are moving towards structured code (hierarchical, block-oriented code.)

Boehm argues that the Software *Testing* phase also presents serious problems during the life cycle of most software systems as software testing and reliability are not considered until the code has been run for the first time, and been found not to run.

*Maintenance* is also a very neglected phase of the software life cycle, and is also the most expensive (Boehm, 1976). If software specifications and software design are done correctly, it should be much easier to maintain the software.

Boehm's software life cycle is depicted in the following diagram (Boehm, 1976):




Software development life cycle

To give a detailed description of all the different SDLC models, is beyond the scope of this dissertation. For the purpose of this dissertation, the author will adopt the same model as Du Plessis in the dissertation "A Software Engineering Environment For Real-Time Systems" (Du Plessis, A.L., 1986). This SDLC has been developed within military circles as an answer to the need for a coordinated effort for a single standard for developing defense system software. A standard, the DoD-STD-2167, developed out of these efforts, and this standard specifies a computer software development cycle which should be followed within the system life cycle.

The standard has a comprehensive and well-structured format. The system life cycle within which the SDLC is structured consists of four phases: concept exploration, demonstration and validation, full scale development, and production and deployment. The SDLC only commences after the second phase of this sytem life cycle, and it extends into the third phase up to the point of system integration and testing (DoD-STD-2167, 1983).

The SDLC incorporates the following phases:

* **Software requirements analysis** - a complete set of functional, performance and interface requirements derived from the system specification.

* **Preliminary design** - a modular top-down design developed from the software requirements.

* **Detailed design** - a modular, detailed design of the system.

* **Coding and unit testing** - coding and testing of each module or unit.

* **Software integration and testing** - the released units are integrated and the integrated modules are tested.

* **Software performance testing** - formal tests are conducted and the results are recorded.

Du Plessis has defined the following definitions which are relevant to this chapter and which are presented here (Du Plessis, 1986):

* **CASE** (Computer-Aided Software Engineering) - the engineering of software supported by computer tools.

* **Method** - a regular, orderly definite way of doing anything, applied to the specific approach to carrying out one or more of the software development and post-deployment support activities; is frequently based on an abstract or intellectual model of how to accomplish an activity; is implemented by utilizing procedures and tools.

* **Methodology** - a system of methods which provides the overall approach to developing and improving software.

* **Procedure** - a manner of carrying out a certain process; refers to the manual activities and steps required to implement a method.

* **SDLC** (Software Development Life Cycle) - the cycle through which the development of the software of a system is undertaken; it is divided into a number of phases each involving a number of activities, and resulting in particular phase products; the SDLC forms part of the system life cycle of the larger system of which the software component forms a part.

* **Technique** - an informal method.

* **Tool** - a mechanism for rendering a method executable; computer tools are the computer programs which enable the execution or implementation of the steps of a method.

## 5.3 MC²/64 system software

The design and development of MC²/64 System Software is incorporated into the adopted SDLC of section 5.2.

### 5.3.1 Software requirements

The main objective of the MC²/64 system is to provide a system user with a distinct user environment consisting of transputers and other resources within MC²/64. A user must be able to manipulate these transputers on command. This includes switching or reswitching of the transputers, linking of transputers to external resources, the running of user software and obtaining information from the transputers. The group of user transputers and external resources will be called a user domain.

The different actions that the system has to fulfil are as follows:

- **Allocate domains**
  Allocation of a logical user domain[1] and the connection of the user to the logical user domain. This function also checks the user's capabilities[2] against the user request.

- **Switch domains**
  Switching and reswitching of the physical user domain to match the requested user configuration.

1.The *logical user domain* is conceptually the domain as the user requests it with the numbering the user gives his transputers and resources. The *physical user domain* will then consist of the physical transputer numbers and where these transputers are located in the system. The physical user domain is transparent to the user, a user only sees the logical user domain.

2.Each registered user of the system will have certain capabilities. Capabilities will, for instance, specify the total number of transputers a user is allowed to allocate, and the types of transputer he will have access to.

- **Control of domains**

  Standard Inmos control supplied within a specific user domain. This includes *reset transputer* and *analyse transputer*.

- **Feedback from domains**

  Feedback from any transputer within a specific user domain. This includes *get transputer performance* and *get transputer error status*.

These functions must provide a user with control over his domain of transputers, although the control must be transparent to the user. A user must be able to access and use his domain as if it is a single-user transputer workstation connected to his IOP (or IBM PC). Users must not influence each other's domains in the MC$^2$/64 system. This is an automatic advantage of a massively parallel machine where resources are not shared in any way by the different users. The performance of a user domain is therefore not influenced by the number of users logged into the system at a specific time.

The front end user-interface giving a user access to his domain, must be user-friendly. Users tend to have an inherent resistance to using such a big and complex system as the MC$^2$/64, and the user-interface must guide and aid them in using the system.

#### 5.3.1.1 Evaluation of formal methodologies

The formal specification of a system can be done during the Software Requirements Analysis phase of the SDLC. The MC$^2$ team investigated possible formal methodologies for the specification of complex parallel software systems. It was decided to evaluate Z and CSP as possibilities to the formal specification of MC$^2$ software.

The theory of formal specification methods has its roots in abstract algebra. Every engineering discipline has a strong mathematical underpinning on which it relies for its analytical, predictive and synthetic power. This has not

always been the case with computing or computer programming, although a lot has been done in this direction recently. The demand for computer-based tools which would help in the design and analysis of the software developing process is growing. There is also a considerable shift towards techniques with formal mathematical foundations such as Z and CSP.

Formal specifications use mathematical notation to describe in a precise way the properties that a system must behave without describing the way in which these properties are achieved. The emphasis is on the *what*, instead of the *how*.

The disadvantage of formal methods is that no tools exist at this stage, and that the learning curve is steep. This represents a problem when deadlines must be met. And this is what happened with the MC$^2$/64 system software. Although the initial intention was to user a formal method to describe the system, the deadlines that had to be met prohibited this. The learning curve was too steep to be able to describe such a complex program as the MC$^2$/64 system control software in the available time.

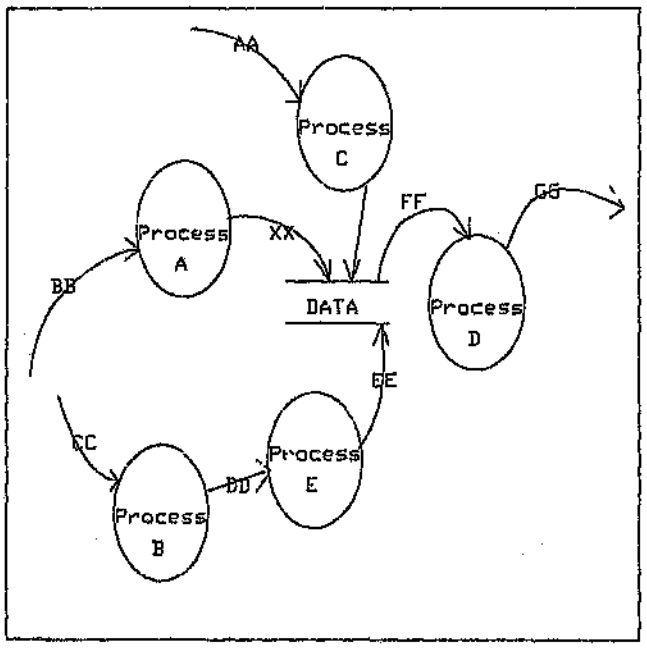
#### 5.3.1.1.1 Z

Z is based on set theory. Z decomposes the system specification into small pieces or schemas. Schemas are used to describe both the static and dynamic properties of the system. The static properties are the states that it can occupy as well as the important relationships that are maintained as the system moves between states.

The dynamic aspects include the operations that are possible in that state, the relationships between the inputs and outputs of the operations and the changes of state which take place.

Z specifications concentrate around a defined state space on which certain operations are described. It is therefore appropriate to specify systems that centre around a specific state space in Z. The examples of system specifications in Z are all describing systems which evolve around a central state such as the formal specification of a microprocessor instruction set or the formal documentation of a block storage device.

An DFD[1] of such a system might look something like this:




A DFD of a system that could be specified in Z. (The AA, BB, CC etc. in the diagram indicates data flows, while A, B and C for instance, specify processes. DATA is a data store.)

1.Data Flow Diagram

Z was evaluated as a possible formal design method to describe $MC^2$ systems. When trying to compose a data flow of the systems specified in Z, an interesting result was obtained. The data flows centred around a data store. In other words, all the processes influence some central data store in one way or another. This seems to be a valid conclusion since Z specifies a state space with the operations valid on the space.

It seems therefore that Z lends itself the best to the specification of such a system controller as the one used in the $MC^2/64$ system. In this specific case, the central state space is the data structure underlying the system controller that specifies all the resources of the $MC^2/64$ system, and how they are allocated and controlled.

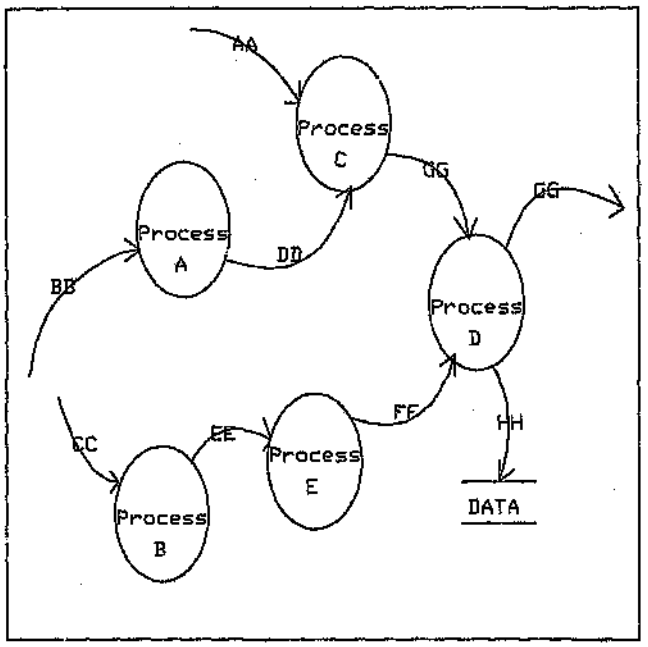
#### 5.3.1.1.2 CSP

CSP is a theory of communicating sequential processes which attempts to provide a notation for designing, specifying, implementing and reasoning about systems of concurrent processes. CSP offers a mathematical notation for the description of processes and a way (alphabets) of controlling the level of abstraction.

CSP provides a secure mathematical foundation for the avoidance of errors such as divergence, deadlock and non-termination, and for the achievement of provable correctness in the design and implementation of computer systems.

Occam, the programming language of the transputer, is a subset of CSP, and CSP therefore lends itself to the specification of concurrent systems and the way in which concurrent processes influence each other.

The DFD of such a system in contrast with the Z diagram might appear as follows:




The DFD of a system that could be specified in CSP. (AA, BB, CC, etc. indicate data flows. A, B, C, etc. are processes and DATA is a data store.)

The data flow diagrams of systems specified in CSP does not centre around a data store as in the systems specified in Z. A system with any concurrent processes, or processes that interact with each other, will lend itself ideally towards specification in CSP.

CSP lends itself best to the specification of any system in which one wants to study the concurrency of the system, and the way in which processes influence each other. This is not the case with the current **MC$^2$/64** system software. The **MC$^2$/64** system control software provides a single entry point for all the users of the system. To prevent user requests influencing each other drastically, we must ensure that a single user request is handled at a time, and the core of the **MC$^2$/64** system software is therefore sequential. There is therefore no need to argue about deadlock or concurrency and CSP need not be used to describe the system. Future

MC²/64 system software might use concurrency, and in these instances it will be useful to use CSP to formally specify the interaction of the different modules.

#### 5.3.1.1.3 Mixing Formal Methods

A question that arose during the evaluation of Z and CSP, was if they could be used toget in the specification of a system.

Z rests on a different mathematical basis than CSP. One of the main reasons for using formal specification methods, is that the user can reason about their correctness. One can therefore not use Z and CSP together without knowing exactly how their mathematical systems interact with each other. Doing this without knowing the consequences, is to lose sight of reason why formal specification methods were decided upon in the first place.

The way in which the different mathematical systems complement each other and interact with each other, is at this stage a field for considerable research, and is beyond the scope of this dissertation.

The conclusion reached therefore, is that Z and CSP can both be used to specify a system, and they will complement each other on different aspects of the design. They must, however, be used in parallel with each other.

Here it is also valuable to mention the work done by Tereul on mixing formal specification styles (Tereul, 1987). Tereul used Z and CSP to describe two problems. In the specification of a heating system, he used CSP. In the specification of a Library, he used Z to specify basic data types and the status changes influencing them, while specifying the interaction between library users and staff in CSP.

## Preliminary Design

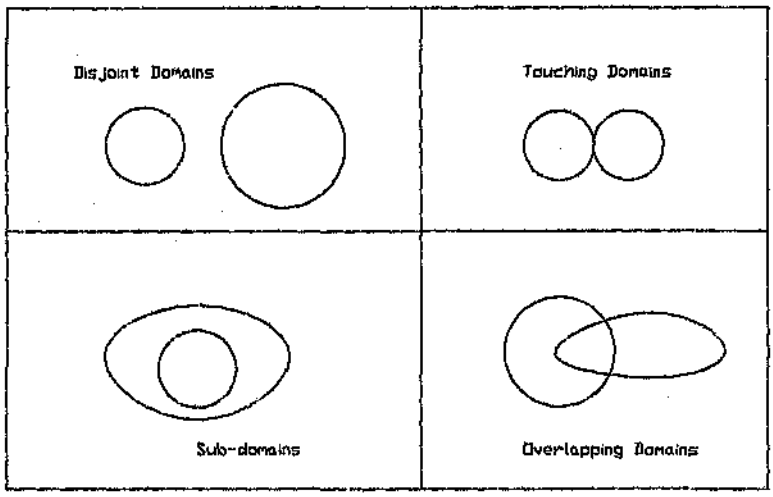
During this SDLC phase an appropriate abstract to the MC$^2$/64 system software was selected which enabled the different programmers to describe the MC$^2$/64 system and the system concepts.

### 5.3.2.1 Domains

The MC$^2$ is a distributed machine and the control of the resources is a difficult task. Because of this, the domain type of object-oriented control has been selected as the best abstract to the problem. Domains are used specifically for the long-term management of large distributed systems. Domains as an abstraction to distributed computing systems was introduced by Robinson and Sloman (Robinson et al, 1988).

Domains is an abstraction which has come to be used in modern operating systems to describe user rights in different areas of a system. In the MC$^2$/64 the implementation will differ from most other implementations (such as in Local or Wide Area Networks) but the general concepts are universal.

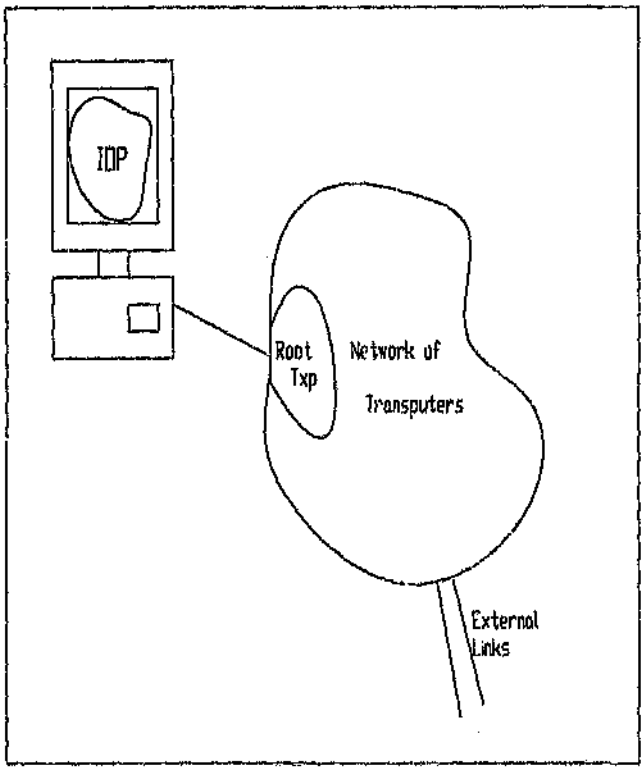
"A domain is a set of resources which share some common attribute. In particular, it is the set of resources to which the same management policy applies... The manager can be a distributed set of software processes which cooperate to manage a set of resources. Domains specify the sphere of influence of a manager." (Robinson et al., 1988). A user can therefore be allocated a domain in MC$^2$/64 consisting of transputers and external resources for which he has the resources. For this domain, the user aided by the *Domain Manager* software, is then the manager of his domain.




Domains

There are different types of domains such as disjoint, sub-, touching and overlapping. In transputer-based machines the lack of support for memory protection implies a single user per transputer which means that only disjoint domains are allowed, and no overlapped domains. Sub-domains however, can be supported.

A user domain can be divided into sub-domains which may be touching on the same level within the same parent (see links 1 and 2 in the above figure). A sub-domain can be defined on its parent's boundary and therefore provides a more flexible touching environment (see link 2 in the above figure).




A user domain

In the MC$^2$/64, a user domain was specified as consisting of all the transputers and resources allocated to a user from the parent domain (PD). All the transputers and resources in the system would be in the parent domain. User domains (UDs) are actually sub-domains of the parent domain. User domains in the MC$^2$/64 can never be overlapping. The IOP is the interface communicating with the user and the domain via a link from the root transputer. External links connect external resources to the domain.

A user domain will be created out of the parent domain (PD) for a user asking for access to the system. When he has finished his session, the resources will be freed back to the PD and another user can get access to them by allocating a domain.

This approach to the $MC^2$ software provided the team with a suitable concept to describe the functionality of the software.

### 5.3.3 Detailed design

#### 5.3.3.1 Design methodology of MC²/64 system software

One of the most important issues around the design of the MC² system software, was to decide on a design methodology for the detailed design phase of the SDLC. There were different individuals who worked on different aspects of the system software. It was necessary for them to communicate and document their designs and interfaces to each other.

The project decided to use the structured analysis methodology of DeMarco, specifically the data flow diagrams for high level specification and analysis.

##### 5.3.3.1.1 DeMarco

One of the advantages of a data flow analysis such as DeMarco's is that various tools exist which simplify the actual use of the methodology, and also make the initial learning curve much shorter. This an important advantage of using such a methodology. The MC² decided to use the HP Teamwork package for use on the IBM PC.

The project team for the development of the system software comprised 5 people, 4 of whom were engineers and 1 a computer scientist. The experience of the team in the area of software design and development varied considerably.

The software design methodology of DeMarco was used very effectively as a tool for communication purposes. The system was first designed at the high level. Each programmer then had to design his part of the MC²/64 system software using the DeMarco methodology. He then had to make a presentation of this design during a round-table-session where all the other programmers could ask questions and give comments. These discussions

assisted the different programmers to clearly identify not only the interfaces between the different modules, but also problem areas. It was easy to design with the future expandability of the system in mind.

One of the most important conclusions that was drawn from this exercise was that a specific design methodology does not supply all the answers and definitely does not automatically ensure unambiguous communications in the design of complex software. The designs need to be discussed in detail by the different programmers on the project team. However, software design methods help any programmer to clearly define and communicate his line of thought during the design of his software.

The best solution to the shortcomings of different methodologies would be to use different design methodologies to design different aspects of the system. Each design methodology focuses on a specific aspect of a system and this must be used to get a clearer picture of the system. Different systems will also need different approaches to their design. The disadvantage of this will be the learning curve involved for the designer(s). The designers or programmers must obtain enough knowledge on each methodology to be able to use them jointly.

As the DMS[1] has a sequential control flow with only one entry point, the data flow analysis in terms of the domain approach was sufficient to describe and design the system. The DFD design approach would definitely not be sufficient for a system with complex control flow, and in such a case an additional methodology would be needed for control flow analysis.

1.Domain Management System

## 5.3.3.2 System design

### 5.3.3.2.1 Contents

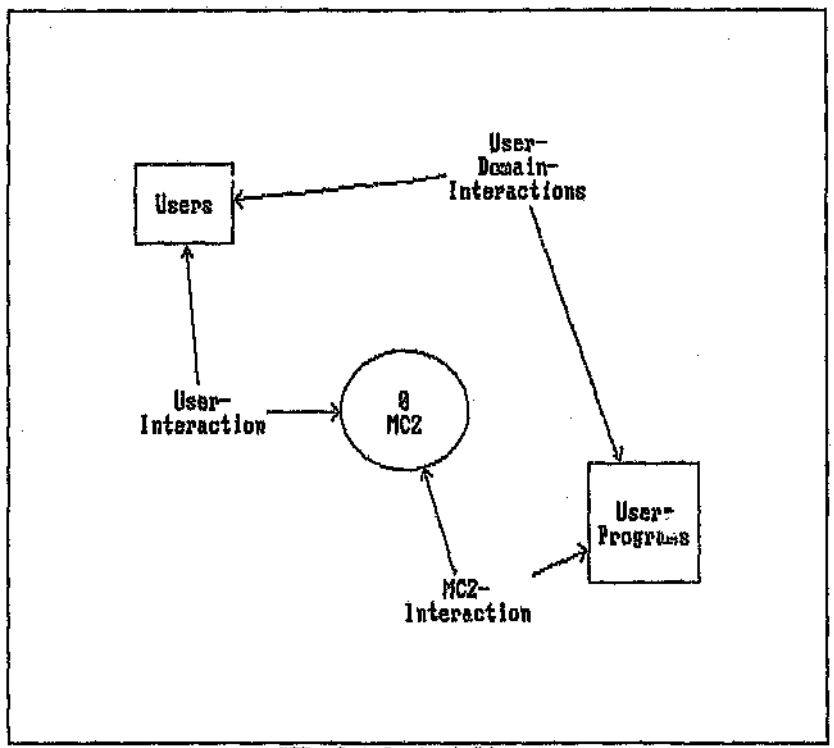
The contents of the De Marco Data Flow Design :

DFD -1: Context Diagram[1]
 DFD 0: MC2
  P-Spec[2] 1: IOP Entry Processes
  DFD 2: Helios Backbone
   DFD 2.1: Helios Remote Entry Device
    P-Spec 2.1.1: IOP Entry Device
    P-Spec 2.1.2: Domain Entry Device
    P-Spec 2.1.3: HRED Router
   P-Spec 2.2: Helios Backbone Resources
   P-Spec 2.3: Helios Backbone Services
   DFD 2.4: Domain Management Services
    DFD 2.4.1: Domain Management Services Server
     P-Spec 2.4.1.1: DMS Access Controller
     P-Spec 2.4.1.2: Logical Domain Handler
     P-Spec 2.4.1.3: DMS Helios Service Handler
    DFD 2.4.2: Domain Mapper
     P-Spec 2.4.2.1: Domain Placer
     P-Spec 2.4.2.2: Domain Switcher
     P-Spec 2.4.2.3: Translator
     P-Spec 2.4.2.4: TDS Resetter
     P-Spec 2.4.2.5: Init PhysPool
    DFD 2.4.3: Hardware Handler
     P-Spec 2.4.3.1: Switcher
     P-Spec 2.4.3.2: TxpControl
    P-Spec 2.4.4: DMS Error Control
  P-Spec 3: Domain Entry Processes

1.The context diagram of the DFD provides the context in which the MC² will operate.
2.P-Spec = Primitive Specification indicating that the process is a primitive process that is described with a PDL or Program Description Language.

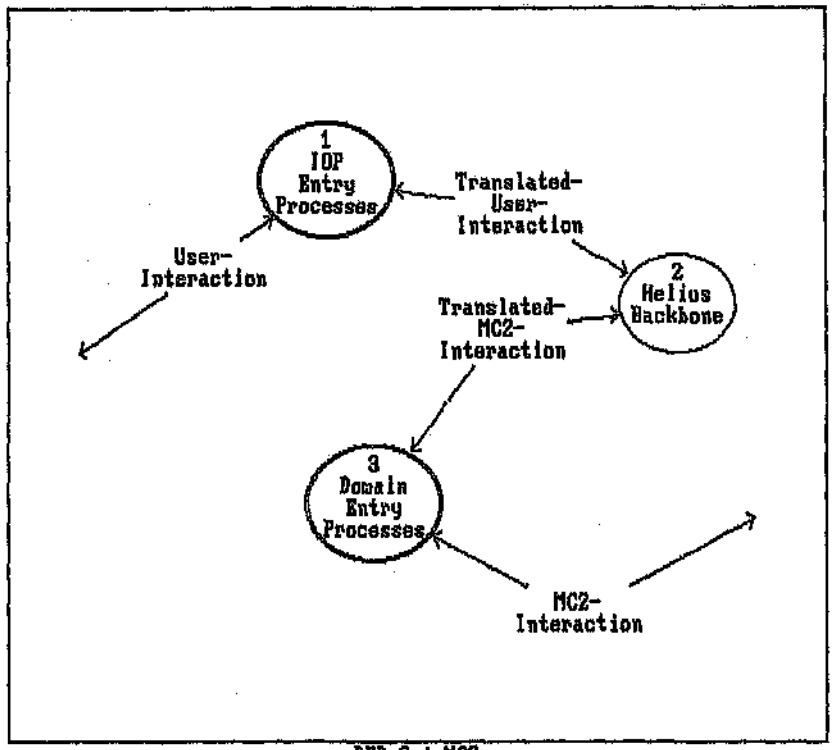
### 5.3.3.2.2 DeMarco data flow diagrams

The data dictionary appears in Appendix E. The definition modules (written for the Modula-2 implementation) were used as the primitive specifications.




DFD -1 : Context Diagram

The context diagram of the system describes either the user interaction with its programs directly, or via the MC$^2$/64 system. The MC$^2$/64 system provides the control and computing environment where the user program will run.

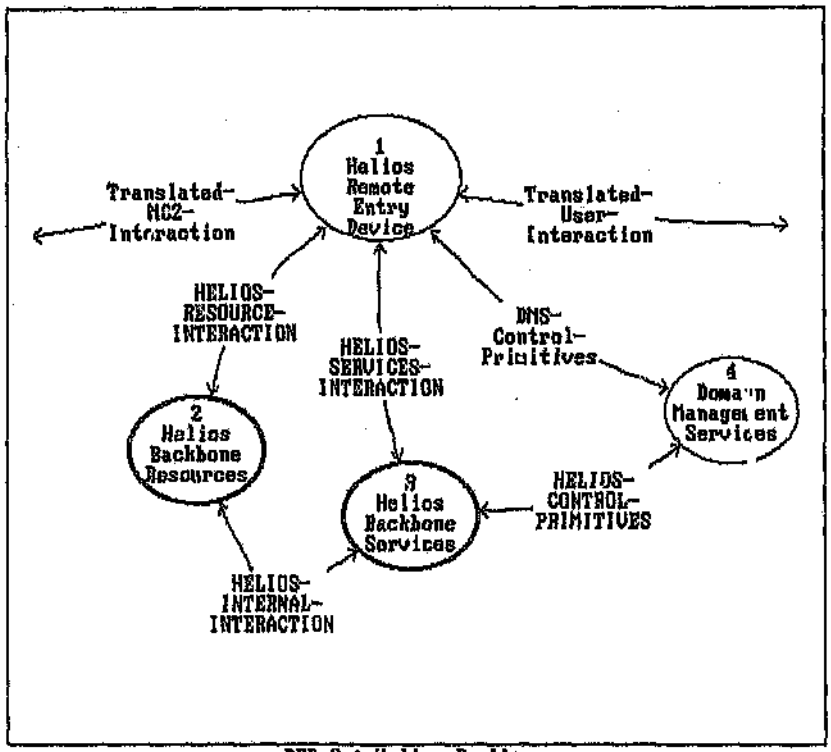



MC$^2$ (or the DMS) consists of:

The IOPEP - providing the access for the user to the Helios backbone

The HB - providing all the control and Helios facilities to make MC$^2$/64 a multi-user distributed computing environment.

The DEP - providing the access of the user programs to the Helios backbone




DFD 2 : Helios Backbone

The[1] HB[2] (Helios Backbone) consists of:

The HRED - all external communications to the services provided by the HB are done by this device.

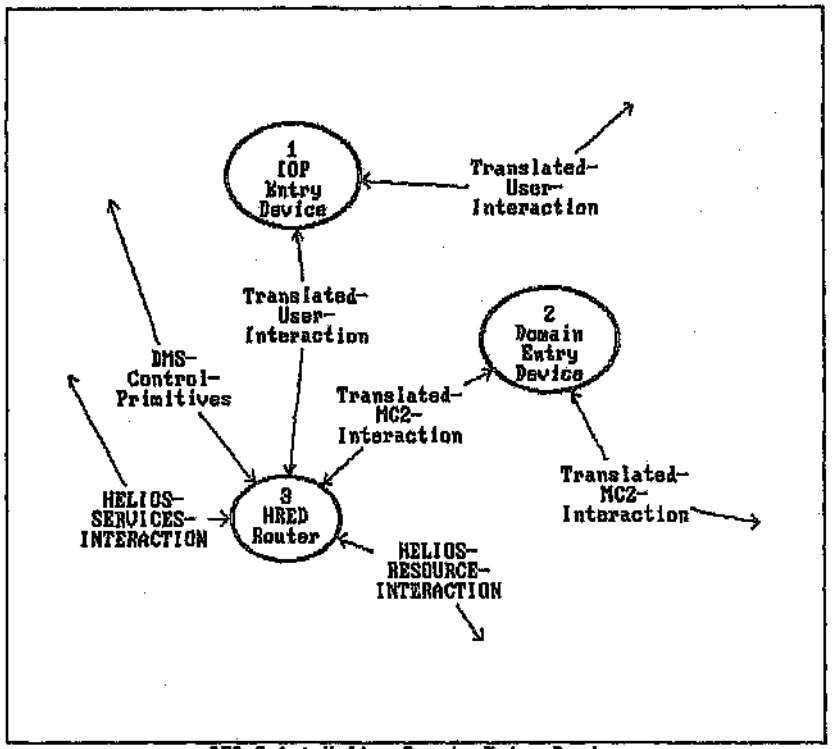
The HB Resources - providing all access to Helios resources.

The HB Services - providing all access to Helios services.

1.The Data Flows that is indicated in capital letters (such as HELIOS CONTROL PRIMITIVES in the above DFD) are primitive data flows.

2.This DFD is a breakdown of process 2 (Helios Backbone) of DFD 0.

DMS - providing all access to the Domain management services.




DFD 2.1 : Helios Remote Entry Device

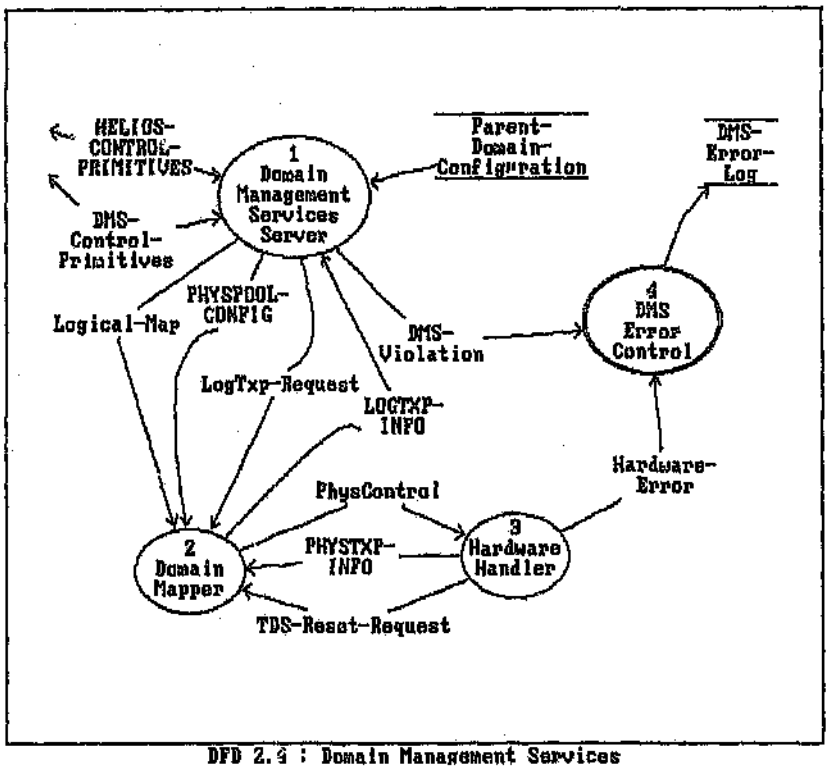
The HRED[1] (Helios Remote Entry Device) consist of:

The IOP Entry Device - providing the interface with the IOP's.

The Domain Entry Device - providing the interface with the user programs running within a UD[2].

The HRED Rooter - distinguishes between the different types of service calls and route it appropriately.

1.This DFD is a breakdown of process 1 of DFD 2.
2.User Domain.




DFD 2.4 : Domain Management Services

The DMS (Domain Management Server) consists of:

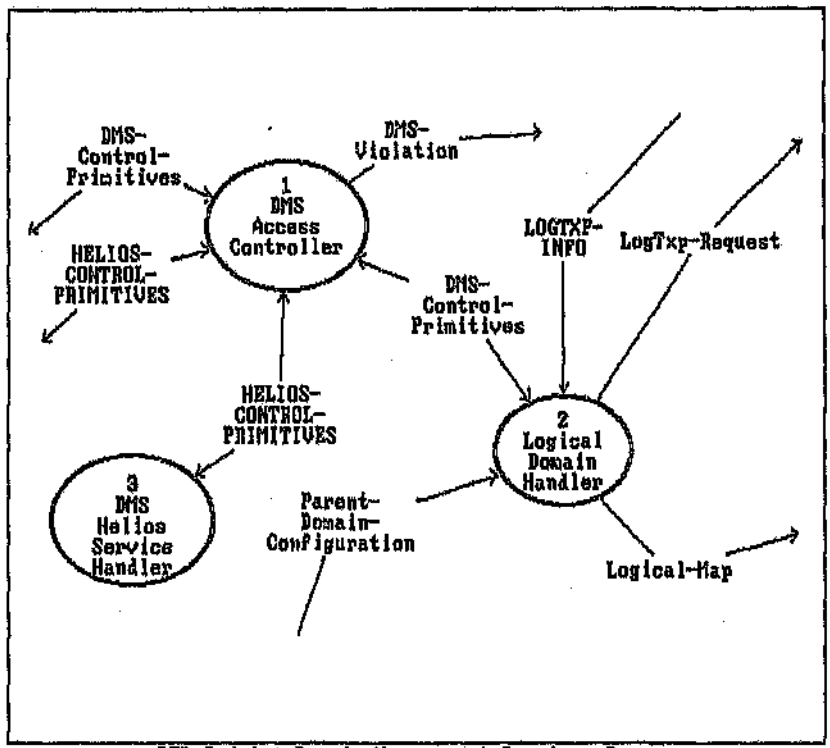
The DMSS - providing all the control and information on the UD's. The DMSS also is responsible for the initialisation of the PD from the PDConfiguration file.

The Domain Mapper - maps logical to physical domains.

The Hardware Handler - provides the low level control of the PD[1].

The DMS Error Control - logs all fatal errors.

1.Parent Domain



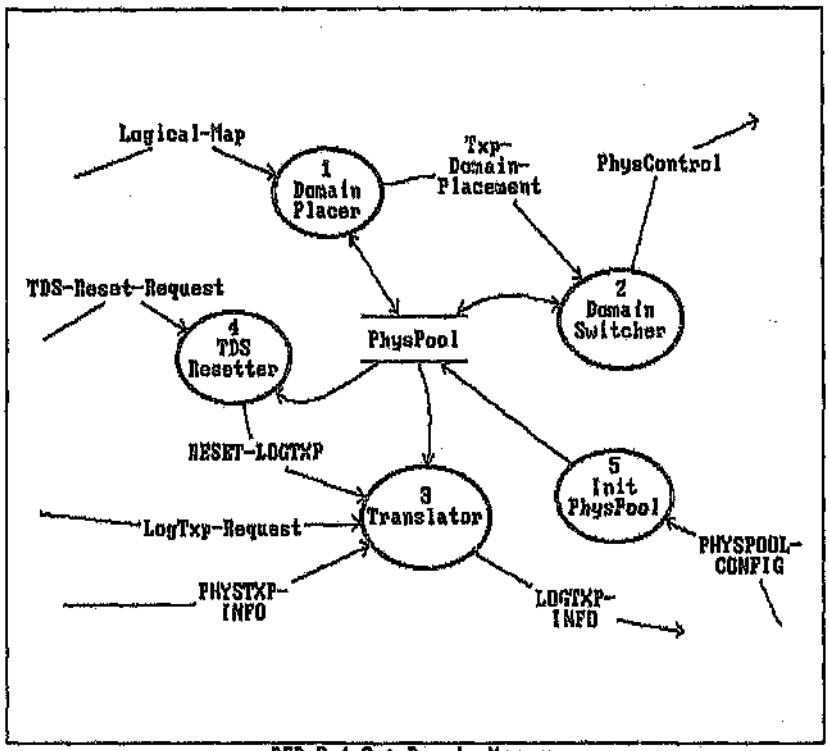

DFD 2.4.1 ' Domain Management Services Server

The DMSS (Domain Management Services Server) consists of:

The DMS Access Controller - provides security and integrity to the DMS.

The DMS Helios Service Handler - handles all the Helios housekeeping services.

The Logical Domain Handler - provides all the services for maintaining the logical domains.




DFD 2.4.2 : Domain Mapper

The Domain Mapper[1] consists of :

The Domain Placer - places a logical user domain onto the physical TP. This process must maintain the effective switchability of the system.

The Domain Switcher - switches the placed user domain according to the requested user graph.

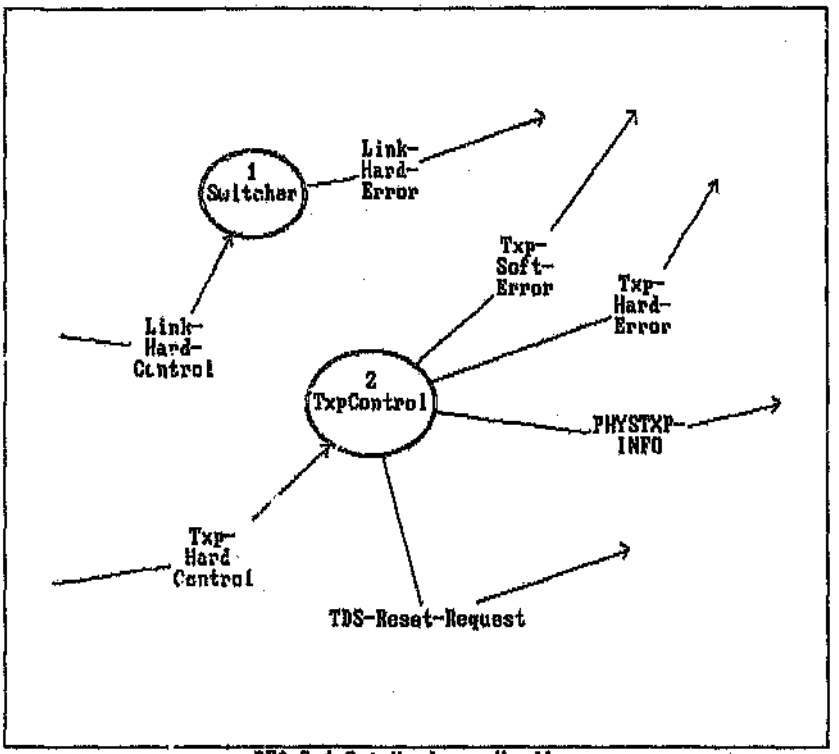
The Translator - translates the logical transputer addresses to a physical transputer.

1.This DFD is a breakdown of process 2 of DFD 2.4.

The TDS Resetter - is a hardware simulation that have to be built into the system to handle the poorly designed TDS reset.

The PhysPool - the data structure containing all the mapping information from the logical to the physical.

The Init-PhysPool - this process initialises the PhysPool with the data from the DMSS.




DFD 2.4.3 : Hardware Handler

The Hardware Handler[1] consists of

The Switcher - switches the User, Clos and Cluster switches.

The TxpHardControl - handles the physTxp control.

1.This DFD is a breakdown of process 3 of DFD 2.4.

## 5.3.4 Coding and unit testing

### 5.3.4.1 Implementation of the MC²/64 system software

#### 5.3.4.1.1 The DMS

The DMS is the Domain Management System running on the control pipeline of the MC²/64 system. The DMS controls all the transputers and resources in the system. As the DMS must control the parent domain of MC²/64, the DMS is a sequential system with a single user entry point.

The DMS consists of different modules :

* *HRED* - The Helios Remote Entry Device that polls all the user links from the user IOP's for user commands. When a user command is received, the command is serviced completely before another command is received. The HRED is therefore a potential bottleneck in the system, but the DMS controls only the domain. After allocation and switching, the user uses his domain independently from the DMS. The DMS does not influence user software executing on the domain in any way.

* *Domain allocater* - this module handles the user requests for a domain. The user requests are checked against the previously defined user names and user capabilities.

* *Domain mapper* - this module handles the mapping of the logical user domain to the physical transputer domain. This module therefore incorporates the placing and switching algorithms explained in the

previous chapters of this dissertation. This module must always translate the logical user information to the physical information, and the logical to physical map is hidden in this module.

* *Low level hardware dependent modules* - these are the modules which directly interface with the hardware of MC$^2$/64. These modules handle all the MC$^2$/64 switches and the different transputers in each MC$^2$ Cluster.

The operating system that was used to implement the system software is Helios Version 1 and later versions (see Appendix C on Helios). The Rowley Modula-2 Compiler with the Helios definition modules was used.

The modularity of Modula-2[1] lends itself very favourably to the implementation of such a system as the DMS of MC$^2$/64. Although different modules were implemented by different users, the interfaces between modules were clearly defined by the definition modules and there

1.Modula-2 is a descendent of its direct ancestors which are Pascal and Modula. Pascal was designed as a general purpose language and has gained a large user base since its implementation in 1970. Modula emerged from experiments in multiprogramming and therefore concentrated on aspects relevant to that field of application.

In 1977, a research project with the aim of designing a computer system (hardware and software), was launched at the *Institut fur Informatik* of ETH, Zurich. This system was to be programmed in a single high-level language. This language therefore had to satisfy requirements of high-level system design as well as those of low-level programming for parts that closely interact with hardware.

Modula-2 includes all aspects of Pascal, but extends some features of Pascal. The main additions with regard to Pascal are:

* The *module* concept. A module can be split into a *definition* and *implementation* part.
* A more systematic approach which facilitates the learning process.

  *Low-level facilities* which make it possible to breach the rigid type consistency rules and allow the mapping of data with Modula-2 structure onto a store without inherent structure.
* *The procedure type* which allows procedures to be dynamically assigned to variables.

was no problem at all to get the different modules linked together during the integration phase. Each module could be tested separately, which was a tremendous aid to the overall debugging of the system. The clearly defined modules and interfaces, as well as the strict type checking of Modula-2 also aided in the implementation of a DMS which could be debugged, integrated and supported easily, even though the code was generated by different programmers.

As mentioned, the DMS runs on the Helios Backbone or Control Pipeline of the MC$^2$/64 (see section 2.5.3 MC$^2$/64). The Control Pipeline consists of the system controller and the four cluster controllers. The main modules of the DMS run on the system controller, while the cluster control modules run on the cluster controllers. Modules communicate through the Helios message-passing system with messages.

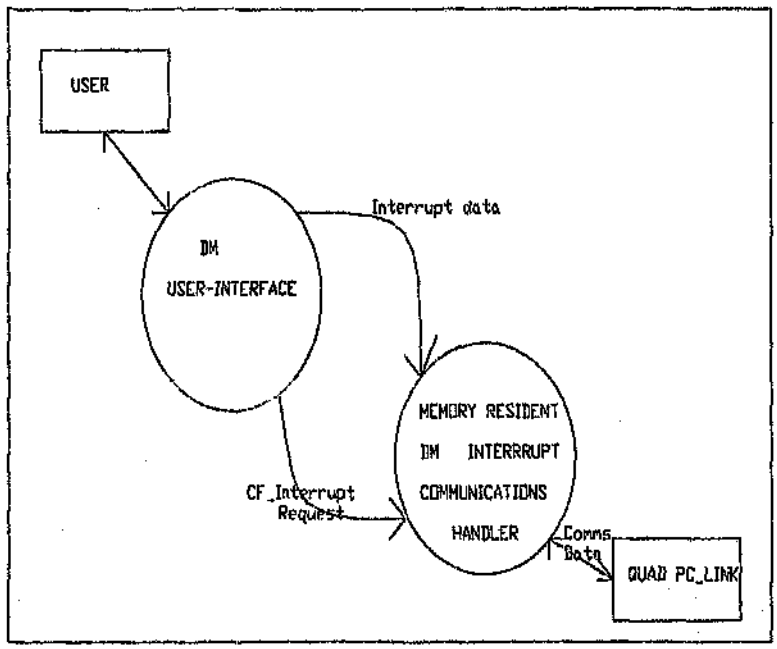
#### 5.3.4.1.2 The DM (Domain Manager)

The domain manager is the control software of MC$^2$/64 user domains running on an IOP (or the user PC that is used as a terminal to MC$^2$/64). The Domain Manager communicates through a link with the DMS to control the specific user domain.

The DM was implemented as two modules. The user-interface module provides a windowing user interface that guides a user through the different steps until he has a registered domain on MC$^2$/64 in the DMS. The DM user interface uses software interrupts to get access to the DM interrupt communications handler module.

The DM interrupt communications handler handles the low level communication to the Quad PC-Link Card which interfaces to the DMS through an Inmos link. Access to this module is through a software interrupt. The DM interrupt communications handler is a memory

resident module. This allows users to write applications for MC$^2$ also using the interrupts which then allows access to the DMS on MC$^2$/64. This facility allows user applications control over the user domain.




A data flow diagram of the DM shows the two main modules of the DM.

### 5.3.4.1.2.1 Windows

MS-Windows was evaluated as a possible user-interface running on the IOP systems of the DMS users. The multi-tasking capability of MS Windows makes it an ideal user-interface for such an environment as needed by the MC$^2$ systems as a user can then also use the processing power of his IOP PC. The PC is not just dedicated to the IO handling of MC$^2$. The DM can run in one window, the user application program

running on his domain in another window, and a word processor (running on the PC) for instance, in another (Windows is discussed in the appendices, section 7.4.).

It can be mentioned here that MIKOMTEK ported Helios to run under Windows. Helios can therefore now run on the user domain, in a window under MS Windows, on the user IOP.

At this stage the DM is a DOS application and not a Windows application. DOS applications can run under Windows, but as a program taking total control of the system and therefore Windows will deschedule itself. The DM will however be ported to be a Windows application in the near future.

### 5.3.5 Software integration and testing

This phase of the software development was supported tremendously by the modular approach of Modula-2. Each Modula-2 module that was developed, could be tested on its own. After debugging, it could be integrated with a previously integrated and tested module, and these two integrated modules could be tested together by just integrating them together with a test module. The interfaces of Modula-2 modules are defined by Modula-2 definition modules and the linking of different modules is therefore straightforward.

The software integration of the MC$^2$/64 system software was done in less than one day, although five programmers were involved in the development of the software and the code amounted to more than 20 000 lines. This was mainly because of the clearly defined interfaces between the different modules, and the fact that the modules could be tested separately before the final integration phase.

### 5.3.6 Software performance testing

After the integration of the MC$^2$/64 system software, the software was tested by different users of the MC$^2$/64. The system software conformed to the system specifications as different MC$^2$/64 users could allocate, switch and control their transputer domains. The user application software could be run on the diffrent user domains without the domains influencing each other. The MC$^2$/64 was also installed at a beta test site at the University of Stellenbosch, and the feedback from an environment such as a university proved valuable for future enhancements on the MC$^2$/64 systems.

### 5.3.7 Conclusions

The design and implementation of the DMS proved to be a valuable experience in the area of software engineering for large systems involving more than one programmer. The use of an appropriate design approach proved invaluable.

Another important factor that emerged from this exercise, was the use of Modula-2. Although language does not provide a fool-proof method of structured implementation, Modula-2 with its modular approach and well defined interfaces aided tremendously the successful testing and debugging of the individual modules during the implementation phase, but also during the integration and final testing of the whole system.

## 5.4 Supercomputer applications software

This section will briefly discuss some of the issues that confront programmers of applications software for supercomputers. This is beyond the scope of this

dissertation, but links closely with it as the MC2/64 could not be used without applications software making use of the powerful architecture to enhance performance[1].

The development of supercomputer applications software confronts all system developers with unique problems. One of the main drawbacks of most supercomputers is the fact that users and programmers have an inherent resistance to learning new software, be it new languages or programming techniques, or new operating systems. The whole transputer environment is new. An interesting development has been to keep user-interfaces similar. Helios for instance, is a Unix look-alike.

There are certain problems that can only run effectively on supercomputers, for example the simulation of the behaviour of electronic circuits. Chip and system designers are far from satisfied with the computer power for simulation which they can afford today, and will welcome new cheaper supercomputers, even though they will have to develop the simulation software. This is of course advantageous for highly parallel machines as they give a cheaper price/performance ratio, but their greatest disadvantage is to find software applications that can run on them making the most of the architecture, or using the machine optimally. The big obstacle that parallel computers face currently, is how to adapt existing application programs to take advantage of the parallel hardware.

An interesting new area in which supercomputers are used, is in the auto industry. Car makers are beginning to feel that they can trust collision simulations, so they do not have to crash-test many expensive prototypes. The simulation of air flow around aeroplanes will increase; this is far cheaper than using wind tunnels and prototypes to test designs.

1 *Application Software* is defined as the software that users will run on their domains. It is therefore not in any way connected or relevant to the MC2/64 DMS or DM as these systems are to do with the control of the MC2/64 system. Application software is the software that runs on MC2/64 and this software must take full advantage of the parallel architecture of MC2.

Supercomputers are also used in the field of chemicals and pharmaceuticals. Drug companies are able to simulate molecules and drug formations much more quickly and cost-effectively than they can by making them according to conventional techniques. Economic modelling is also becoming increasingly more attractive to financial institutions.

There is also a move towards the "visualization of data", to allow scientists to work with the visual models rather than mathematical or computational models. This is one area in which the small supers may prove outstanding. They can combine supercomputing power with high-performance graphics and put it at the single user's disposal.

Certain kinds of applications, like fluid dynamics modeling, are highly parallel and for these applications there are immediate advantages to be gained by using the new massively parallel supercomputers, even if the programs must be rewritten. However for other applications, the best way to make use of the parallel supercomputers is not always obvious.

Parallel programming is at an embryonic stage. Though there are a large number of parallel processors, most do not run parallel code. Most systems execute one program each per processor, not putting multiple processors to work on one program. The automatic exploitation of parallelism may not extend beyond systems with relatively few processors in the near future. In many ways, parallel programming is a more natural way to program than sequential programming, but it requires a change in thought patterns.

Approaches in parallel programming range from the assignment of each processor to a different program on a multi-user system to what amount to whole new ways of thinking for programmers. The parallel machines with the best chance of making it in the market, will apply parallel hardware in ways which are invisible to the programmer. These are the systems that can be put to use for existing applications quickly and easily. Getting applications running on a system is a vital part for any

new supercomputer, large or small. The user wants a solution, not a software porting problem, and if his software does not run on the system, he is unlikely to buy it.

Much work is being done on compilers which automatically find ways to separate program "threads" that can run in parallel. This however, does not always work, as extracting the program threads is not always the best way to parallelise a problem. In some instances it is better to try a whole new algorithm that can make the most effective use of the architecture.

Software trends which are likely to affect the chances of the commercial success of specific parallel machines can be split into (i) trends related to the present Von Neumann-dominated market, and (ii) trends relating to an eventual future market dominated by parallel processing. These trends interact deeply in that the weaker the development of parallel software, the stronger the prevalence of sequential software. The reason is simply that the transition to parallelism implies nothing less than a massive change in programmers' understanding and skills from a sequential concept of programming to a parallel concept of programming.

# 6 CONCLUSIONS

The $MC^2/64$ multi-user transputer-based system was completed in April 1990. Two of these systems are operational[1]. The network control design and implementation formed an integral part of the realisation of these systems.

## 6.1 Configurability

Two of the design goals for $MC^2$ were modularity and configurability. These were the goals which influenced the network control of $MC^2/64$ the most. To keep the whole system configurable, each connection component coupling the different transputer nodes has to be configurable as well.

The $MC^2$ Cluster houses up to 16 transputers and uses two 48X48 crossbar switches to connect these transputers. Extending the algorithm of the Esprit project team, the $MC^2$ project team produced $MC^2$ Clusters which are completely configurable. The Esprit algorithm is based on the extraction of Euler cycles from the requested graph.

The $MC^2/64$ system consists of four $MC^2$ Clusters connected together using a modified Clos network. The Clos network provides a number of paths between any two clusters in the system, and is in no way blocking. However, since $MC^2$ Clusters release only 16 links per Euler colour for inter-cluster connections, the $MC^2/64$ system is not completely configurable.

The *placing algorithm* was introduced to the system to enhance the user configurability of the system. This algorithm attempts to minimise the use of the restricted resources by allocating the transputers a user requests in one cluster where possible. As an $MC^2$ Cluster is completely configurable, the transputers in a cluster

1.February 1991.

can be connected into any configuration regardless of connections already made. This algorithm ensures complete user configurability from a homogeneous $MC^2/64$ system as the usage of restricted resources is minimised.

Different aspects influencing user configurability are also discussed in this dissertation. These aspects include the homogeneousness of the system, and the way in which users currently request a domain from $MC^2/64$. By optimising these factors, the user configurability is enhanced.

The configuration simulation MAPSIM shed light on the behaviour of a non-homogeneous $MC^2/64$ system, as well as the way in which user requests influence user configurability. The homogeneousness of the system affected the user configurability tremendously as it is not possible to minimise the usage of the restricted resources (inter-cluster links) if the requested transputers are located in different clusters.

## 6.2 Network control software

The network control algorithms were implemented as part of the DMS. The design issues included a study on suitable design approach. The DMS was designed using the DeMarco data flow methodology. The DMS could be designed with this methodology because of its nature, but not all systems could be addressed sufficiently by using the DeMarco methodology. Other aspects of systems such as real-time constraints and control flow analysis are addressed in some way by other methodologies, and the ideal design approach would be a conglomerate of methodologies each describing some aspect of the system. The disadvantage of using such a design approach includes a steep learning curve, as well as a sound knowledge of how methodologies interact.

The use of Modula-2 as the implementation language aided the project team tremendously towards tested and debugged modules. As the module interfaces of Modula-2 are clearly defined, the final integration and testing of the system was painless.

## 6.3 Areas for further research

Further research could not really be undertaken on the network control or configurability of $MC^2$ systems. The new $MC^2$ systems would contain $MC^2$ Clusters ensuring complete configurability.

Further research can definitely be undertaken in the area of parallel software design and development. This is currently a very active research field which will influence the future of high performance computing.

# 7 APPENDICES

## 7.1 Appendix A - Link propagation through a C004

The Inmos communication link is a high speed interconnect which provides full duplex communication between members of the Inmos transputer family, according to the Inmos serial link protocol. The Inmos C004 is a transparent programmable link switch designed to provide a full crossbar switch between 32 link inputs and 32 link outputs.

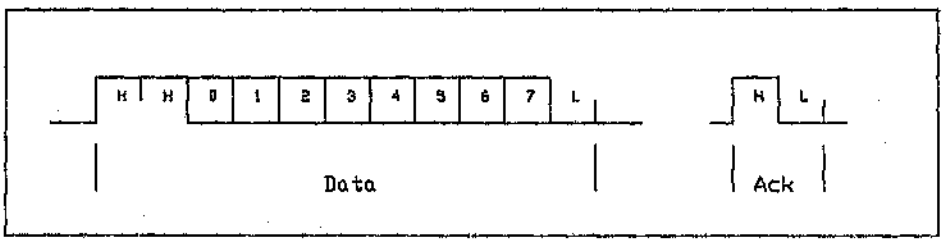
The IMS C004 will switch links running at either the standard speed of 10 Mbits/sec, or at the higher speed of 20 Mbits/sec. It introduces, on average, a 1.75 bit time delay on the signal. Link switches can be cascaded to any depth without loss of signal integrity and can be used to construct reconfigurable networks of arbitrary size. The switch is programmed via a separate link called the *configuration link*.

### 7.1.1 Links

Inmos bidirectional serial links provide synchronized communication between Inmos products and the outside world. Each link comprises an input channel and an output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both the data and the control information.

A receiver can transmit an acknowledge as soon as it starts to receive a data byte. In this way the transmission of an acknowledge can be overlapped with receipt of a data byte to provide continuous transmission of data. This technique is fully compatible with all other Inmos transputer family links.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte, the sender waits for the acknowledge which consists of a high start bit followed by a zero bit. The acknowledge signifies that the receiving link is able to receive another byte.




IMS C004 link data and acknowledge packets

### 7.1.2 Delay through a C004

A single C004 inserted between two transputers fully implements the overlapped acknowledges and causes no reduction in data bandwidth, the delay through the switch being hidden by the overlapped acknowledge. The maximum number of C004s's through which a link can be routed before a noticeable reduction in bandwidth occurs, is 5 (total in both directions as the acknowledge signal is also delayed through a IMS C004). This is important in the design of large systems, using a number of C004's to realise the link switch devices.

## 7.2 Appendix B - Supercomputer operating systems

* DISTRIX

  This is a port of UNIX to the transputer as undertaken by the Laboratory for Advanced Computing at UCT. DISTRIX is only a minimal implementation of UNIX and it has not been commercially released.

* MERCURY

  A single user operating system

* TROS

  TROS is a fault-tolerant OS kernel aimed for stand-alone real-time applications.

* VXS

  Very little is known about this OS other than that it provides an operating environment on transputers.

* QIX

  QIX is a Linda-based OS with a MACH kernel.

* SURFACEWARE

  This is a kernel providing run-time support for applications running on the MEIKO computing surface. It is not a full-blown operating system. Applications developed within this kernel are not easily ported to non-MEIKO systems.

* EXPRESS

  EXPRESS is a parallel operating system but it does not fully support multi-user systems, stand-alone filing systems, LAN and third party vendor support.

* TROLLIUS

TROLLIUS is an attempt by academics at the Cornell Theory Centre of the Cornell Research Foundation and the Research Computing of the Ohio State University to provide an industry standard operating system for transputers as well as opening various issues of research within truly distributed operating systems in distributed memory processor networks. It is available in source format from the Ohio State University and this appears to be the main reason for its survival. It is therefore more of an academic operating system rather than an industrial one and it is very much still under development. Trollius does have an industry version called GENESIS.

* MEIKOS

MEIKOS is a tried and tested implementation of UNIX, based on the AT&T System 5.3 and Berkley 4.3, providing full multi-u· support (LAN access, security, filing system, parallel source debuggers etc.). It does restrict the parallel system to MEIKO. MEIKOS drives only the MEIKO computing surface.

* GENESIS

GENESIS is the operating system for TRANSTECH systems. GENESIS relies on a SunOS host to provide the disk-storage and multi-user access. It therefore requires a Sun workstation as front-end processor.

* TransIDRIS

PARSYS supports TransIDRIS as an industry standard in Europe. It conforms to the IEEE POSIX standard and is aUNIX 5.3 and Berkley 4.3 look-alike which provides multi-user support, XWindows, SCSI, parallel source debuggers and other high-level support, as well as giving LAN accessibility. It is reliable and secure with regard to multi-user security and provides third party vendor support.

* HELIOS

PERIHELION wants to see Helios as Europe's industry standard. Helios, as TransIDRIS, conforms to the IEEE POSIX standard, and is UNIX 5.3 and Berkley 4.3 look-alike (although HELIOS only conforms to the sequential sections of POSIX and to a limited extent the parallel/communications sections). It provides multi-user, XWindows, SCSI, parallel source debuggers and other high-level support, as well as LAN accessibility. It is reliable and secure with regard to multi-user security and provides third party vendor support. More third party vendors at this stage support Helios than the other operating systems.

Helios does not require a host machine and it provides a UNIX-like system with POSIX compatibility in most system calls. POSIX compatibility therefore exists in the file system, but not in communications and parallelism where the POSIX standard was designed for shared memory systems and not distributed memory systems. A POSIX standard call would result in a poor efficiency for Helios on distributed parallel platforms. Helios introduces its own efficient structure to deal with these aspects of parallelism and communications. Multi-user, TCP/IP, XWindows and LAN support are well provided for, as well as a host of compilers such as C, Fortran, Modula-2, ADA, Occam, Linda, Pascal and a parallel symbolic debugger.

## 7.3 Appendix C - The Helios operating system

Helios is an operating system specifically designed for the transputer, and was intended from the start to run on multiple processors. Although appearing similar to Unix at the user level the underlying implementation is entirely different in order to handle this multiple processor environment. Helios was therefore designed from the start as a distributed operating system. The original implementation was intended for the transputer but the design allowed for other processors with different communication mechanisms and memory management.

Helios implements functions which are largely compatible with the proposed Posix standard (IEEE Std 1003.1-1988) and therefore Helios is a Unix look-alike. Certain groups of functions, however, have been omitted, or their specifications altered slightly because they make assumptions about the implementation of the system which are invalid under Helios. Experience has shown that it is reasonably easy to port an existing Unix program to Helios, normally taking about the same time as a port from one flavour of Unix to another.

### 7.3.1 Multi-tasking under Helios

The multi-tasking nature of Helios is based on the concept of making the entire Helios system highly sympathetic to the design philosophy of the transputer. Helios supports two types of multi-tasking: processes and tasks. A process is also called a thread, and processes run on the same chip. A process shares memory with other processes, and the switch from one process to another is very quick. On the transputer a process maps directly onto Occam processes supported by the firmware of the chip. Processes are scheduled by the chip, not by Helios.

The second object type is called a task which is also called a process under Unix. A task has a name, and is created by a Helios system call. Different tasks may not share memory and communicate through message passing. Different tasks can be placed on different processors.

The tasks can communicate with each other, from within the program code, using standard reads and writes over channels of communication called *pipes*. The use of standard reads and writes for communication means that Helios tasks can be written in standard languages such as C, Fortran and Pascal. Tasks written in different languages can be mixed together. The routing of pipes between tasks, across the network of processors, is handled automatically by Helios.

A group of related tasks is called a *task force* (related refers to tasks which work together within a single application). Helios has a task force manager which resides on the root processor, automatically distributing tasks across the processors in the network. The network is the collection of processors available for parallel programming.

A *parallel program* consists of many self-contained parts (which can be sub-routines) which run simultaneously on different processors. Helios has been designed for parallel programming. The smallest unit of parallelism in Helios is called a task. A task is a self-contained unit (i.e. program) which has been separately compiled and linked. Helios then provides an environment which enables more than one task to be running at any one time, either all running on the same processor or distributed amongst many processors.

For Helios to be distributed across the network, it needs to know what the network is. This information includes what processors and resources are available and how they are connected. This information is provided in a *resource map file*. The *network server* runs on the root processor which is the server connecting the network to the host computer. The network server (or network manager) reads the resource map file before loading Helios onto the network. Normally all the processors in the network will contain the Helios nucleus.

The *task force manager* also uses information from the resource map file. Before the task force manager can place a task on the network, it needs to know what

resources the task needs and what resources are available. The task force manager tries to satisfy all the task force requirements while trying to distribute the tasks equally among the processors (load balancing).

### 7.3.2 Client/Server model

Helios is based on the client/server model. All tasks are either clients or servers. Server tasks control access to the system resources such as disks, screens and keyboards. Client tasks are application programs which access the system resources by sending requests to the appropriate servers. A client task which needs to read from a file on disk would send a read request to the IO server. The server would then access the file and return the requested data to the client.

Helios servers are distributed across the network of processors in the system. Normally a server is located on the processor which is attached to the resource that the server is controlling. Client tasks can be located anywhere in the network, and do not need to know where the server is located.

### 7.3.3 Distributed operating system

Helios is called a distributed operating system because it is distributed across the processor network. The central core of the Helios operating system is known as the *nucleus*. Normally every processor in the network has at least a copy of the nucleus. The nucleus loads and schedules tasks on a processor and enables the tasks to communicate with each other.

### 7.3.4 User interface

There are two main user interfaces with Helios. The first is the shell, which provides a command line interface at which commands and parameters may be typed. The other is the graphical interface provided by Xwindows, although the

shell runs within an Xwindows window in this case as well.

### 7.3.5 Languages

An operating system is of no use unless applications can be written for it, and the first step in producing an application is to have available the language in which it is written. With an application intended for a parallel architecture machine there is also the requirement that it must be possible to use multiple processors.

Some languages are specifically designed for parallel computers. The most famous example is the language Occam, and here the language has constructs which map directly onto the transputer hardware. In many ways the fact that Occam does this makes it the most difficult to support under Helios, as Helios wishes to control the use of the hardware (such as the transputer links).

Helios itself is written in C, and a new C compiler conforming to the proposed ANSI standard was written as part of the Helios project. The C compiler supports the standard language with no explicit extensions for parallelism. Helios C provides parallelism through the use of system calls. These calls may be used by programmers in C, or in any other language running under Helios. Other compilers which are already available under Helios are Pascal, Fortran, Modula-2, C, Ada, Prolog and Strand-88.

### 7.3.6 Communication

The tasks need to communicate efficiently with each other in some way. The most common way of performing this communication is by means of pipes, which provide a bidirectional communication channel.

The pipes used for communication can be set up in two ways. The easiest way is to let Helios do this for you. The Helios Component Distribution Language,

CDL, provides a way to describe a parallel program; it also provides pipes between the tasks of which it is comprised. Alternately the programmer may choose to open the pipes directly.

All Helios input and output eventually maps onto messages. It is also possible to send messages directly from one task to another. In this case the pipe is still opened, because only in that way can Helios create valid message ports in the processors involved.

## 7.4 Appendix D - Microsoft Windows

MS Windows is an operating system running under DOS 2.0 or later, generally on IBM PC or compatible type machines. The installed base of IBM PCs and compatibles running MS DOS is currently more than 10 million.

For the user Windows provides a multi-tasking graphical-based windowing environment that runs programs especially written for Windows and some current programs written for MS-DOS. Many other programs written for MS-DOS can run under Windows but will not be windowed or multitasked.

Programs written especially for Windows have a consistent appearance and command structure and are thus often easier to learn and use than conventional MS-DOS programs. Users can easily switch between different programs running under Windows and exchange data between them.

For the program developer, Windows provides a wealth of built-in routines that allow the easy implementation of menus, dialog boxes, scroll bars and other components of a friendly user interface. Windows also contains an extensive graphics programming language that includes the use of formatted text in a choice of fonts. Programmers can treat the keyboard, mouse, video display, printer, system timer and RS-232 communication ports in a device-independent manner. Windows programs run identically on a variety of hardware configurations.

For the future, Windows is also an integral part of the new protected mode operating system developed by Microsoft and IBM, OS/2. Under this system, Windows is called the Presentation Manager (PM). Microsoft expects OS/2 to establish foundations that will carry the PC-compatible microcomputer industry for the next ten years. The OS/2 Presentation Manager is seen as the principle environment for the OS/2 application programs.

Windows was originally announced by Microsoft Corporation in November 1983 and released two years later, in November 1985. The first version of Windows was Windows 1.01.

### 7.4.1 "Visual Interface"

In contrast to the command line interface of MS-DOS, Windows provides a visual interface. The initial screen of Windows provides the user with nearly all the functionality of the old command line DOS interface.

The concepts behind this type of visual interface date from the mid-1970s with work done at the Xerox Palo Alto Research Center (PARC). The interface was first used in the Xerox Alto and later in the 8010 Start Information System. In particular, the Star (introduced in April 1981, four months before IBM introduced the original PC) has the look and feel that is instantly recognizable to Windows users. Xerox PARC took a revolutionary approach in designing a user interface - they did real research using actual naive users rather than relying on the instincts of programmers.

The visual interface pioneered at Xerox PARC was brought into the mainstream and popularized by Apple Computers, first in the ill-fated Lisa and a year later in the Macintosh, introduced January 1984. The Apple Macintosh remains the only real challenger to IBM's dominance in the business market. It is not the hardware but the operating environment that makes the machine so appealing to users. The Mac is simply easier to use and learn than an IBM PC running MS-DOS. The IBM PC running Windows meets the Mac challenge.

Users do not need to spend a lot of time learning to use the computer or mastering a new program (a very urgent need), as Windows programs have the same fundamental look and feel. The program is identified by the caption and most program functions are initiated through the program menus. A user is prompted for additional information through a dialog box. This dialog box is kept similar or nearly similar in all Windows programs.

From the programmers perspective, the consistent user interface results from using routines built into Windows for constructing menus and dialog boxes. All menus have the same keyboard and mouse interface because Windows handles this job rather than the application program.

### 7.4.2 Multi-tasking

Although some people question whether multi-tasking is really necessary on a single-user computer, users definitely are ready for multi-tasking and can benefit from it. The popularity of MS-DOS RAM-resident pop-up programs (such as SideKick) proves it. Pop-ups are not strictly speaking multi-tasking programs, but they provide fast context switching, and this involves many of the same concepts as multitasking (from the user's side).

Several Windows can be displayed and running at the same time. Each program occupies a rectangular window on the screen. A user can move the windows around on the display, change their size, switch between programs and transfer data from one program to the other.

As Windows runs under the single-tasking MS-DOS, it does not employ the most traditional form of multi-tasking, which is pre-emptive multi-tasking based on the system clock. Instead, Windows uses "nonpre-emptive multi-tasking" that still allows programs to run nearly simultaneously but avoids problems that would result from treating MS-DOS as if it were a multi-tasking system. Because OS/2 is a real multi-tasking operating system, the OS/2 PM use pre-emptive multi-tasking.

An operating system cannot implement multi-tasking without also doing something about memory management. As new programs are started up and old ones terminate, memory can become fragmented. The system must be able to consolidate free memory space, which requires that it must move blocks of code and data in memory.

Programs running under Windows can overcommit memory. A program can contain more code than can fit into memory at any one time. Windows can discard code from memory and later reload the code from the program's .EXE file. A user can run several copies (instances) of the same program. All these instances have the same code in memory. The 640KB memory limit of the PC's architecture is effectively stretched without requiring additional memory.

Programs running under Windows can share routines located in other .EXE files. The files that contain these shared routines are called dynamic link libraries. Windows includes the mechanism to link the program with the routines in the dynamic link library at run time.

### 7.4.3 Disadvantages

Windows is not a perfect environment for the PC under DOS as it needs a lot of disk space and memory. Windows can be run on a standard XT but in this mode it is practically unusable as it is too slow. Windows needs at least an 80286-based machine with 1 MByte of memory.

Another disadvantage is the steep learning curve from the programmer's side. Windows has 450 function calls and for a person with experience of programming in the DOS environment, the concepts of Windows are difficult to grasp.

## 7.5 Appendix E - Data dictionary of the DMS

### 7.5.1 Data dictionary

```
**********************
* Data Dictionary *
**********************

Add-Ext = TBD

Add-Txp = TBD

AddConn = TBD
```

```
Address = TBD

Alloc-Domain = TBD

Analyse-Txp = TBD

BreakConn = TBD

Buffer = TBD

Buffer-Length = TBD

ClearConn = TBD

Close-DMS = TBD

Close-Domain = TBD

Cluster-Number = TBD

DMS-Control = Open-DMS + Close-DMS +
              Alloc-Domain + Free-Domain + Get-Domain-Desc +
              Switch-Domain + Open-Domain + Close-Domain +
              Add-Txp + Del-Txp +
              Add-Ext + Del-Ext +
              Reset-Txp + Analyse-Txp +
              Read-Txp-DPM + Write-Txp-DPM + Performance-Txp

DMS-Control-Primitives = DMS-ALLOC-DOMAIN + DMS-FREE-DOMAIN +
                         DMS-SWITCH-DOMAIN + DMS-GET-DOMAIN-DESC +
                         DMS-OPEN-DOMAIN + DMS-CLOSE-DOMAIN +
                         DMS-ADD-TXP + DMS-DEL-TXP +
                         DMS-ADD-EXT + DMS-DEL-EXT +
                         DMS-RESET-TXP + DMS-ANALYSE-TXP +
                         DMS-WRITE-TXP-DPM + DMS-READ-TXP-DPM +
                         DMS-GET-TXP-PERFORMANCE

DMS-Error-Log = TBD

DMS-Violation = TBD

DPM = Address + Buffer-Length + Buffer


Del-Ext = TBD

Del-Txp = TBD

Domain-Number = TBD

External-Link = TBD

Free-Domain = TBD

Get-Domain-Desc = TBD

Get-Txp-Performance = TBD

HardWare-Error = Txp-Hard-Error + Txp-Soft-Error + Link-Hard-Error


Initialise-Txp-Pool = TBD

Link-Hard-Command = AddConn + BreakConn + ClearConn
```

```
Link-Hard-Control = Link-Hard-Command + PhysLinkMap

Link-Hard-Error = TBD

LogLink = TBD

LogTxp = Domain-Number + Txp-Number

LogTxp-Command = ADD-LOGTXP + DEL-LOGTXP +
                 ANALYSE-LOGTXP + RESET-LOGTXP +
                 WRITE-LOGTXP-DPM + READ-LOGTXP-DPM +
                  GET-LOGTXP-PERFORMANCE

LogTxp-Request = LogTxp-Command + LogTxp

Logical-Map = Mapper-Command + 0(LogTxp)999 +
              0(External-Link)999 + 0(LogLink)999

MC2-Interaction = DMS-Control + HELIOS-SERVICES

Mapper-Command = Initialise-Txp-Pool + Switch-Domain

Open-DMS = TBD

Open-Domain = TBD

Parent-Domain-Configuration = TBD

Performance-Txp = TBD

PhysControl = Txp-Hard-Control + Link-Hard-Control

PhysLink = TBD

PhysLink-Map = TBD

PhysLinkMap = 0(PhysLink)999

PhysPool = PhysTxp-Map + PhysLink-Map

PhysTxp = Cluster-Number + Txp-Position

PhysTxp-Map = TBD

Read-Txp-DPM = TBD

Reset-Txp = TBD

Switch-Domain = TBD

TDS-Reset-Request = TBD

Translated-MC2-Interaction = [DMS-CALLS + DMS-RESPONSES] +
                             [HELIOS-CALLS + HELIOS-RESPONSES]

Translated-User-Interaction = [DMS-CALLS + DMS-RESPONSES] +
                              [HELIOS-CALLS + HELIOS-RESPONSES]

Txp-Domain-Placement = TBD
```

```
Txp-Hard-Command = Reset-Txp + Analyse-Txp +
                   Read-Txp-DPM + Write-Txp-DPM +
                   Get-Txp-Performance

Txp-Hard-Control = Txp-Hard-Command + 0(PhysTxp)999 + DPM

Txp-Hard-Error = TBD

Txp-Number = TBD

Txp-Position = TBD

Txp-Soft-Error = TBD

User-Domain-Interactions = TBD

User-Interaction = DNS-Control + HELIOS-SERVICES

Write-Txp-DPM = TBD
```

## 7.6 Appendix F - Definition module

The definition module DOMMAP.DEF is an example of the definition modules of the DMS that were used as the Primitive Specifications of the DeMarco Design of the DMS.

```
(*****************************************************************************)
DEFINITION MODULE DomMap;
  (*                  Copyright(c) 1989 - MIKONTEK NC'                       *)
  (*                    NCC Domain Mapper definition module                  *)
  (*                                                                         *)
  (* Author : A.J. Gerber                                                    *)
  (* Version : 0.0                                                           *)
  (* File : DOMMAP.DEF                                                       *)
  (* This module contains all the functions to map the logical domain of a  *)
  (* user onto the physical hardware. This module will place and switch a    *)
  (* requested user graph and because all the physical information is        *)
  (* located in this module, all the translation of logical addresses to     *)
  (* physical addresses will be handled by this module. This module also     *)
  (* handles the TDS Reset.                                                  *)
  (*                                                                         *)
  (* Revision record                                                         *)
  (*                                                                         *)
  (*   0.0    1989-09-21 by A.J. Gerber                                      *)
  (*          1.  Version 0.0 is this version described above                *)
  (*                                                                         *)
  (* The following procedures are implemented :                              *)
  (*     InitPhysPool                                                        *)
  (*     MapDomain                                                           *)
  (*     ReleaseDomain                                                       *)
  (*     AnalyseLogTxp                                                       *)
  (*     ResetLogTxp                                                         *)
  (*     FreezeLogTxp                                                        *)
  (*     GetLogTxpError                                                      *)
```

```
  (*     GetLogTxpPerform                                                  *)
  (*     WriteLogTxpDPM                                                    *)
  (*     ReadLogTxpDPM                                                     *)
  (*     TDSResetRequest                                                   *)
  (*                                                                       *)
  (* The following procedures are to be implemented :                      *)
  (*     AddLogTxp                                                         *)
  (*     DelLogTxp                                                         *)
  (*     AddLogExt                                                         *)
  (*     DelLogExt                                                         *)
  (*                                                                       *)
  (*************************************************************************)

(* IMPORTS ***************************************************************)
  FROM DNSErrCodes IMPORT ErrorCode;
  FROM DNSConst    IMPORT NumOfTxps,
                          TxpLinks;
  FROM SYSTEM      IMPORT BYTE;
(* TYPES *****************************************************************)
  TYPE
    IOPNum       = CARDINAL;
    TxpNum       = LONGCARD;
    ExtNum       = LONGINT;
    DomNum       = LONGCARD;         (* The Domain Number *)
    TxpLinkNum   = LONGCARD;         (* Link Numbers can be from 0..3 *)
    PhysTxpNum   = CARDINAL;

    TxpTypeDesc  = RECORD
                     TNum      : TxpNum;  (* Domain number, amount or physpos *)
                     Processor : LONGCARD;  (* As defined by PDConfig *)
                     Memory    : LONGCARD;  (* In Kilo Byte *)
                   END;

    TxpConnTos   = (ToTxp, ToExt, ToIOP, ToNone);
    TxpConnDesc  = RECORD
                     CASE To : TxpConnTos OF
                       ToTxp :
                         Txp : TxpNum;
                         Lnk : TxpLinkNum;
                     | ToExt :
                         Ext : ExtNum;
                     END;
                   END;

    TxpConns     = RECORD
                     Txp       : TxpTypeDesc;
                     Group     : CARDINAL;
                     Static    : BOOLEAN;
                     Link      : ARRAY[0..TxpLinks-1] OF TxpConnDesc;
                   END;

(* PROCEDURES ************************************************************)
PROCEDURE InitPhysPool ( AmountTxps   : CARDINAL;
                         TxpDesc      : ARRAY OF TxpTypeDesc) : ErrorCode;
    (*********************************************************************)
    (* This procedure initialises the PhysPool data structure containing *)
    (* all the physical information of the system such as how the Txp's  *)
    (* are switched, the logical as well as physical address of each Txp, *)
```

```
    (* the external link connections etc.                                  *)
    (*                                                                     *)
    (* Inputs                                                              *)
    (*   * The Number of Transputers                                       *)
    (*   * The information on each Txp : Transputer type, amount of memory *)
    (*         and the physical position in the system.                    *)
    (*                                                                     *)
    (* Outputs :                                                           *)
    (*    InitPhysPool                                                     *)
    (*       = 0 - No error                                                *)
    (*       < 0 - Could not initialise PhysPool - system must be rebooted *)
    (*               INVALID DATA                                          *)
    (*                                                                     *)
    (* Side effects - None                                                 *)
    (*                                                                     *)
    (***********************************************************************)

PROCEDURE MapDomain(     Domain     : DomNum;
                         AmountTxps : CARDINAL;
                     VAR ProcLnks   : ARRAY OF TxpConns) : ErrorCode;
    (***********************************************************************)
    (* This procedure switches the domain in the requested way. The        *)
    (* transputers of a domain will be placed and switched in such a way   *)
    (* that the system resources are used optimally.                       *)
    (*                                                                     *)
    (* Inputs                                                              *)
    (*  * The Domain Number used to identify the domain.                   *)
    (*  * The number of transputers in the domain.                         *)
    (*  * The link connections of each transputer.                         *)
    (*                                                                     *)
    (* Outputs                                                             *)
    (*    MapDomain :                                                      *)
    (*         = 0 : No error                                              *)
    (*         < 0 : Could not be switched                                 *)
    (*                 DATA ERROR                                          *)
    (*                 CANNOT BE SWITCHED BY SYSTEM                        *)
    (*                                                                     *)
    (* Side effects                                                        *)
    (*       None                                                          *)
    (*                                                                     *)
    (***********************************************************************)

PROCEDURE ReleaseDomain( Domain : DomNum) : ErrorCode;
    (***********************************************************************)
    (* This procedure releases the requested domain by disconnecting and   *)
    (* deallocating all the transputers of the given domain.               *)
    (*                                                                     *)
    (* Inputs                                                              *)
    (*   * The Domain Number used to identify the domain.                  *)
    (*                                                                     *)
    (* Outputs                                                             *)
    (*    ReleaseDomain :                                                  *)
    (*         = 0 : No error                                              *)
    (*                                                                     *)
    (* Side effects                                                        *)
```

```
    (*       None                                                          *)
    (*                                                                     *)
    (***********************************************************************)

PROCEDURE AddLogTxp( Domain : DomNum;
                     Txp    : TxpNum;
                     Link   : TxpConns) : ErrorCode;
    (***********************************************************************)
    (* This procedure adds a transputer to the domain of the user. The way *)
    (* in which this transputer must be connected to the domain is also    *)
    (* supplied and the transputer is placed and switched in the requested *)
    (* way by this procedure through the procedures supplied by the module *)
    (* Switcher.def                                                        *)
    (*                                                                     *)
    (* Inputs                                                              *)
    (*   * The Domain number used to identify the domain.                  *)
    (*   * The logical transputer number used to identify the transputer.  *)
    (*   * The link connections of the added transputer.                   *)
    (*                                                                     *)
    (* Outputs                                                             *)
    (*   AddLogTxp :                                                       *)
    (*       = 0 : No error                                                *)
    (*       < 0 : Could not be switched                                   *)
    (*               DATA ERROR                                            *)
    (*               CANNOT BE SWITCHED BY SYSTEM                          *)
    (*                                                                     *)
    (* Side effects                                                        *)
    (*       None                                                          *)
    (*                                                                     *)
    (***********************************************************************)

PROCEDURE DelLogTxp( Domain : DomNum;
                     Txp    : TxpNum ) : ErrorCode;
    (***********************************************************************)
    (* This procedure deletes a Transputer from the user domain. Domain    *)
    (* and Txp (logical address of the transputer) is translated to a      *)
    (* physical address and with the aid of the procedures in the module   *)
    (* switcher.def, the transputer is deleted from the user domain        *)
    (*                                                                     *)
    (* Inputs                                                              *)
    (*   * The Domain number used to identify the domain.                  *)
    (*   * The logical transputer number used to identify the transputer.  *)
    (*                                                                     *)
    (* Outputs                                                             *)
    (*   DelLogTxp :                                                       *)
    (*       = 0 : No error                                                *)
    (*       < 0 : Could not be deleted                                    *)
    (*               DATA ERROR                                            *)
    (*               CANNOT BE DELETED BY THE SYSTEM                       *)
    (*                                                                     *)
    (* Side effects                                                        *)
    (*         None                                                        *)
    (*                                                                     *)
    (***********************************************************************)
```

```
PROCEDURE AddLogExt( Ext    : ExtNum;
                     Domain : DomNum;
                     Txp    : TxpNum;
                     Link   : TxpLinkNum) : ErrorCode;
   (*************************************************************************)
   (* This procedure adds a external link to the domain of the user. The    *)
   (* way in which the link must be connected to the domain is also         *)
   (* supplied and the link is placed and switched in the requested         *)
   (* way by this procedure through the procedures supplied by the module   *)
   (* Switcher.def                                                          *)
   (*                                                                       *)
   (* Inputs                                                                *)
   (*   * The External Link Number                                          *)
   (*   * The Domain number used to identify the domain.                    *)
   (*   * The logical transputer number used to identify the transputer.    *)
   (*   * The link number of the transputer to which the link must be added *)
   (*                                                                       *)
   (* Outputs                                                               *)
   (*   AddLogExt :                                                         *)
   (*      = 0 : No error                                                   *)
   (*      < 0 : Could not be switched                                      *)
   (*               DATA ERROR - TXP NOT AVAILABLE                          *)
   (*               DATA ERROR - EXT LINK NOT AVAILABLE                     *)
   (*               DATA ERROR - TXP LINK NOT AVAILABLE                     *)
   (*               CANNOT BE SWITCHED BY SYSTEM                            *)
   (*                                                                       *)
   (* Side effects                                                          *)
   (*       None                                                            *)
   (*                                                                       *)
   (*************************************************************************)

PROCEDURE DelLogExt( Domain : DomNum;
                     Ext    : ExtNum) : ErrorCode;
   (*************************************************************************)
   (* This procedure deletes an external link from the indicated user       *)
   (* domain. The link is deleted from the user domain with the aid of the  *)
   (* procedures in the module Switcher.def                                 *)
   (*                                                                       *)
   (* Inputs                                                                *)
   (*   * The Domain number used to identify the domain.                    *)
   (*   * The External Link Number                                          *)
   (*                                                                       *)
   (* Outputs                                                               *)
   (*   DelLogExt :                                                         *)
   (*      = 0 : No error                                                   *)
   (*      < 0 : Could not be deleted                                       *)
   (*               DATA ERROR                                              *)
   (*               COULD NOT BE DELETED BY SYSTEM                          *)
   (*                                                                       *)
   (* Side effects                                                          *)
   (*         None                                                          *)
   (*                                                                       *)
   (*************************************************************************)
```

```
PROCEDURE AnalyseLogTxp( Domain : DomNum;
                         Txp    : TxpNum) : ErrorCode;
    (*************************************************************************)
    (* This procedure translates Domain and Txp (logical address) to a       *)
    (* physical transputer and through the procedure supplied by the         *)
    (* TxpCntrl module analyses the transputer.                              *)
    (*                                                                       *)
    (* Inputs                                                                *)
    (*   * The number of the domain                                          *)
    (*   * The (logical) number of the transputer in the domain              *)
    (*                                                                       *)
    (* Outputs                                                               *)
    (*   AnalyseLogTxp :                                                     *)
    (*         = 0 : No error                                                *)
    (*         < 0 : Transputer could not be analysed                        *)
    (*                   DATA ERROR                                          *)
    (*                   Receives an error from TxpCntrl                     *)
    (*                                                                       *)
    (* Side effects                                                          *)
    (*      The transputer is stopped with an analyse signal                 *)
    (*                                                                       *)
    (*************************************************************************)

PROCEDURE ResetLogTxp( Domain : DomNum;
                       Txp    : TxpNum) : ErrorCode;
    (*************************************************************************)
    (* This procedure translated Domain and Txp (logical address) to a       *)
    (* physical transputer and through the procedure supplied by the         *)
    (* TxpCntrl module resets the transputer.                                *)
    (*                                                                       *)
    (* Inputs                                                                *)
    (*   * The number of the domain                                          *)
    (*   * The (logical) number of the transputer in the domain              *)
    (*                                                                       *)
    (* Outputs                                                               *)
    (*   ResetLogTxp :                                                       *)
    (*         = 0 : No error                                                *)
    (*         < 0 : Transputer could not be reset                           *)
    (*                   DATA ERROR                                          *)
    (*                   Receives an error from TxpCntrl                     *)
    (*                                                                       *)
    (* Side effects                                                          *)
    (*      The transputer is reset                                          *)
    (*                                                                       *)
    (*************************************************************************)

PROCEDURE FreezeLogTxp( Domain : DomNum;
                        Txp    : TxpNum) : ErrorCode;
    (*************************************************************************)
    (* This procedure translated Domain and Txp (logical address) to a       *)
    (* physical transputer and through the procedure supplied by the         *)
    (* TxpCntrl module raise the reset of the transputer to freeze the       *)
    (* transputer.                                                           *)
    (*                                                                       *)
    (* Inputs                                                                *)
    (*   * The number of the domain                                          *)
    (*   * The (logical) number of the transputer in the domain              *)
```

```
    (*                                                                  *)
    (* Outputs                                                          *)
    (*   FreezeLogTxp :                                                 *)
    (*         = 0 : No error                                           *)
    (*         < 0 : Transputer could not be reset                      *)
    (*                   DATA ERROR                                     *)
    (*                   Receives an error from TxpCntrl                *)
    (*                                                                  *)
    (* Side effects                                                     *)
    (*      The transputer reset is raised to freeze the transputer.    *)
    (*                                                                  *)
    (********************************************************************)

  PROCEDURE GetLogTxpError(Domain : DomNum;
                           Txp    : TxpNum) : ErrorCode;
    (********************************************************************)
    (* This procedure translated Domain and Txp (logical address) to a  *)
    (* physical transputer and through the procedure supplied by the    *)
    (* TxpCntrl module get the error from the transputer.               *)
    (*                                                                  *)
    (* Inputs                                                           *)
    (*   * The number of the domain                                     *)
    (*   * The (logical) number of the transputer in the domain         *)
    (*                                                                  *)
    (* Outputs                                                          *)
    (*   GetLogTxpError :                                               *)
    (*         = 0 : No error                                           *)
    (*         < 0 : Transputer could not be reset                      *)
    (*                   DATA ERROR                                     *)
    (*                   Receives an error from TxpCntrl                *)
    (*                                                                  *)
    (* Side effects                                                     *)
    (*      None ?                                                      *)
    (*                                                                  *)
    (********************************************************************)

PROCEDURE GetLogTxpPerform(     Domain      : DomNum;
                                Txp         : TxpNum;
                          VAR Performance : CARDINAL) : ErrorCode;
    (********************************************************************)
    (* This procedure translated Domain and Txp (logical address) to a  *)
    (* physical transputer and through the procedure supplied by the    *)
    (* TxpCntrl module reads the performance of the transputer.         *)
    (*                                                                  *)
    (*   Performance : The variable into which the performance is returned *)
    (* Inputs                                                           *)
    (*   * The number of the domain                                     *)
    (*   * The (logical) number of the transputer in the domain         *)
    (*   * The variable into which the performance is returned          *)
    (*                                                                  *)
    (* Outputs                                                          *)
    (*   GetLogTxpPerform :                                             *)
    (*         = 0 : No error                                           *)
    (*         < 0 : Transputer performance could not be read           *)
    (*                   DATA ERROR                                     *)
    (*                   Receives an error from TxpCntrl                *)
    (*                                                                  *)
```

```
    (* Side effects                                                    *)
    (*      None                                                      *)
    (*                                                                *)
    (******************************************************************)

PROCEDURE WriteLogTxpDPM( Domain  : DomNum;
                          Txp     : TxpNum;
                          Addr    : CARDINAL;  (* < 256 *)
                          Lenght  : CARDINAL;  (* Address + Lenght < 256 *)
                          Buff    : ARRAY OF BYTE) : ErrorCode;
    (******************************************************************)
    (* This procedure translates the Domain and Txp (logical address of the *)
    (* transputer to a physical address. The Dual Port Memory of the  *)
    (* debug interface card is then written at the Addr address with the  *)
    (* data supplied by Buff.                                         *)
    (*                                                                *)
    (*   Address : Address to write at                                *)
    (*   Lenght  : Data length in bytes                               *)
    (*   Buffer  : The data to write                                  *)
    (* Inputs                                                         *)
    (*   * The number of the domain                                   *)
    (*   * The (logical) number of the transputer in the domain       *)
    (*   * The Address to write at                                    *)
    (*   * The Data length in bytes                                   *)
    (*   * The data to write                                          *)
    (*                                                                *)
    (* Outputs                                                        *)
    (*   WriteLogTxpDPM :                                             *)
    (*              = 0 - No error                                    *)
    (*              < 0 - Could not write DPM                         *)
    (*                    DATA ERROR                                  *)
    (*                    SYSTEM COULD NOT WRITE                      *)
    (*                                                                *)
    (* Side effects                                                   *)
    (*   The DPM is changed on the specified transputer.              *)
    (*                                                                *)
    (******************************************************************)

PROCEDURE ReadLogTxpDPM(     Domain  : DomNum;
                             Txp     : TxpNum;
                             Addr    : CARDINAL;  (* < 256 *)
                             Length  : CARDINAL;  (* Address + Lenght < 256 *)
                         VAR Buff    : ARRAY OF BYTE) : ErrorCode;
    (******************************************************************)
    (* This procedure reads the Dual Port Memory from the debug interface  *)
    (* card of the specific transputer whose logical address is supplied by *)
    (* Domain and Txp. The DPM information is stored in the Buff variable.  *)
    (*                                                                *)
    (* Inputs                                                         *)
    (*   * The number of the domain                                   *)
    (*   * The (logical) number of the transputer in the domain       *)
    (*   * The Address to write at                                    *)
    (*   * The Data length in bytes                                   *)
    (*   * The variable to store the data                             *)
    (*                                                                *)
    (* Outputs                                                        *)
    (*   ReadLogTxpDPM :                                              *)
```

```
    (*            >= 0 - Performance.                                    *)
    (*            <  0 - Cannot read DPM                                 *)
    (*                   DATA ERROR                                      *)
    (*                   SYSTEM COULD NOT READ                           *)
    (*                                                                   *)
    (* Side effects                                                      *)
    (*   None.                                                           *)
    (*                                                                   *)
    (*********************************************************************)

PROCEDURE TDSResetRequest( PhysTxp : PhysTxpNum ) : ErrorCode;
    (*********************************************************************)
    (* This procedure handles the TDS Reset Request. The PhysTxp variable *)
    (* is translated to the root transputer of a domain running the TDS.  *)
    (* All the transputers in that domain is then resetted to emulate the *)
    (* TDS Reset.                                                        *)
    (*                                                                   *)
    (* Inputs                                                            *)
    (*   * The physical address of the root transputer of the TDS domain. *)
    (*                                                                   *)
    (* Outputs                                                           *)
    (*   TDSResetRequest :                                               *)
    (*            = 0 : No error                                         *)
    (*            < 0 : Cannot issue a TDS Reset                         *)
    (*                    INVALID TXP NUMBER                             *)
    (*                    NOT ROOT TRANSPUTER OF DOMAIN                  *)
    (*                                                                   *)
    (* Side effects                                                      *)
    (*       All the transputers in the domain of the user running the TDS *)
    (*       is resetted so that a TDS Reset is emulated.                *)
    (*                                                                   *)
    (*********************************************************************)

END DomMap.
```

# 8 REFERENCES

Boehm, B.W., Software Engineering, *IEEE Transactions on Computers*, Vol.C-25, no.12, December 1976.

Bokhari, S.H., On the Mapping Problem, *IEEE Transactions on Computers*, Vol.C-30, no.3, March 1981.

Du Plessis, A.L., *A Software Engineering Environment for Real-Time Systems*, Vol.1, Dissertation for Doctor of Philosophy, UNISA, June 1986.

Du Plessis, A.L., *A Software Engineering Environment for Real-Time Systems*, Vol.2, Dissertation for Doctor of Philosophy, UNISA, June 1986.

Hill, G., *Designs and applications for the IMS C004*, Inmos technical note number 19, Inmos Ltd.

Hockney, R.W. and Jesshope, C.R., *Parallel Computers 2, Architectures, Programming and Algorithms*, Adam Hilger - IOP Publishing Ltd, Second edition 1988.

Inmos, *IMC C004 programmable link switch*, Inmos C004 Data Sheet, Inmos, April 1987.

Nicole, D.A., Lloyd, E.K., Ward, J.S., Switching Networks for Transputer Links, *Proceedings of the 8th Occam User Group Technical Meeting*, pp.147-65, Sheffield, UK, 27-29 March 1988.

Robinson, D.C., Sloman, M.S., Domain-based access control for distributed computing, *Software Engineering Journal*, Vol.3, no.5, September 1988.

Teruel, A., On Mixing Formal Specification Styles, *Proceedings of the 4th International Workshop on Software Specification and Design: 3-4 April 1987*, pp.28-33, IEEE Comp. Soc. Press.

# 9 BIBLIOGRAPHY

Abidi, M.A., Agrawal, D.P., On conflict-free permutations in multistage interconnection networks, *Journal of Digital Systems*, Vol. V, Summer 1980, pp.115-134.

Agrawal, D.P., Graph Theoretical Analysis and Design of Multistage Interconnection Networks, *IEEE Transactiona on Computers*, C-32, July 1983, pp.637-648.

Agrawal, D.P., Testing and Fault-tolerance of multistage interconnection networks, *Computing*, Vol.15, Apr. 1982, pp.41-53.

Association of Computing Machinery, Special Issue on Computer Architecture, *Comm. of ACM*, Vol.21, no.1, Jan 1980.

Benes, V.E., Algebraic and Topological Properties of Connecting Networks, *Bell System Technical Journal* No. 41, 1962.

Benes, V.E., *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

Boehm, B.W., Software Engineering, *IEEE Transactions on Computers*, Vol.C-25, no.12, December 1976.

Bokhari, S.H., On the Mapping Problem, *IEEE Transactions on Computers*, Vol.C-30, no.3, March 1981.

Bowen, J., *The Formal Specification of a Microprocessor Instruction Set*, PRG Monograph 60, Oxford University Press, 1987.

BYTE, What's New International, *BYTE 64IS-54*, BYTE May 1990.

Clos, C., A Study of Non-Blocking Switching Networks, *The Bell Systems Technical Journal*, March 1953.

Cohen, B., A rejustification of formal notations, *Software Engineering Journal*, January 1989.

Cohen, B., Justification of formal methods for system specification, *Software Engineering Journal*, January 1989.

DeMarco, T., *Structured Analysis and System Specification*, Yourdon Press Computing Series, Prentice-Hall, 1979.

Dias, D.M., And Jump, J.R., Analysis and Simulation of Buffered Delta Networks, *IEEE Transactions on Computers*, C-30, Apr. 1981a, pp.273-282.

Dubois, M., *Analytical Methodologies for the Evaluation of Multiprocessing Structures*, Ph.D. Thesis, Purdue Univ., Ind., 1982.

Dugdale, D.A., Megaflops for Minibucks, *Defense Electronics*, August 1987, pp.68-76.

Du Plessis, A.L., *A Software Engineering Environment for Real-Time Systems*, Vol.1, Dissertation for Doctor of Philosophy, UNISA, June 1986.

Du Plessis, A.L., *A Software Engineering Environment for Real-Time Systems*, Vol.2, Dissertation for Doctor of Philosophy, UNISA, June 1986.

ECS support group, *Edinburgh Concurrent Supercomputer Newsletter*, Number 3, November 1987.

ECS support group, *Edinburgh Concurrent Supercomputer Newsletter*, Number 5, June 1988.

Enslow, P.H., Multiprocessor Organization, *Computing Surveys*, Vol.9, Mar.1977, pp.103-129.

Feng, T.Y., A Survey of Interconnection Networks, *IEEE Computers*, Dec. 1981, pp.12-27.

Feng, T.Y., Parallel Processors and Processing, Special Issue, *ACM Computing Surveys*, Vol.9, no.1, Mar. 1977a.

Fisher, E.M., The Dream Machine, *Datamation*, 1 October 1986, pp.79-84.

Franklin, M., VLSI Performance Comparison of Banyan and Crossbar Communications Networks, *IEEE Transactions on Computers*, April 1981.

Gimson, R., *The Formal Documentation of a Block Storage Device*, PRG Monograph 62, Oxford University Press, 1987.

G [illegible] K., Supercomputing in the Real World, *Datamation*, 1 October 1986, pp.71-74.

Hill, G., *Designs and applications for the IMS C004*, Inmos technical note number 19, Inmos Ltd.

Hill, G., Designs and Applications for the IMS C004, *Inmos Technical Note 19*, June 1987.

Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall, 1985.

Hockney, R.W. and Jesshope, C.R., *Parallel Computers 2, Architectures, Programming and Algorithms*, Adam Hilger - IOP Publishing Ltd, Second edition 1988.

Hoey, D., And Leiserson, C.E., A Layout for the Shuffle-Exchange Network, *Proceedings of the 1980 IEEE International Conference on Parallel Processing*, August 1980.

Homewood, M., May, D., Shepherd, D., Shepherd, R., The IMS T800 Transputer, *IEEE MICRO*, October 1987.

Hwang, K., Advanced Parallel Processing with Supercomputer Architectures, *Proceedings of the IEEE*, Vol.75, no.10, October 1987.

Inmos, *IMC C004 programmable link switch*, Inmos C004 Data Sheet, Inmos, April 1987.

Inmos, *Transputer Reference Manual*, Prentice-Hall, UK, 1988.

King, Dr. Tim and Powell, J., *The Helios Operating System - Fundamentals*, Perihelion Software Ltd.

Lang, T., and Stone, H.S., A Shuffle-Exchange Network with Simplified Control, *IEEE Transactions on Computers*, C-25, Jan. 1976, pp.55-56.

Lipovski, G.J., and Malek, M., *A Theory for Multicomputer Interconnection Networks*, Tech Report TRAC-40, Univ. Of Texas, Austin, Mar. 1981.

Manuel, T., Supercomputers, *Electronics*, 3 March 1988, pp.51-56.

MEIKO - David Barr, Fax Message MEIKO, FAX 0454618188, MEIKO Ltd, 30 July 1990.

Mott, J.L., Kandel, A., Baker, T.P., *Discrete Mathematics for Computer Scientists and Mathematicians*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

Nassimi, D., and Sahni, S., A Self-Routing Benes Network and Parallel Permutation Algorithms, *IEEE Transactions on Computers*, Vol. C-30, No.5, May 1981, pp. 332-340.

Newport, J.R., An Introduction to Occam and the development of parallel software, *Software Engineering Journal*, July 1986.

Nicole, D.A., Lloyd, E.K., Ward, J.S., Switching Networks for Transputer Links, *Proceedings of the 8th Occam User Group Technical Meeting*, pp.147-65, Sheffield, UK, 27-29 March 1988.

Opferman, D., and Tsao-Wu, N., On a class of rearrangeable switching networks, *Bell Systems Technical Journal*, Vol. 50, pp.1579-1618, May-June 1971.

Payne, W.A., Makedon, F., Daasch, W.R., High speed interconnection using the Clos network, *Proceedings of the 1st International Conference on Supercomputing*, pp.96-111, Greece, 8-12 June 1987.

Perihelion, *Helios Technical Manual*, Perihelion Ltd, June 1988.

Perihelion, *The CDL Guide*, Distributed Software Ltd, Bristol, UK, January 1990.

Perihelion, *The Helios Operating System*, Perihelion Software Ltd, Prentice Hall ISBN 0-13-386004-3, 1989.

Perihelion, *The Helios Parallel Programming Tutorial*, Distributed Software Ltd, Bristol, UK, January 1990.

Pippenger, N., On crossbar switching networks, *IEEE Transactions on Communication*, Vol. COM-23, pp.646-659, June 1975.

Robinson, D.C., Sloman, M.S., Domain-based access control for distributed computing, *Software Engineering Journal*, Vol.3, no.5, September 1988.

Sanders, J.W., *An Introduction to CSP*, PRG Monograph 65, Oxford University Press, 1988.

Shepherd, R., Thompson, P., Lies, Damned Lies and Benchmarks, *Inmos Technical note 27*.

Siegel, H.J., A Model of SIMD Machines and a Comparison of Various Interconnection Networks, *IEEE Transactions on Computers*, vol C-28, pp.907-917, Dec.1979.

Siegel, H.J., Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks, *IEEE Transactions on Computers*, Vol.C-26, pp.153-161, Feb.1977.

Slotnick, D.L., Borck, W.C., McReynolds, R.C., The SOLOMON Computer, *AFIPS Conference Proceedings*, 1962.

Spivey, J.M., An introduction to Z and formal specifications, *Software Engineering Journal*, January 1989.

Supercomputing Review, Hardware Buyers' Guide 1990, *Supercomputing Review*, February 1990.

Taylor, R., Transputer Communication Link, *Microprocessors and Microsystems*, Vol.10, no.4, May 1986.

TELMAT - S. Flieller, Fax Message TELMAT Informatique - 3389742734, T.Node Parallel Computer, 2 July 1990.

Teruel, A., On Mixing Formal Specification Styles, *Proceedings of the 4th International Workshop on Software Specification and Design: 3-4 April 1987*, pp.28-33, IEEE Comp. Soc. Press.

Thompson, C.D., Generalized Connection Networks for Parallel Processor Intercommunication, *IEEE Transactions on Computers*, vol C 27, no. 12, Dec 1978, pp.1119-1126

Viljoen, N., *An overview of Transputer based computers and a description of the MC2 machine*, Division of Microelectronics and Communications Technology of the CSIR, 1989.

Waksman, A., Permutation networks, *Journal of Associated Computing*, Vol. 15, Jan. 1986, pp.159-163.

Wayman, R., OCCAM 2: An overview from a software engineering perspective, *Microprocessors and Microsystems*, Vol.11, no.8, October 1987.

Wirth, Niclaus, *Programming in Modula-2*, Third Corrected Edition, Springer-Verlag, Germany, 1985.

Wu, C.L., and Feng, T.Y., Routing Techniques for a class of multistage interconnection networks, *IEEE Transactions on Computers*, Vol.C-29, Aug.1980, pp.694-702.

Wu, C.L., and Feng, T.Y., The reverse exchange interconnection network, *IEEE Transactions on Computers*, Vol. C-29, Sept. 1980, pp.801-810.

Young, J., Supercomputer Software: The Floodgates are opening, *Electronics*, 3 March 1988, pp.75-77.