A GENERAL PURPOSE MULTI-PROCESSOR COMPUTER ARCHITECTURE

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Brian Louis Stuart, B.S. C.S./E.E.

Director

Department of Electrical and Computer Engineering

Notre Dame, Indiana

July 1987

A GENERAL PURPOSE

MULTI-PROCESSOR COMPUTER ARCHITECTURE

Abstract

by

Brian Louis Stuart

A number of approaches have been taken to using multi-processing for increasing system performance. Another, which allows the connections among the processors to be reconfigured, is presented here. The array of processors is made reconfigurable in order to achieve high performance and extendibility to large numbers of processors. Furthermore, the system is designed to be applicable to all general purpose environments.

To my loving wife Mary who was patient enough
to put up with the many months of work,
and who "encouraged" me to keep working
when I would have been otherwise lazy.

LIST OF FIGURES

PREFACE

This thesis research has been motivated by several factors. First is an interest in the various approaches to multi-processing. Much of the recent research into multi-processing has shown the applicability of particular architectures to appropriate algorithms. Unfortunately, no one architecture has proven itself highly applicable to all algorithms. A belief that a widely applicable architecture should therefore be able to switch among the useful organizations has contributed to the development of this reconfigurable architecture.

Another motivation for the development of this system arose from the price differential between microcomputers and minicomputers. It was believed that a system with minicomputer performance could be constructed using multiple microprocessors. If such a system could be built with a minimum of overhead in the processor interconnections, then it would cost significantly less than minicomputer systems.

The ideal multi-processor system has a linear speed-up in the number of processors. The field of multi-processing has seen many architectures presented in an effort to find this ‘‘ideal’’ system. The architecture presented in this thesis is not the first of these and it will certainly not be the last. It is another step in the process of developing cost effective, high performance and widely applicable multi-processor architectures.

ACKNOWLEDGEMENTS

Throughout the course of this project, many people have been of great help and support. First I would like to thank my advisor, Dr. David Cohn, for allowing me to work in a environment free of pressures about "how it is supposed to be done." He has also been the source of several good ideas leading to refinements of the system. The company that I have been working for during this period, Tellabs Inc., has, of course, been of great support financially and as a source of parts. Finally, my wife Mary has been of immeasurable support. She has been there to help in every possible capacity: wire-wrapping circuit boards, proofreading related documentation, and the all important moral support.

CHAPTER I

Introduction

...in order to attain the
maximum speed of computation,
full advantage must be made
of the methods of interposition.

– Manual of Operation
Harvard Automatic Sequence Controlled Calculator

The primary focus of this thesis project has been to develop and present a new multiple processor computer architecture. In conjunction with the development of the architecture, a small amount of conceptual work was done on a model of multi-processing in general. Also, in order to demonstrate the principles of this architecture, an example implementation has been constructed. This document will present the results of all of this work.

The architecture presented here has a number of interesting features that will be discussed. The primary new aspect is processor interaction through a reconfigurable communications network. This is done to create a high degree of interconnection while allowing the system to be extended to a large number of processors.

1.1. Structure of the Thesis

This thesis is divided into six chapters. This first chapter presents an introduction to the rest of the document. It describes in the most general terms the world of multiple processor computer systems. The second chapter reviews the work done by others in creating multiple processor architectures. An attempt has been made to include machines representative of most approaches to multi-processing. Chapter Three takes a detailed look at the motivations and objectives of multi-processing. Next, a general model of multiple processor systems is presented. A discussion of a measure of coupling as a classification of these architectures closes Chapter Three. In Chapter Four, the primary work of the thesis project is presented. The architecture is developed from the motivations that open the chapter. Chapter Four is closed by an evaluation of the architecture. In Chapter Five, the example implementation of the architecture is presented along with an example application program. Next, Chapter Six discusses some analytical characteristics of multi-processor systems. This discussion will be presented for systems in general and the new architecture in particular. Finally, conclusions of the thesis are presented in Chapter Seven. This chapter is divided into three parts. The first echos the results from Chapters Four and Five. Next, some ideas for future directions in development are presented. The closing section is a discussion of the implications that the architecture has on the software.

1.2. Fundamentals

In the course of presenting this thesis, a number of terms and concepts from the area of computer design will be used. Many of these are not universally defined.

Therefore, this section will describe some of the fundamental ideas used in general computer design. The next section introduces some of the topics used in multiple processor computer design.

The model for what is by far the most common computer organization is due to John von Neumann.[1,2] The von Neumann architecture specifies a computer composed of three parts. First, the *memory* is used to store both instructions and data. In such a system, the instructions and data are stored in the same memory space and are indistinguishable. Next, the *central processing unit* (CPU) is the portion of the system that fetches instructions from memory and executes them. Some of the instructions are used to manipulate data and others are used to control the instruction flow. The third part of a von Neumann system is the *input/output* (I/O) subsystem. This portion of the computer is the one responsible for communicating between the computer and the outside world.

For purposes of this discussion, the CPU can be considered to be made of two parts. The *instruction* or *control* unit and the *processing* unit. Instruction units, or instruction execution units, are the parts of CPUs that fetch the instructions, decode them and issue control signals to the rest of the system. Finally, the processing unit, or processing element, manipulates the data in the system under the control of the instruction execution unit.

Another concept from conventional computer design that will be used extensively in this thesis is direct memory access (DMA). In the von Neumann model of a computer, only the CPU has access to the system memory and it must perform transfers between the I/O devices and that memory. For increased performance, designers have chosen to endow I/O devices or controllers with the ability to access the memory directly. By eliminating the need for the CPU to execute instruction for these I/O transfers, the transfers may be faster and the CPU is freed to spend more time on users' programs.

Throughout this thesis, the discussion will be restricted to von Neumann architectures. In particular, only parallelism of multiple von Neumann type processors or von Neumann style control units controlling multiple processing elements will be considered here. The realm of parallelism in data flow architectures is not addressed here in any way.

1.3. An Overview of Multi-Processing

Throughout virtually all of the 48 year history of modern computing, computer designers have investigated the idea of using more than one central processing unit to cooperate on programs. There is clearly a belief that the use of multiple processors to execute various parts of the programs in parallel leads to faster execution. Intuitively, the best that one can hope for is that (in the absence of non-recurring administrative overhead) n processors will execute programs n times faster than one processor. No architecture to date has achieved this ideal case with an arbitrary mix of applications.

Many approaches have been taken to realize this vision. Some of these will be presented in the remainder of this thesis. A new approach will also be presented as the thrust of the thesis and of its associated research project. The remainder of this chapter is devoted to an overview of some of the general ideas present in many of these architectures.

1.3.1. Shared Memory: Clearly, if several processors have access to the same instructions and data, then they will have a great deal of freedom in cooperating on the task at hand. This fundamental idea leads to the concept of *shared memory,* where more than one processor does, in fact, have access to the same memory space. In effect, the memory is shared among the various processors. This concept is illustrated in Figure 1.1.

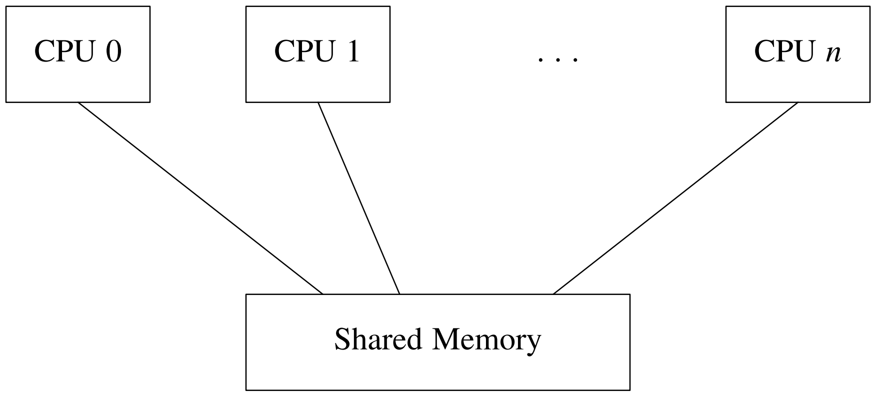

Figure 1.1

Shared Memory

In general, whenever two or more processors are connected to the same memory space, there must exist some form of arbitration to resolve simultaneous requests. This arbitration creates two disadvantages. First, the hardware that performs the arbitration adds cost to the system. Second, any time two memory accesses are attempted at the same time, one of them must wait until the other is completed. This delay degrades system performance and increases as the number of processors connected to the memory is increased. In order to minimize this delay, various methods of interleaving and banking the memory space have been devised. The system memory space can be divided into several contiguous sub-spaces called banks. Arbitration delays can then be decreased by organizing memory usage in such a way that most accesses by different processors take place in different memory banks. This division of memory into banks is done according to higher order address bits. Memory can also be divided according to lower order bits. Such a division, called interleaving, uses the sequentiality of access to alleviate the arbitration delays. If there are as many memory divisions as there are processors, and if all processors generate only sequential accesses, and if all processors generate accesses at the same frequency, the once the processors have "synced up" to the memories, there will be no more delays.

Since shared memory systems have been around for some time and since they are conceptually simple, a great deal of study has been devoted to their use. Several of the architectural and hardware approaches to multi-processing by shared memory will be discussed in the next chapter. On the software side, much work has been done in the area of the control and synchronization of the cooperating processors in a shared memory environment.

1.3.2. Multi-Processing by Communications Networks: While shared memory systems do represent a very useful and common approach to multi-processing, they do have some disadvantages. As an attempt is made to put a large number of processors into the system, the costs and delays due to the memory arbitration systems become unacceptable. Therefore, many designers have elected not to use shared memory in their systems. The general idea is that direct channels of communication can exist among the various CPUs, and that these channels can be used to pass the information needed for the processors to cooperate.

Communication channels among the processors are grouped into networks. These networks can take on many forms, several of which will be discussed in the examples of the next chapter. However, it is useful to consider a few broad classes of networks first.

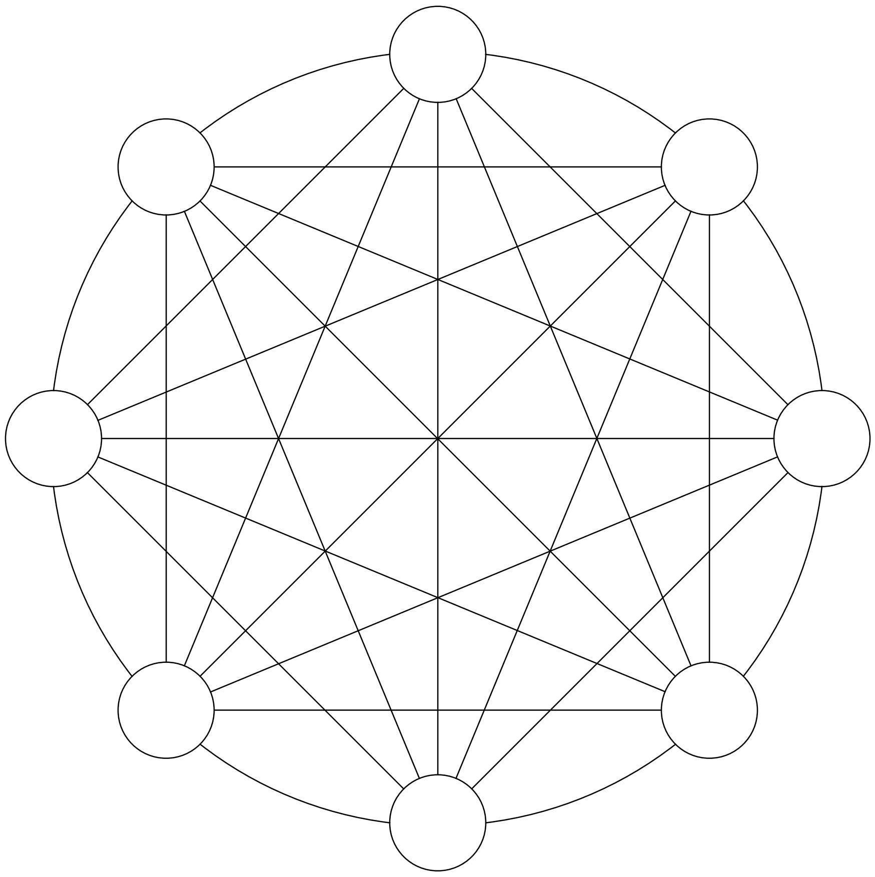

Figure 1.2

Fully Connected Mesh

The most obvious approach to combining processors (or processing elements) through communication channels is to provide a set of direct connections between selected pairs of processors. If every processor is connected to every other processor, then the network is called a *fully connected mesh* and is illustrated in Figure 1.2.

At the other extreme, the processors can be connected to form a topological *ring.* In a *ring,* each processor can send information to exactly one other processor, and it also receives information from exactly one other (different) processor. Furthermore, the *ring* has all processors participating in a closed connection. This topology is illustrated in Figure 1.3. Between these two extremes, a number of arrays, cubes and trees have been built and studied.

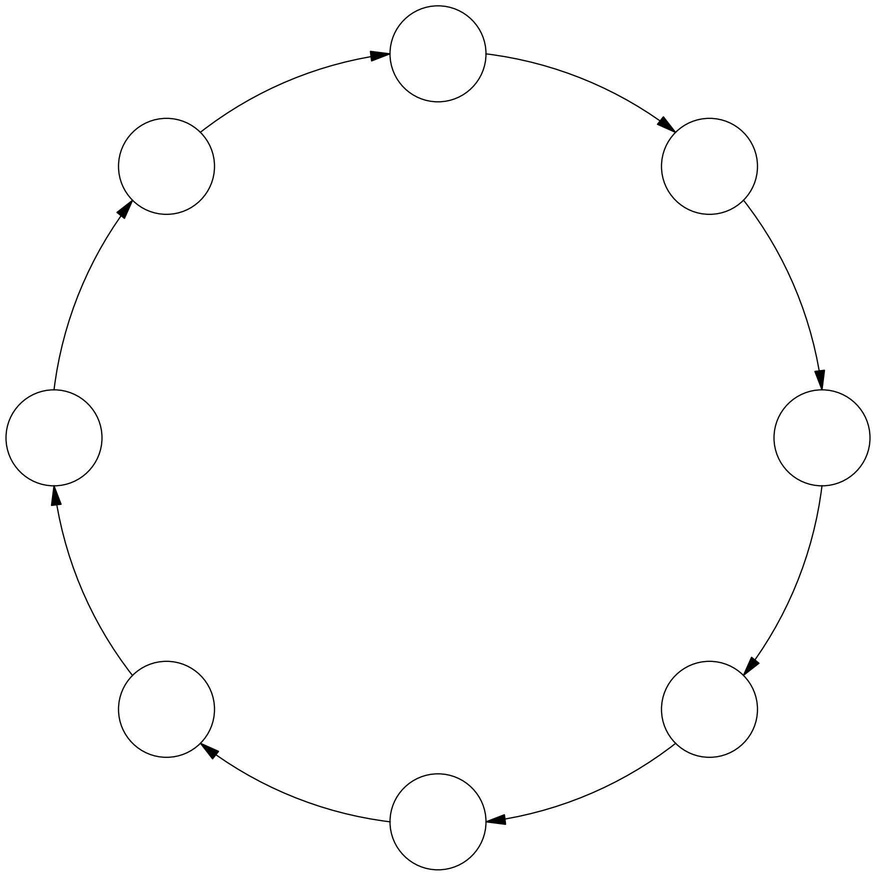

Figure 1.3

Ring Topology

Another connection approach uses an intelligent network that routes messages from source to destination processors through a limited set of physical interconnections. If one chooses a system where processors send messages to other processors, then the messages can have information regarding the source and destination processors embedded in a packet. The packets can then be routed by the network to the appropriate destination processor. Such a system is called a *packet switching network.* The limiting case of such a network is one in which all processors have one connection to the same packet switch. Such a topology is called a *star* and is illustrated in Figure 1.4. In a packet switching system, connections between processors are virtual. The messages sent between processors are routed through the network based on information present at each node along the way. Since many messages may be sharing links and nodes in the network, the system may experience queueing and arbitration delays like those discussed in regards to shared memory.

An interconnection topology which is most important for the new architecture is the *cross-point switch* or *cross-bar switch* (from its incarnation in early telephone switching systems). Conceptually, one can envision a matrix of wires in which the vertical wires are attached to ports of some devices and the horizontal wires to some other devices. If a connection is established at the point where two of these wires cross, then the device attached to the vertical wire can send information to the one on the horizontal wire. A pictorial representation of such a switch is shown in Figure 1.5.

More commonly, cross-point switches are implemented in contemporary logic through the use of multiplexers. In such an implementation, each output of the switch is the output of a multiplexer which selects from among all of the inputs. Placing an ad-

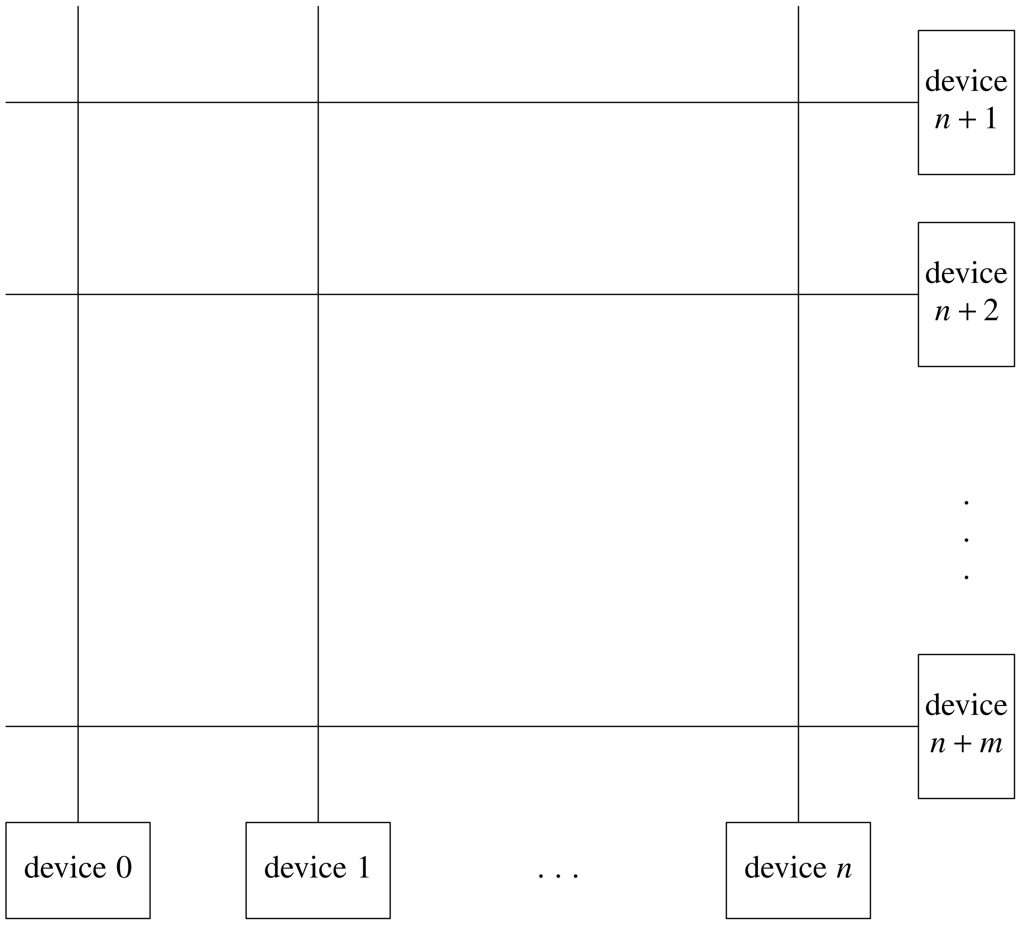

Figure 1.5

Cross-Bar Switch

dress n on multiplexer m is equivalent to connecting the m^{th} horizontal wire to the n^{th} vertical wire in the previous implementation. This cross-point implementation can be seen in Figure 1.6.

A cross-point switch can be used to connect any collection of devices. The devices on one axis of the switch can be CPUs and the other axis can be connected to memories. In such a system, the network switches address and data buses, creating a mechanism of banking or interleaving a memory space to be shared among several processors. An example of this concept called C.mmp will be discussed in the next chapter.

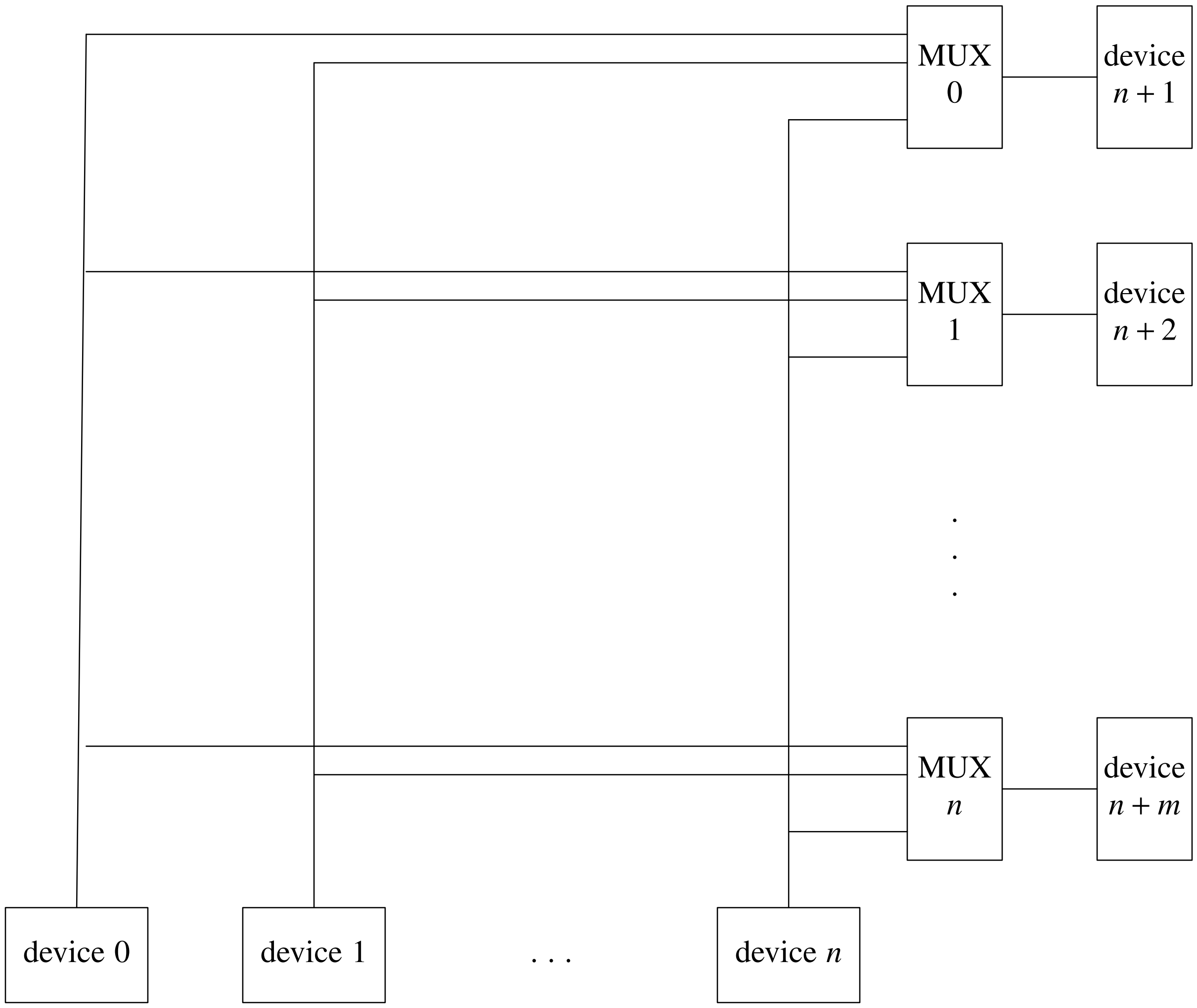

Figure 1.6

Cross-Point Switch Implementation Using MSI Logic

On the other hand, output ports of a set of processors can be connected to one side of the switch and the corresponding inputs of the same processors can be connected to the other. By properly configuring the switch in such a system, any processor can send information to any other processor. In Chapter Four, this communications topology will form the basis of a new multi-processing architecture.

These are not the only means of interconnecting CPUs to form a multi-processing system. However, in various forms they do provide the foundation for most approaches.

CHAPTER II

A Short History of Multi-Processing

... an interest in the
history of ideas is good
for the scientist's soul.

– Benoit B. Mandelbrot

As with most fields of endeavor, the creation of new ideas in computer design is best done in the context of a historical perspective. This chapter is presented for that reason. An attempt has been made to present some of the more significant and more interesting incarnations of multi-processing ideas. More importantly, a foundation has been built for the model and the architecture to be presented in later chapters.

2.1. BINAC

The idea of performing more than one operation simultaneously has been around since the beginnings of computing. The first stored program computer in the United States was the BINAC, built in 1949. This system, built by Eckert and Mauchly,[3] contained two processors in order to provide redundancy-based fault detection. The two processors would perform the same operations and compare the results. If the results matched, then processing would continue; otherwise, the system would stop.

2.2. I/O Processors

The next step in the use of multiple processors was to enhance performance. Early in computing history, it was realized that the use of the central processing unit to both control I/O devices and to process the data stream was inefficient. However, by adding another processor, one could offload the I/O processing from the main CPU. Typically, such an I/O processor would be significantly less powerful than the main CPU. This idea was the basis of the IBM I/O channels which originated with the IBM 709.[4]

In the case of channels, the I/O processor is one that is specifically designed for this purpose. Another example of a specially designed I/O processor exists in the form of the CRAY IOS for the CRAY X-MP and CRAY 1S computers.

An alternative approach is to use a small general purpose processor. The Digital Equipment Corporation used this concept with the PDP-15/76,[5]

which used a PDP-15 as the main CPU and a small PDP-11 as the I/O processor. This idea also exists today in the form of file and terminal servers in modern local area networks (LANs).

2.3. Early Commercial Multi-Processors

In an attempt to use this division of labor concept to increase the performance of the computational system, many manufacturers have produced machines with more than one CPU.

2.3.1. IBM 370/168 and Progeny: An early representative example of these machines was the IBM 370/168. This machine was available in two configurations: the attached processor (AP) configuration and the multi-processor (MP) configuration. In both cases, two processors were connected to the main memory by means of a multi-system control unit (MCU). This system was a dual-processor shared-memory system with a cache invalidation signal and an inter-processor communications channel between the two processors. In the attached processor configuration, the second CPU, called an attached processor, differed from the first processor in that it could not have I/O channels attached to it. In this respect, then, the two processors were asymmetric. The multi-processor case was distinguished by fully symmetric processors where both processors could be attached to I/O channels.[6]

IBM has more recently used this same multi-processing architecture in the 3033 multi-processor and attached processor configurations.[7]

In this system, the MCU is known as the multiprocessor communications unit and handles the cache control and inter-processor communications as well as interfacing with the main memory.

With the 3081, IBM has again changed the name and function of the MCU. In this system, it is called the system controller (SC). Also in this system, an external data controller (EXDC) is attached to the SC, replacing the channels that were connected to the individual CPUs in previous systems. Another difference with the 370/168 and 3033 systems is that the processors in the 3081 are coupled in such a way as to prevent them from being able to act as two independent single processor systems. Finally, two 3081 systems may be coupled in either an AP or MP configuration to make a four-processor 3084.[8]

The latest member of the IBM dual processor shared memory lineage is the 3090. Its multi-processor configurations improve over the 3080 series by providing multiple data paths into the shared memory space. There is also some parallelism within each processor through the use of vector processing facilities.

2.3.2. Multi-Processing by Digital: The Digital Equipment Corporation has also not been idle in producing dual processor shared memory computer systems. At about the same time IBM was introducing the 370/168, DEC introduced the DECsystem 10 Model KL-10. This version of the PDP-10 had two processors connected to as many as 16 global memory modules. Like the 370/168, the DECsystem 10 Model KL-10 could be configured in two versions. In the Master-Slave (M/S) system, all I/O devices were connected to the master processor. The slave processor did not have the ability to respond to or to control I/O devices directly. This configuration was designed for environments with an even mix of I/O and compute-bound applications. For environments where both processors need to be connected to the I/O devices, a symmetric multiprocessing (SMP) configuration was available. In this configuration, both processors had I/O controllers attached. Many of the devices were dual-ported and could be connected to both sets of controllers.[9]

Unfortunately, the two processors could not be easily used to cooperate on the same task through the shared memory. This is because there was no cache invalidation signal between the two processors.[10]

The most recent DEC implementation of commercial multi-processing is the VAX 8800. Like the other systems discussed thus far, the VAX 8800 is a dual processor system using shared memory. This machine, the most powerful ever marketed by DEC, differs from the DECsystem 10 in several important respects. First, there is hardware support for maintaining cache consistency for all memory writes.[11]

This ensures that both processors may be used to cooperate on a single task using shared data structures and shared memory inter-processor communications. There also exists a mechanism for inter-processor interrupts. Another improvement in the design is the ability for both processors to access the device controllers and to accept interrupt from them. However, no DEC operating system at present supports symmetric control of the I/O. Currently, the first processor booted is designated as the primary CPU and has sole responsibility and control for I/O processing.[12]

2.3.3. CRAY X-MP: Earlier, the CRAY IOS I/O processor was discussed as a form of multi-processing. In order to provide improved computational performance, Cray Research has introduced the CRAY X-MP, with two or four processors. The CPUs in the X-MP are functionally equivalent to CRAY 1 processors. They share a common multi-ported memory space. Additionally, there is a common CPU intercommunication section containing three clusters of shared register used for synchronization. Furthermore, both processors have access to the IOS and Solid-State Storage Device (SSD) through a CPU-I/O interface. Figure 2.1 shows the overall structure of a dual processor X-MP.[13]

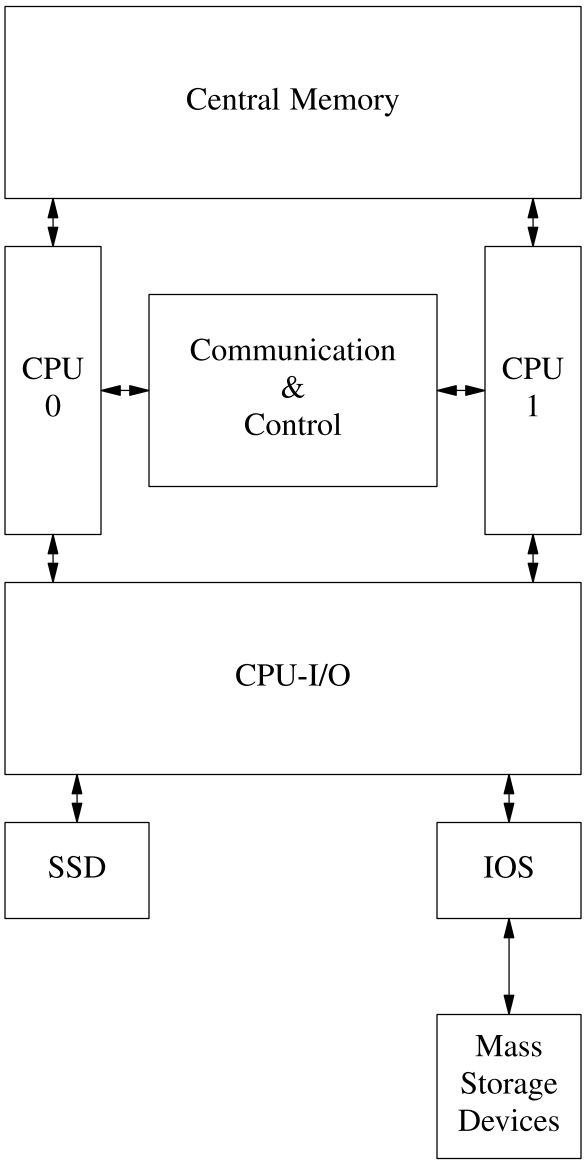

Figure 2.1

Cray X-MP System Architecture

2.4. C.mmp

Many of the “real world” constraints of manufacturing and marketing do not exist in the realm of academic research. For this reason, academic researchers are frequently at more liberty to explore more interesting (although less practical) ideas. Many times these ideas prove to be useful and applicable, although at other times they do not. Without regard for their practicality, a number of the more interesting and significant of these machines are discussed in the next few sections.

The first of these systems is the C.mmp (Computer: multi-mini processor) system designed at Carnegie-Mellon University. In this system, 16 PDP-11s are connected to as many as 16 global memory modules with a cross-point switch. Each processor also has some local memory. There is an additional interprocessor bus that is used for control functions. On this bus is a global real-time clock and a set of control function registers. These registers are used for controlling console operations on each processor. This system structure is illustrated in Figure 2.2. The I/O on the C.mmp system is associated with specific processors. While any processor may have I/O devices attached to its UNIBUS, only the attached processor can access the device’s controller. As a result, when any processor has need of an I/O operation, it must send a request to the appropriate processor.[14]

When a processor accesses a location in the global memory space, the cross-point switch passes the access request to the memory module selected according to the high order address bits. If no other processors are requesting access to the same module, the access is granted in the same memory cycle. Otherwise, all of the processors

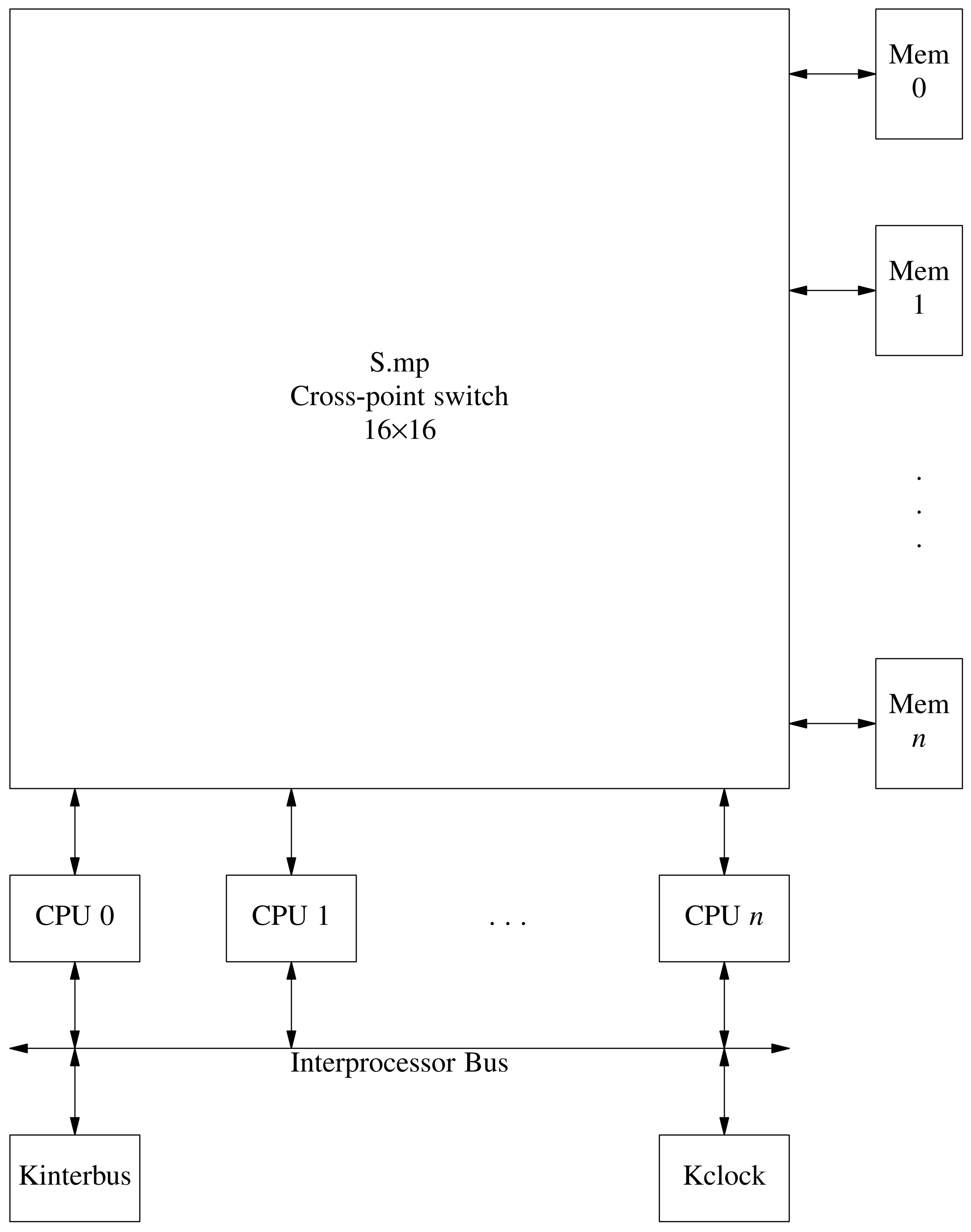

Figure 2.2

Organization of C.mmp System

but one must wait on access until those with higher priority have completed their access.[15]

Experiments done on C.mmp indicate that with highly parallel algorithms, if the executable code is placed in all of the local memory spaces of the processors, then the performance increases approximately in proportion to the number of processors. On the other hand, if the code is placed in shared memory, then with more than three processors the memory contention through the switch degrades performance more than extra processors improve it. It is for this reason that commercial shared memory systems rarely have more than two to four processors.[16]

2.5. Cm*

Another of the interesting products of research at Carnegie-Mellon is the Cm* system. The name of the system comes from the Kleene closure of computer modules (Cm's).[17]

This arbitrary growth of processors arises out of a tree structure of processor interconnections. Each of the computer modules consists of an LSI-11 processor with associated local memory.

Interprocessor communications take the form of memory shared throughout the system. Each of the processors can access the memory on each of the others through the interconnection tree. At the bottom of the tree are clusters of up to twelve Cm's. The access control for memory at each processor is called Slocal, and it arbitrates both local and remote accesses. Each Slocal in a cluster is connected to the Map bus for that cluster. Each Map bus is connected to two intercluster busses through a Kmap. The Kmap performs a message passing function among the Cm's on its Map bus as well as between its cluster and other clusters. As a result, any access by a processor to memory belonging to another processor in the same cluster only takes mapping through two Slocal's and one Kmap. Accesses to memories in other clusters take mapping through two Slocal's and two or more Kmap's.[18]

The structure of Cm* is illustrated in Figure 2.3.

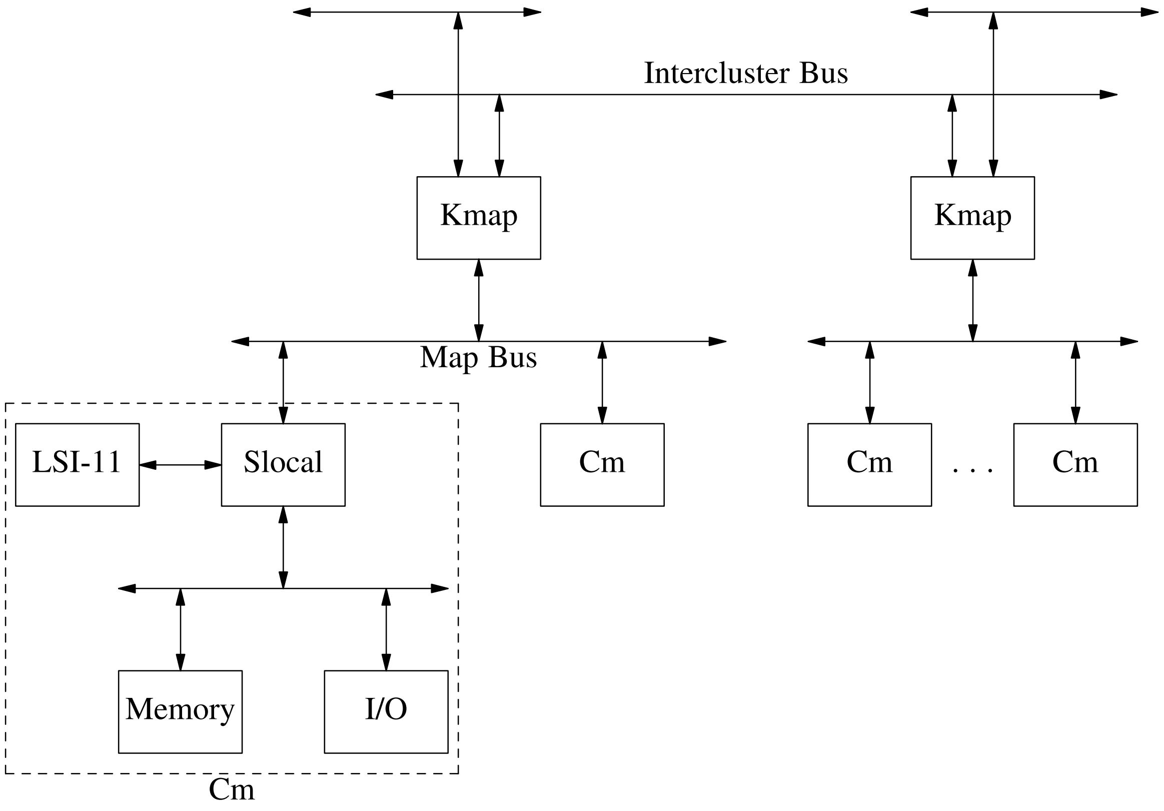

Figure 2.3

Cm*

2.6. ILLIAC IV

Up until now, all of the systems discussed have been collections of independent CPUs in the sense that they all execute their own instruction streams. In other words, the processors are not all performing the same operations at the same time. Another approach to increasing performance through multiple processing units is to provide common control of multiple processing elements from one instruction stream. While such a system cannot compute on more than one process at once, it can be very efficient at large, highly regular applications.

An example of this type of system is the ILLIAC IV, designed at the University of Illinois and built by Burroughs. The original design for the ILLIAC IV called for four quadrants, each composed of one control processor and 64 processing elements. As finished systems in this level of research rarely achieve the full grandeur of the original concept, the hardware incarnation of the ILLIAC IV had only one of the four quadrants running at half of the originally designated speed.[19]

In the original design, each of the control processors took instructions from its own stream and then broadcast control information to all 64 PEs under its control. Each of the PEs had 2048 words of local memory and four communication channels to its "neighbors."[20]

The processors were organized in a two dimensional array in such a way as to make the longest path between any two processors seven hops.

The ILLIAC IV was designed as a back-end processor for a B6500 mainframe computer system. All I/O and operating system functions were performed by this machine.[21]

2.7. MPP

Another, more extensive, example of this single control unit for multiple processing element approach is the Goodyear MPP, or Massively Parallel Processor. In this system, designed for the NASA Goddard Space Flight Center, 16,384 processing elements are arranged in a 128×128 square array. Each 1-bit processing element has an associated 1024-bit local memory and a 100ns cycle time. Also, all processing elements except those on the outside edges are connected to their four nearest neighbors in the array. The top and bottom edges may be left open or may be connected to the corresponding elements on the opposite edge, at the programmer's discretion. Four options are available for the left and right edges. Like the top and bottom edges, the left and right edges may be left open or connected to corresponding elements. The other two options are forms of spiral connection in which elements on the right side are connected to elements on the left side one row down. In the *open spiral* mode, the bottom right and the top left elements are left open. The *closed spiral* mode connects these two elements.[22]

In addition to the internal interconnections, the processing elements have two 128-bit parallel interfaces to the outside world. The input side of the system is on the left edge of the array with one bit connected to each processing element. The right hand edge forms the output port of the system. Global control of the array is achieved through a microprogrammed array control unit (ACU). A DEC PDP-11/34 is used as a back-end program and data management unit (PDMU). This system provides storage and communications management as well as interfacing to any external computer system.[23]

2.8. Hypercubes

The first few machines presented in this chapter made up several examples of shared memory multiprocessing. Next the ILLIAC IV and the MPP were examples of machines with many processing elements having local memory but with a single control unit. Now examples of a third approach to multiprocessing will be presented. In these systems, a number of independent CPUs, each with local memory, are connected with some communications network.

One of the most popular such systems of late is the hypercube. This architecture, also know as a binary-n cube, was originally developed at the California Institute of Technology. The construction of a hypercube is a recursive procedure. The 0^{th} order hypercube is a single processor with no connections to other processors. An n^{th} order hypercube is constructed from an $(n-1)^{th}$ order hypercube by replicating this $(n-1)^{th}$ order cube and connecting corresponding elements of the two cubes. By simple construction, one can see that an n^{th} order hypercube has 2^n processors each with n connections to other processors.[24]

While the hypercube is still a subject of study, it is known to have a number of desirable properties. Its current popularity stems from its applicability and its growth properties. This architecture has been shown to be well suited to many of the computationally demanding algorithms such as the Fast Fourier Transform (FFT). The hypercube is also a desirable architecture because of its extendibility. Since the number of connections increases as log(n), the size of the system can grow large before the connections dominate the amount of hardware in the system.[25]

Because of these desirable features, a number of commercial implementations of the hypercube have appeared in recent years. Intel has made a number of machines using their microprocessors. Another company, NCube, is producing machines using a custom VLSI processor. One of the most interesting incarnations of the hypercube architecture is the Connection Machine. This machine, produced by Thinking Machines, Inc., was originally designed as a Ph.D. dissertation by Danny Hillis who founded TMI. It is a 16^{th} order hypercube with 65,536 one-bit processors. Each processor has 4K-bits of memory and 16 links to other processors in the network.[26]

2.9. The RW-400

A most interesting older machine has recently come to light in this research. This machine called the RW-400 was introduced in 1960 and was called ‘‘a new polymorphic data system.’’ A Ramo-Williams RW-400 system was any one of a number of possible configurations of the various system modules. These modules included a Central Exchange (CX), Computer Modules (CM), Buffer Modules (BM), Magnetic Tape Modules (TM), Tape Adapters (TA), Magnetic Drum Modules (DM), Peripheral Buffer Modules (PB), and Display Buffer Modules (DB). The CX module had the capability of connecting up to 16 CMs or BMs and up to 64 of the other peripheral modules. In addition, the CX could be extended beyond these limits when needed. By program demand, the CX could connect any CM to any BM or peripheral module or any BM to a CM or peripheral module. No direct connection was possible between pairs of CMs or pairs of BMs.[27]

Each CM was a general purpose two-address CPU with 1K words of local memory. The buffer modules each had two banks of 1K words of storage which could be used as an extension to a computer module connected to it through the central exchange. The BMs also had a limited instruction set of their own and could be used as I/O processors for controlling the operation of the peripheral modules.[28]

It is clear that this multi-processor system had some capability for cooperative operation through using buffer modules as shared memories. Of course, the BMs had to be switched among cooperating processors, but this provided a built-in locking mechanism. It is also interesting to note the reconfigurable nature of the system. While no network of processors existed to be reconfigured, it is clear that the idea of a system with the capability of restructuring itself is not without precedent.

2.10. Summary

In this chapter a number of approaches to multiprocessing were discussed. By no means can the list of examples presented here be considered exhaustive. However, an attempt has been made to present some of the more interesting examples of several particular areas of multiprocessing research.

CHAPTER III

A Model of Multi-Processing Architectures

Science seeks generally
only the most useful systems of classification;
these it regards for the time being,
until more useful classifications are invented
as true.

– S. I. Hayakawa

In the previous chapter, a number of multi-processing systems and architectures were presented. In reviewing these one tends to focus on the details of the actual implementation. However, in order to avoid the possibility of "losing sight of the forest for the trees," it is time to take a step back and begin to look at the problem of multi-processing with new eyes.

3.1. Motivations for Multi-Processing

As a first step in re-examining multi-processing, one must consider the reasons that so much of computer research has been devoted to improving performance through parallelism. There are many motivations for using more than one processor, but only four of the most encompassing and powerful reasons will be discussed here.

3.1.1. Economic Considerations: One of the aspects of modern technology is that the cost of constructing a processor vs. its computing power rises with a slope greater than one. That is, a processor with twice the computing power costs more than twice as much to build. Therefore, if one can build a dual processor system that can operate with twice the computing power but only twice the cost of the single processor counterpart, it would be more economical than building a single processor system of twice the computing power.

The catch in this scenario is, of course, the pair of assumptions. In general, a dual processor system will not have twice the performance of one of the processors. Such a system will also generally cost more than twice as much as a single processor system. So, when "the bottom line" is the prime consideration, the designer must balance the added cost of inter-processor communications and sub-optimal performance increases against the increased cost of a more powerful processor. This is the foundation for attempting to keep the cost of the inter-processor communications low in relationship to the cost of the CPUs.

3.1.2. Fault Tolerance: The classic means of detecting and tolerating faults is redundancy. Using more than one processor to perform the same operations allows the system to determine when a fault occurs by comparing the results. The use of this technique in the BINAC computer was mentioned in the previous chapter. This principle can also be extended to allow for fault tolerance. If more than two processors are used to concurrently perform an operation, then a voting mechanism can be used to select which processor is faulty. (The complexities of such a system were amply demonstrated at the launch of the first space shuttle flight.)

In a typical single processor environment, if the processor fails, the system is useless. Conversely, if one of the processors in a multi-processor system goes down, then the system will lose functionality and computing power, but the system will still be usable at a lower level of capability.

While this is an important field of study, it is not a mechanism for increased performance. As a result, this motivation for multi-processing will not be considered further here.

3.1.3. Maximum Absolute Computing Power: For any given technology, there is a limit on the maximum speed of a single processor system. There will always be applications which call for more computing power than the fastest single processor systems can offer. Weather forecasting and image processing are good examples of these power hungry applications. In order to address these applications, one can always make use of more than one of the most powerful single processor computers to increase the total computing power. Clearly, this was the motivation behind the development of the CRAY X-MP. By making use of parallelism in the algorithms to be processed, a multiprocessor system can solve the problems faster.

3.1.4. The academic point of view: It can be argued that, despite whether or not multiple processor systems are advantageous in practice, it is important to study them in order to gain a greater understanding of designing and programming computer systems. By building and studying systems with more than one processor, a great deal of insight into many aspects of computer science can be obtained. Clearly, the understanding of communications in any collection of computing systems can be enhanced. A number of the concepts of parallelism can and have been transferred into the world of single processor design. On the software side, the process of implementing programs for multiprocessor systems brings about re-examination of the algorithms that, in turn, builds new understanding into their nature. Also, the influence of multi-processing on compilers and operating systems has and will continue to contribute to the general improvement of the computing environment.

3.2. A proposed Model

With the objectives of multi-processing in mind, a general model of multiple processor systems can now be described. Any model for multi-processing must encompass a very wide range of approaches. Such a requirement tends to influence either an extremely simple or an extremely complex model. Rather than attempt to construct a model with enough complexity to describe in detail all possible approaches, a simple model that can accommodate any system will be described.

The motivation behind this chapter has been to approach multi-processing as if the field were being created for the first time. The model developed here will be in keeping with this approach and will be based on the objectives described above. In creating a system that increases performance through the use of more than one CPU, one must consider the general form of the resources that are to be used. It is clear here that the resources are a pool of processors which must be connected in some manner that will provide for cooperation in executing programs. In addition to CPUs, many memories and I/O devices are available to use in the operation.

A model of multi-processing can then be described as a collection of processors, memories and I/O devices, all with one or more connections to a communications network. This model is pictured in Figure 3.1.

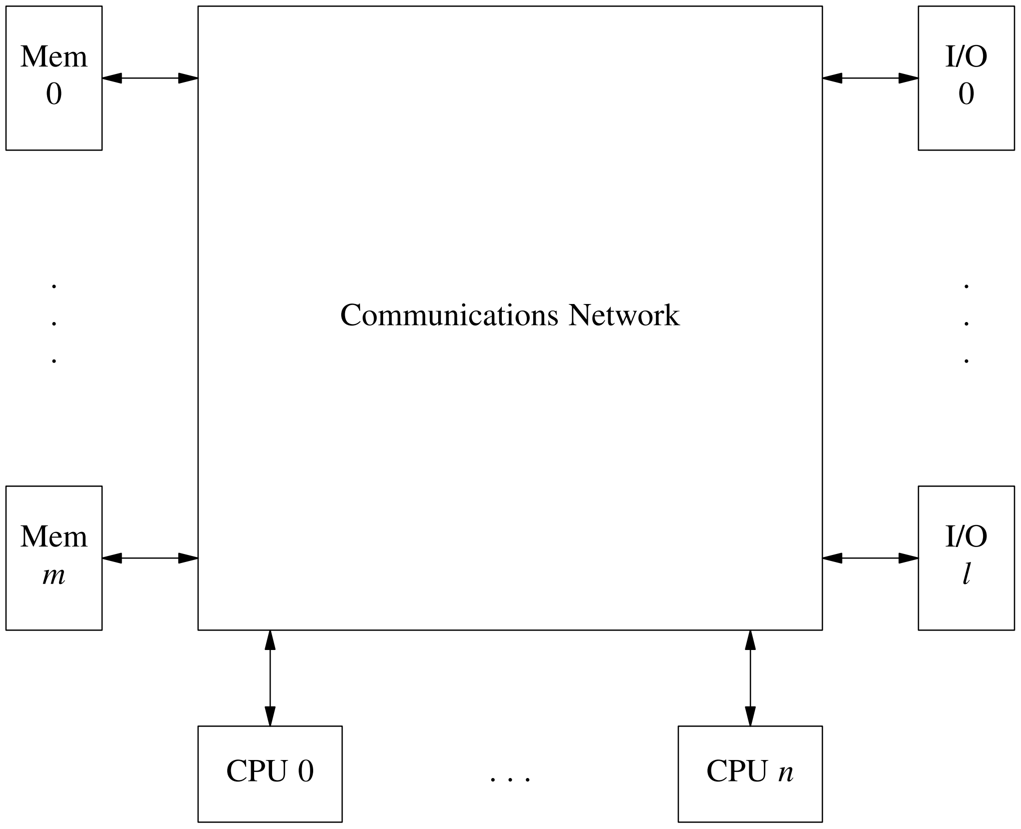

Figure 3.1

A Model of Multi-Processing Architectures

It is clear that the "meat" of any architecture lies in the structure of this communications network. A number of desirable features of this network will be considered later, but at this time it is appropriate to return to the present state of multi-processor development and to consider how the machines described in the previous chapter fit into this model. All of the systems discussed in the previous chapter can be placed into this model by describing a suitable communications network. Perhaps the simplest fit of those machines is the C.mmp system. In this machine, the connection between processors and memories is the cross-point switch. The connections to I/O devices are simple, direct connections between them and their host processors.

All of the commercial dual processor systems can be described in this model as having a simple switch selecting the processor which may access the memory. With the larger systems that use memory interleaving schemes, this description is not sufficient. Here it is better to show multiple memories or one memory with multiple ports, each of which has a switch to the processors. It should also be noted that most of these systems use I/O processors as well as multiple general purpose CPUs. From a memory access point of view, these processors can be modeled simply as more of the system CPUs, each contending for memory accesses.

With machines such as the hypercube, the ILLIAC IV and the MPP, the fit of the machines in the model is likewise straightforward. Since each of the processors or processing elements in these systems operates with local memory, the ports of the memories are directly connected to the corresponding processors. Each of the processors also has one or more connections to other processors.

From the viewpoint of the model being discussed, a comparatively complex machine is Cm*. The communications network of Cm* can be represented as a message-passing system between the processors and the memories. This system does not have any form of direct connection between processors, but, like the shared memory systems, must communicate through multiple ports to the memories.

3.3. Classification of Architectures

In the previous section, a model was put in place to describe the diverse approaches to achieving the objectives of multi-processing. It is now time to consider means of classifying individual architectures and implementations. The remainder of this chapter consists of a presentation of several such systems. The first few are some of the many methods discussed in the literature. Finally, a new classification mechanism is developed in the light of this model as an extension of an old classification.

3.3.1. Flynn's classification: Probably the most well-known of the computer classification schemes is due to Flynn.[29]

His classification divides the space of possible computer processor systems into four quadrants. These are as follows:

1) SISD – Single Instruction; Single Data
2) SIMD – Single Instruction; Multiple Data
3) MISD – Multiple Instruction; Single Data
4) MIMD – Multiple Instruction; Multiple Data

The SISD description applies to most general purpose computer systems on the market. In these systems there exists a single instruction execution unit which controls manipulations on a single stream of data.

If the single instruction execution unit has control over multiple data manipulations in a single instruction, then the processor can be classified as a SIMD system. This is true of most vector processor systems. In these systems, single instructions specify operations on groups of data elements. SIMD is also a description of the ILLI-

AC IV and the MPP discussed in the previous chapter. These systems have one instruction execution unit for groups of 64 and 16,384 processing elements respectively.[30]

No realizations exist for the third class (MISD) of processors. Conceptually, these machines have multiple instruction streams controlling operations on a single data stream.[31]

The final classification, MIMD, represents computers where more than one instruction execution unit controls more than one data stream.[32]

If there is one data stream for each instruction stream, then the system is a collection of conventional SISD CPUs. All of the shared memory machines including Cm* and C.mmp, discussed in Chapter 2, are in this class of machines. Also, most hypercubes fit into this classification. It is this group of computer systems that is of most interest here.

3.3.2. Classification by Parallelism: A number of researchers have chosen to classify systems according to a degree of parallelism. Among these are Feng, Händler and Shore. Feng defined a doublet, which specified the basic word length and the *bit slice* length of the system. His bit slice is not the same as the building block of processor words found in bit slice processor design. Instead, Feng's bit slice is indicative of the number of words being operated on at one time. A machine described by an (n, m) pair operates on m words, each n bits wide at a time. The early, general purpose microprocessors could all be described by the pair (8,1). Likewise, the ILLIAC IV, described in the previous chapter, is classified by (64,64) for the single quadrant implemented. Also, the Connection Machine is described by (1,65536), and the DECsystem 10 Model

KL-10 has a classification of (36,2).[33]

In Feng's classification, pipelining of processing elements is treated as parallel executions. Händler, however, makes a distinction. Furthermore, he provides separate descriptors for processors which execute instructions and processing elements such as ALUs. His classification system is based on a triplet of the following form:

$< K \times K', D \times D', W \times W' >$

where:

K = the number of processors in the system

K′ = the maximum length of a processor pipeline

D = the number of processing elements in the system

D′ = the maximum length of a processing element pipeline

W = the word width of the processors or processing elements

W′ = the number of pipeline stages within a processing element

Using this notational classification, the ILLIAC IV is described by <1,64,64>, the MPP by <1,16384,1> and the Cray 1 by $<1,12\times 8,64\times(1\sim 14)>$.[34]

Like Flynn, Shore divides the space of possible computer systems into several groups. A type I machine is the conventional bit parallel, word serial machine. Type II machines operate in a bit serial, word parallel mode. Here, processing elements operate in a bit serial fashion, but data is fetched from memory by getting appropriate bits from many words in parallel. The type III machine is the combination of both type I and II machines. A type IV machine is an array of processing elements connected to memory but with no interconnections between the elements. In the type V machine, the array of processors have communication channels among them. Finally, the type VI machine is described as a logic-in-memory machine, where no distinction is made between processing logic and data storage. In general, these can be viewed as subdivisions of Flynn's SISD and SIMD classifications.[35]

3.3.3. The Hockney, Jesshope Classification: Like Feng and Händler, Hockney and Jesshope present a method of description as well as of classification.[36]

This fairly elaborate system provides a good, detailed description of the processors, the memories and the interconnections. Like Shore, Hockney and Jesshope present classifications refining the SISD and SIMD classes of Flynn.

Without delving into too much detail about their notation, it is useful to look at at least one example of it. The Hockney-Jesshope notation for the ILLIAC IV is:

$$C(\text{ILLIAC IV(4 quadrant)}) = C1[4C2^{80}[64\bar{P}]_l^{1-nn}];\ P = F_{64}^{600} - M_{2K*64}^{400}$$

This chemical-like formula indicates that the ILLIAC IV has a control computer, $C1$ which controls four other control computers, $C2$. Each of the $C2$s has an 80ns cycle time and controls 64 identical processing elements P. The $1-nn$ superscript on the processing elements indicates that each element has a direct connection to its nearest neighbors, and the l subscript indicates that the processing elements operate in a lock-step fashion. Finally, the processing elements are each made up of a 64-bit floating-point execution unit with a 600ns cycle time and a 2048 word memory with a 400ns access time. In addition to the components that make up the systems, the Hockney and Jesshope notation is capable of describing the structural relationships between the processors in the system.

In classifying computer systems, Hockney and Jesshope first divide the space into machines with a single instruction execution unit and those with multiple instruction execution units.[37]

Only the single instruction stream machines (SISD and SIMD according to Flynn) are considered further. This group of machines is divided into two classes: machines with a single unpipelined data execution unit and those with either pipelined or multiple execution units. These classifications closely parallels Flynn's SISD and SIMD machines. The class of single unpipelined data execution unit machines is further divided according to integer vs. floating-point operation and according to bit-serial vs. bit-parallel operation. The pipelined and multiple data execution unit machines are also divided by the structure of control mechanisms and type of data used.

3.3.4. A Connectivity-Based Classification Method: The concept of a set of processors or processing elements in a computer being *tightly* or *loosely* coupled has been frequently used to differentiate between large groups of architectures. This section, however, proposes to apply a metric to the idea of coupling to indicate, in some sense, the strength of the connectivity.

If one looks at the multi-processing model discussed above, it is clear that the ability of processors to cooperate is directly related to the amount of information that they can transfer among themselves. By viewing the communications network as a multiport input/output system, a measure of this cooperation potential can be created from the various data transfer rates at the ports.

Before developing the measure of coupling, the actual data transfer rates must be isolated. Consider the total bi-directional data flow for all processors. This aggregate data rate D_T is made up of several components as follows:

$$D_T = D_{DC} + D_{LM} + D_{SM} + D_{IO}$$

where:

D_{DC} =	the aggregate data rate of bi-directional communication directly between processors
D_{LM} =	the aggregate data rate of communications between processors and the memory local to each of them
D_{SM} =	the aggregate data rate of communications between processors and shared memory
D_{IO} =	the aggregate data rate of communications between processors and external I/O devices.

Of these elements, only the D_{DC} and D_{SM} portions can be used for cooperations among processors. The local memory bandwidth and the external I/O bandwidth provide no mechanism for cooperative processing. Therefore, the connectivity data rate D_C is that portion of D_T thus:

$$D_C = D_{DC} + D_{SM} = D_T - D_{LM} - D_{IO}$$

The next step is to describe some of the characteristics that this metric should have. In general, this measure is to assign a degree of coupling among the various processors in a system, not the overall performance of a particular implementation. Consequently, it should be independent of the number of processors in the system and the computing power of the individual processors. Since the aggregate data rate is proportional to the number of processors and approximately proportional to the instruction execution rate, these two parameters must be divided out of the data rate. The degree of coupling is then defined as:

$$C = \frac{D_C}{E\,N}$$

where:

E = instruction execution rate

N = number of processors or processing elements

In addition to the properties mentioned above, this measurement of coupling has some other interesting properties. First, the degree of coupling is measured in words per instruction-processor. It is then a per-processor measure of how many words may be transferred between processors per instruction cycle. It is also interesting to note that any number of autonomous computers have a zero degree of coupling. Since memory accesses take place at approximately the same rate that instructions are executed, shared memory systems, such as C.mmp, the IBM or DEC or Cray systems discussed in the previous chapter, all have a C of roughly unity. For a machine such as Cm*, however, the degree of coupling must be less than one. Even though the inter-processor communications take place through shared memory, the mechanisms of accessing remote memory involve several stages of message passing and translation. Therefore, access to shared memory takes place at a slower rate than average instruction execution. It should be clear that no system can have a connectivity greater than one, unless it has special hardware that can transfer more than one datum per instruction cycle. It is, however, questionable as to how significant it is to be able to transfer information to/from a processor faster than it can use/create it.

3.3.5. The Connectivity Number Line: With the connectivity measure defined above, one can imagine a "number line" running from zero through one and beyond wherein each multi-processing system occupies one point on that line. It has already been pointed out that a collection of completely autonomous computers has a connectivity of zero. A dial-up network of systems then has a small but greater than zero degree of coupling. The next step is the various local area networks (LANs), where a greater degree of communications is possible but still small in relation to the instruction execution rates. Figure 3.2 shows this "number line" with several machines identified on it. No attempt has been made to calculate the exact number corresponding to each of these systems. Instead, their relative placement is based on the qualitative arguments above.

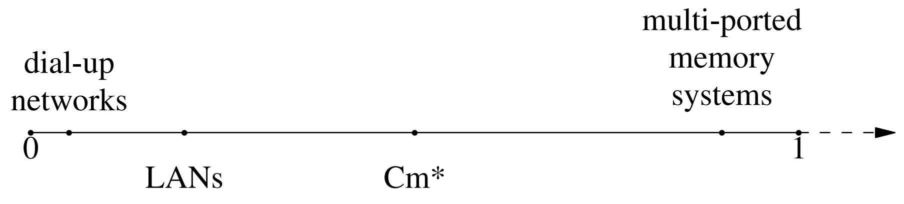

Figure 3.2

A "Number Line" of Connectivity

CHAPTER IV

A General Purpose Architecture

A doctor can bury his mistakes
but an architect can only
advise his clients to plant vines.

– Frank Lloyd Wright

This chapter will propose a new multi-processing architecture. The first portion will present the motivations behind the architecture. The next two sections give the details of the proposed new architecture and establish its relationship to local area networks (LANs). Final sections of the chapter present an evaluation of the results and conclusions of the project.

4.1. Motivations and Objectives

Before creating any new architecture, it is beneficial to describe some of the desirable features for such architectures. Since one of the motivations for multi-processing is to increase performance, it is desirable that a significant performance enhancement should result from the architecture. It is also desirable for an architecture to be extendible so that the performance increase is not restricted to a small number of processors. As with most computer systems, this architecture should be applicable to a wide range of applications and computing environments. Finally, it is useful to design the architecture so that existing general purpose computers can be used as component processors. Such simplicity of design is beneficial in several respects. It would allow a manufacturer to include both single and multiple processor computers in the same family. Another result of such a characteristic is the possibility of converting LANs into highly coupled multiprocessors. These four attributes form the basis of the motivations and objectives of the new architecture.

Clearly, in the ideal case of multi-processing, performance of a system increases linearly with the number of processors it contains. Unfortunately, this ideal case is never realized in actual architectures. One of the primary reasons for this is the overhead in inter-processor communications. This overhead takes two forms. First, processor time is used in performing the data transfer to and from other processors. In general, little can be said at an organizational level about how to avoid this. However, implementational details should be aimed at making communications as simple as possible. Secondly, if a processor cannot get or send information as fast as it can use or generate it, then performance will suffer. This implies that an architecture should have a degree of coupling (as defined in the previous chapter) close to one.

Another reason for sub-optimal performance increases comes from the inability to easily map algorithms onto multi-processor implementations. If algorithms cannot be mapped onto an architecture in such a way that all of the processors are used all of the time, then the idle processors will be wasted and the performance enhancement will be limited. A great deal of research has been done to determine both how algorithms can be mapped onto specific architectures and which architectures are best suited to specific algorithms. Most approaches to multi-processing attempt to create an architecture that is well suited to a large group of algorithms. However, in this chapter, an architecture that is reconfigurable to suit the algorithm at hand will be presented. Another motivation for a reconfigurable machine comes from a multi-programming software environment. The ability for the interconnections among processors to be changed on demand gives an operating system the freedom to allocate multiple processors to more than one simultaneously running process.

4.2. A General Purpose Architecture

4.2.1. Overview: At this point, it is appropriate to begin presenting the details of the architecture. These features will be described in light of the model of multi-processing presented in the previous chapter. As discussed in Chapter Three, this architecture is a methodology for connecting many processors, memories and I/O devices in order to provide cooperative processing with them. Several such methodologies were considered in Chapter Two. Some of these were based on shared memory while others were based on communications networks.

The architecture here is based on a communications network rather than shared memory. This choice was made in order to achieve the goal of extendibility mentioned above. Whenever two or more processors attempt to access the same area of shared memory, there will be some delay introduced due to the contention. Since this degradation increases with the number of processors accessing the memory, globally shared memory is not appropriate for systems with a large number of processors. By placing only the cooperative information in the shared memory spaces, delays are made to occur only when multiple processors access cooperative information. Thus it is possible to decrease these undesirable effects through the use of both local and shared memory, but this approach is still subject to increased degradation with larger numbers of processors.

The structure of the I/O subsystem has been chosen similarly. Rather than give each processor a networked access to all actual devices, devices are attached to selected processors that may act as I/O processors. Typically, there will only be a few of these processors and they will be used as I/O servers to all of the other processors. However, there is no restriction that prevents these processors from running other code.

Since the memories and I/O devices are directly connected to corresponding processors, the communications network in this architecture is an inter-processor communications network (IPCN). Each processor module is connected to the IPCN through some form of communication port or ports. This can be illustrated as in Figure 4.1. The processors in the figure depict CPUs with their local memory and communication ports.

As mentioned above, this communications network is reconfigurable. This reconfigurability permits connections to be established between any pair of processors in the system. These connections can then be used to transfer cooperative information between the processors. When the connection is no longer needed or the processors must be connected to different processors, the network can be reorganized with different connections. The connections in this system correspond with the logical session connections found in computer networks. This parallel will be discussed in more detail in Section 4.3.

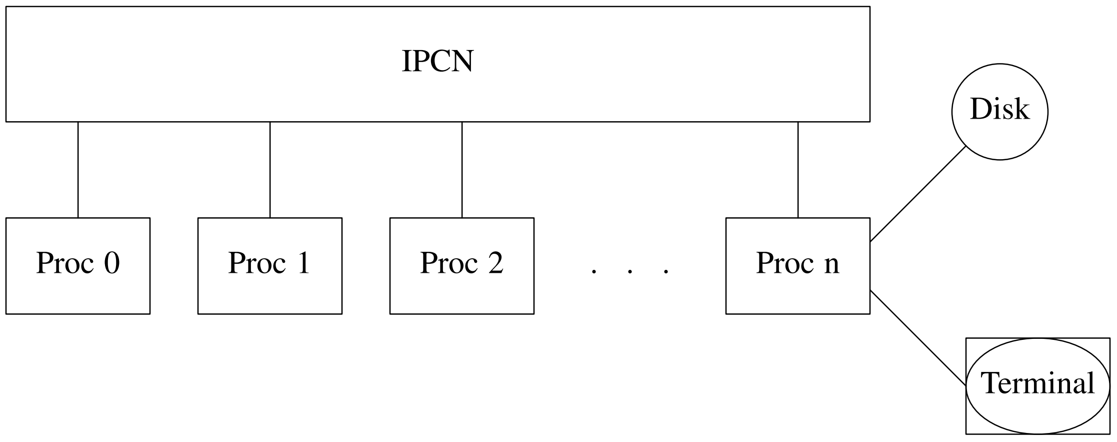

Figure 4.1

General Purpose Architecture

4.2.2. The Processor Modules: As illustrated in Figure 4.1, this new architecture is an interconnection of many processor modules. These modules consist of a CPU, its local memory, one or more communications ports into the IPCN and optional external I/O controllers and devices. This structure is illustrated in Figure 4.2.

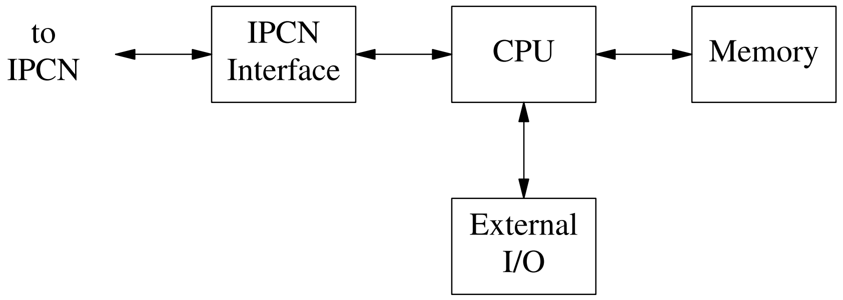

Figure 4.2

Processor Modules

Aside from the IPCN ports each module is a conventional von Neumann computer system. All of the characteristics which are desirable in ordinary single processor systems are also desirable for the processor modules in this architecture. Since there is no shared memory, caches can be used without special purpose hardware for maintaining consistency among the multiple processors. As in conventional single processor computers, it is an implementation decision as to whether or not the individual processor modules are to support multiple processes. Likewise, if they are to support multiple processes, then some form of memory management must exist in the module.

Another matter related to memory should be addressed now. That is the issue of process residency and mobility. If the processor modules can host more than one process, then it will sometimes be useful for the processes to move from one processor to another. In a shared memory multi-processor system, all processes reside in the common memory space. Thus each processor may "pick up" and run any process not currently being run by another processor. However, because this system has only local memory for each processor, in order for processes to migrate between processors, they must be copied from the memory of one module to that of the other. Since this method of migration is somewhat inefficient, processes should generally stay on the same processor that they begin execution.

In this architecture, the CPUs in the component processor modules need not be identical. Indeed, it is advantageous for the system to be composed of heterogeneous processors which allow processes to be divided up according to the processors most suitable for the task. For instance, the processors attached to the I/O devices should be selected to be those most suited to I/O control. There may also be processors in the system that are designed for mathematically intensive applications, such as digital signal processors or there may be those that have very good string handling facilities. These different processors may be allocated to the portions of an application that require the respective capabilities.

The I/O devices in this system are directly connected to single, associated controlling processors. Design of the controllers for I/O devices follows the same motivations that appear in their single processor system counterparts. To the rest of the system, these processor/device subsystems appear much like the IBM channels or the Cray IOS. The system works most efficiently when these processors execute the I/O portions of the operating system. Other processors in the system can then directly route I/O requests through the IPCN. Likewise, the actual data transfers associated with I/O requests also take place through the IPCN.

4.2.3. The IPCN: The general structure of the IPCN is divided into two parts, the network controller and the reconfigurable communications network. The function of the network controller is to take configuration requests from the processor modules and control the configuration of the communication network. Another characteristic of the network controller is that it performs a process to processor translation for connection requests. In the same way that a telephone switching office translates a phone number into a physical line, the network controller will take requests based on the destination process. Then using internal tables updated by the operating system, the controller will make the connection to the appropriate processor. This characteristic has several advantages. First, it allows processes to be allocated among the various processors without necessitating that the cooperating processes be informed of the actual processor modules involved. It further provides a means for tagging messages with a destination process if the processor modules support multiple processes. Without such a system, there would be no way to distinguish which connections are established for which processes.

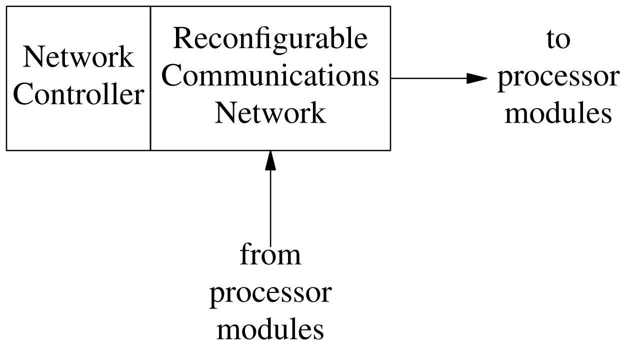

Figure 4.3

Inter-Processor Communications Network

The speed with which the network controller effects the changes in system configuration will be implementation dependent. It is desirable, though, that the configuration time be as short as possible to avoid delays in process execution. By making configuration changes fast relative the the lifetime of a process, changes to system topology during process execution become feasible. It also makes the overhead in setting the system up during process creation less significant. For most systems, servicing connection requests in less than 100μsec should be sufficient.

Clearly, the heart of this new architecture is the reconfigurable communications network. This portion of the IPCN is the switching center that connects the processor modules in various topologies. When the network controller processes a connection request, the communications network must provide a direct communications path between the source and destination processors.

Since the objective of this network is to provide the most flexible set of configuration possibilities, it is desirable for all processors to be connectable to all others. The cross-point switch presented in Figure 1.6 is suitable for this purpose. Another advantage to using such a mechanism is that delay through a cross-point switch is much shorter than the instruction execution time of a typical micro or mini-computer. One characteristic of this switch is that its complexity grows as n^2 with the number of IPCN ports, n. This characteristic bodes ill for the extendibility of the system. In order to obtain both flexibility and extendibility, it is wise to take a lesson from the development of the telephone network. A hierarchical group of cross-points can be powerful and economical. A two level hierarchy is illustrated in Figure 4.4. This system is not dissimilar to the hierarchical bus structure of Cm* discussed in Chapter Two. Another approach to limiting complexity is to run several cross-points in parallel. If more than one IPCN port is attached to each processor, then each port can be connected to a different cross-point network. A full cross-point interconnection would still grow as the square of the number of processors, but only linearly with the number of ports on each processor.

The switching network above provides hardware connections between processor modules, and the connections are owned exclusively by those modules. Another approach to providing the desired flexibility is to provide logical (or virtual) connections between processors. This can be done with a bus that has access to the memory spaces of all of the processors. Information can be transferred over the bus from one processor module memory to another. Since this approach shares the bus resource among all si-

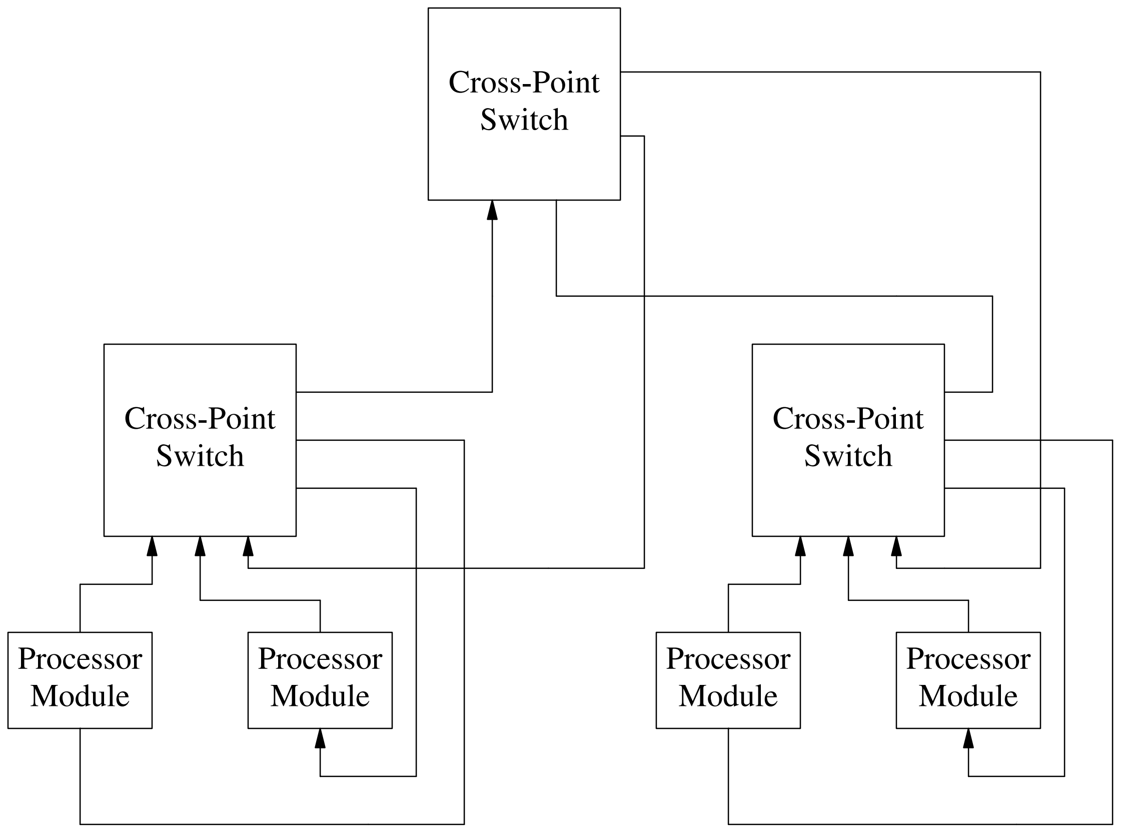

Figure 4.4

IPCN Implemented With
A Tree of Cross-Point Switches

multaneous transfers, contention arises as in the shared memory cases discussed earlier. This contention, in turn, causes delays and limits the extendibility of the system. Although a logical connection approach has its disadvantages, it can be used in parallel with a switching network to provide for efficient bulk I/O transfers.

4.2.4. Interface Between Processor Modules and the IPCN: One topic not yet discussed is the characteristics of the communications ports used to connect the processor modules to the IPCN. There is a great deal of flexibility in the implementational details of these interfaces. However, careful consideration of the objectives of the system leads to some characteristics that they should have. First the hardware complexity should be considered. From an economic standpoint, it is desirable that the hardware be as simple as possible. For the IPCN, data switched in serial rather than parallel streams would be less complex. This must be balanced against the added complexity of serial interfaces over parallel interfaces on the processor modules.

The next objective to be considered is the performance of the communications subsystem. As mentioned earlier this system should have a low overhead and a degree of coupling close to unity. Both of these objectives imply that data should be capable of being transferred to and from the IPCN communication ports on the processor modules in a single instruction cycle. The use of DMA on the IPCN ports easily meets this requirement. A less complex, though less efficient, approach is to use a memory mapped register for the IPCN ports.

If parallel data transfers are used, then transmission delay will be much shorter than the instruction execution time. Therefore, the predominate time for data transfer will be the instruction execution time for loading and reading the communication ports. Conversely, if serial ports are used, then the bit rate must be sufficient for a word to be transferred in the time it takes for the next instruction to be executed. This allows data to be transferred at the same rate that instructions are executed to effect this transfer. This bit rate must be n times the instruction execution rate for n bit transfers.

Another aspect of the inter-processor communications that must be designed for high performance is the control of data flow between processors. This design task closely parallels the design requirements of communications I/O sub-systems. If both ends of the transfer have DMA controllers that operate at the full speed of the network, then no flow control need be present. If a more economical software approach is taken, then some form of handshaking must be designed. Like the ports into the IPCN, this handshaking mechanism must be designed so as to impose as little software overhead as possible. An interrupt based system represents a good approach.

4.2.5. The Network Configuration Mechanisms: The final aspect of the processor module to IPCN interface is the mechanism for configuring the network. As previously indicated, one or more processor modules can provide configuration requests to the network controller which translates the requests into control signals for the network.

The mechanism chosen for implementing configuration requests is closely tied to the process control mechanisms. If a single processor is chosen to make the configuration decisions, then there must exist some mechanism of specifying what topology each program runs on. This can be accommodated by including a coding for the topology in each program header or attributes. When a program is started by the operating system the network is appropriately configured for the program. This approach makes most sense when the operating system performs the task of creating all of the sub-processes on the appropriate processors at the program initialization. Such a topology control mechanism tends to imply that the set of connections available to a program are static throughout its life. If a program, for which this limitation is not acceptable, is to be run, then a path of communications between the process and the operating system should be opened. In such a structure, the process can request configuration changes of the operating system which passes the requests to the network controller.

Alternatively, if the UNIX process creation and inter-process communications mechanisms are used for a model, a program will always be started as a single process. This process then becomes the parent process to all sub-processes created as needed. In this approach, communication channels between processes can only be created by a common ancestor of the processes prior to their creation. Therefore, the operating system will perform all configuration operations at the time it creates processes as in the approach above.

Another, more flexible, approach to network configuration is to allow processes on any processor modules to request configuration changes. Using this approach, processes request a unidirectional connection between one of the output ports on its processor and an input port of the processor hosting the destination process. When all of the data has been transferred or when another connection is needed, the process requests that the connection be torn down. Adding the hardware necessary to support this capability will clearly increase the cost and complexity of the system. However, this approach adds a feature not easily obtained with the other methods. That is the ability to simulate the existence of more ports than physically exist by switching the existing ones among the various destinations.

When the implementation allows processors to host multiple processes on a timesharing basis, another connection to process management is made. A program's sub-processes communicate over switched hardware. Therefore, these sub-processes must execute simultaneously in order to pass cooperative information among themselves. This implies that there must be central scheduling and control of context and process switching throughout the system. It also implies that in addition to the usual information that must be saved and restored in process switch, the network topology must also be saved for each process switch.

The protocol for making configuration requests is implementation dependent. If a single processor is responsible for making these requests, then a simple channel of communications between that processor and the network controller is sufficient to pass the requests and the acknowledgements. Otherwise, if all processors are able to make requests, then a more complex interfacing mechanism is necessary. Many such interfaces are possible, but perhaps one of the most extendible is patterned after the DEC UNIBUS. A global request bus is provided that has several data lines and two control lines. One of the control lines is for making requests and the other is for acknowledgement. The acknowledgement control line is daisy-chained along the processors in such a way that the signal is passed from a processor module only if it is not currently making a request. Normally the data lines and the request line are floating inactive until a processor makes a configuration request. To make a request a processor module makes the request line active, alerting the network controller to the request. The controller then asserts the acknowledge line which is passed down the bus until it is received by the first requesting processor module. Upon detecting the inactive to active transition, the requesting processor will place the request information on the data lines. After the network controller processes the information, it deactivates the acknowledge line causing the requesting processor to release the bus. If the controller does not detect an inactive state on the request line within some time limit, then another processor down the line must also be making a configuration request and the cycle starts again.

4.3. The Multi-Processing System as a LAN

The structure shown in Figure 4.1 clearly bears a great deal of resemblance to the overall design of local area networks (LANs). It can also be used to describe any other architecture which does not use shared memory. This dual nature of the description leads to a fundamental principle that the new architecture is based on. Namely, the general characteristics of a LAN can be used to effect cooperative processing by multiple processors.

A primary difference between most LANs and the communications based multiprocessing systems discussed earlier is the performance of the communications networks. The design motivations of typical LANs do not require as high a data throughput as do multi-processing systems. This and the economic and geographic considerations lead to lower total bandwidth communications than the aggregate bandwidth of the reconfigurable architecture. Such systems, therefore, have a low degree of coupling as defined in Chapter Three. This difference in capability of the communications networks leads to significantly different software support of multiple processors. In a multi-processor system with a higher performance communications network, the application can be divided into more parallel executing sub-processes that all share cooperative information. Conversely, if the communications facility can be saturated by the cooperative information, then further division of the application is of no avail.

If the performance of the LAN communications network is increased to a degree of coupling close to one, then the network can be used for highly cooperative multi-processing. In a generalized sense, this situation exists for all communications based multiprocessor systems. The set of processing element interconnections for machines such as the ILLIAC IV, the MPP and the hypercube can be considered high end implementations of LANs. In these systems, though, the communications are not as flexible as in a typical LAN. The combination of the flexibility of the communications in a LAN and the high performance of static multi-processing networks is a cornerstone of the design of the new architecture.

The architecture described here provides flexibility approaching the LAN. However, since a LAN routes data on a per message basis, the set of possible destinations for a message is not restricted to the set of connections that exist. The new architecture requires that connections be allocated and released to achieve arbitrary routing characteristics.

Another difference between LAN communications and that of more tightly coupled networks is the reliability of the transmissions. Since LANs are designed to operate over large distances compared to the size of component computers, they must contend with the non-zero probability that a transmission will fail to be received. When designing a communications based multi-processor architecture such as those in Chapter Two, the communications can reasonably be assumed to be reliable. This difference is a crucial one. If no process can ever be certain of what another has received, then the two processes can never synchronize their operations on common data structures.

To illustrate this inability, consider a process attempting to notify another that it is locking an element of a shared structure. For simplicity, let there be a copy of the structure on each cooperating processor. Cooperation can be enforced by exchanging locks before changes are made. Now since the locking process cannot make a change without notifying the other process and since it cannot know when or if the other process receives the locking message, it must wait for a response. On a LAN, if the response is never received, then the locking process cannot know whether the original message or the response has been lost. As a result it cannot know in what state the other process believes the lock to be. Conversely a tightly coupled communications network can assume that no messages are lost unless a major failure has occurred (taken to be very rare) since the communications facility of the new architecture is over geographically short distances and designed accordingly. Therefore, it can be used for lock based synchronization mechanisms. A response must still be returned for such a system, but this is to determine **when** not **if** the message was received.

It is clear that the new architecture and the LAN are topologically equivalent. Namely, any processor can send information to any other processor directly. As discussed in this section, however, there are performance and reliability differences that significantly affect the range of applications for which each is suited. This line which divides this new architecture and the class of systems known as LANs can become blurred. It is possible for LAN implementations to be realized that utilize several high bandwidth communications avenues of high reliability. These characteristics would endow the LAN with those necessary to realize the reconfigurable network architecture.

4.4. Summary

In previous sections of this chapter, several objectives for this new architecture were set forth. It is appropriate now to review the objectives and try to determine how well the general architecture and the specific implementation meet them.

4.4.1. Enhanced Performance: Any time a multi-processing system is assembled, it is done to create a system of higher performance than a single one of the component processors. Both the example architecture and the implementation discussed in the next chapter seem to achieve this. Most processes can be broken down into cooperative sub-processes. Each of these sub-processes can run on a different processor resulting in parallel execution of the application. Whenever this can be done, a system such as this will provide increased performance.

4.4.2. Extendibility: It is also desirable that the system be extendible to large numbers of processors. The IPCN described here is designed with this characteristic in mind. A large number of processors can be accommodated while still maintaining a flexible and high bandwidth network through the use of hierarchical cross-point switches. The example implementation is also somewhat expandable.

4.4.3. Wide Applicability: Since the communications network is reconfigurable, many different network topologies can be implemented upon demand. If a program is to be run that is best suited to an ILLIAC style array, then the network can be configured accordingly when the program is started. Likewise, if another program is to be run which is ill suited to an array but runs well in a hypercube, then the network can be reconfigured to accommodate that program. In general, any topology or network architecture can be implemented if two conditions are met. First there must be enough ports on each processor module to support the topology. For a hypercube, there must be $\log_2(n)$ ports for an n processor cube. Likewise, simulating the ILLIAC IV topology requires four ports per processor module. The other condition is that there be enough connections between branches of the switch hierarchy. If a full cross-point has been implemented, then this condition is always met.

4.4.4. Use of General Purpose Computer Systems as Component Processors: As will be discussed in more detail in the next chapter, the example implementation made use of an existing DEC LSI-11 micro-computer. This system was incorporated into the example implementation with the addition of an interface board that plugged into the computer's existing bus. No other changes to the system were necessary. This demonstrates that the architecture is well suited to the use of existing general purpose computer designs as component processor modules.

4.4.5. Conclusions: In this chapter, a new organization for multiple processor computers has been presented. It has been designed to meet the objectives of high performance, extendibility and wide applicability. These objectives are achieved by the use of a reconfigurable network of interconnections between processor modules.

This network gives the system flexibility and simplicity like a LAN and high performance closer to shared memory systems. In Section 4.3, the relationship between this new architecture and current LAN organizations was discussed. From this discussion, it became evident that the reconfigurable organization provides a higher performance communications capability than the LAN. It was also brought out that the new architecture has an advantage through more reliable communications. These two factors make this reconfigurable architecture amenable to a wider range of applications. Conversely, shared memory systems are capable of higher degree of coupling than this new organization for small numbers of processors, but memory contention delays limit the number of processor that can effectively be used in such systems. This new architecture provides a more extendible organization for multiple processors. Also since it is capable of supporting the same locking mechanisms as shared memory systems, it is usable for most of the applications appropriate for shared memory. This balance of the flexibility of a LAN with performance closer to a shared memory system makes this reconfigurable architecture a good balance between these two organizations.

There also exist a number of other communications based multiple processor architectures that can demonstrate these desirable features in one form or another. However, these organizations generally have a fixed topology of processor interconnections. This makes each of them well suited to some applications, but ill suited to others. As discussed in this chapter, this new organization is capable of assuming the topology of many of these other architectures upon demand. This feature gives it the ability to emulate these other architectures. This direct emulation capability is present only for those architectures which require the same number or fewer processor-IPCN ports than are available in any given implementation. If this requirement is not met, then the system can make use of its reconfigurable nature to switch ports among several possible connections and therefore simulate other topologies. This reconfigurability, then, makes the new architecture more flexible and therefore applicable to more algorithms than its fixed topology counterparts.

The ability to directly emulate and to simulate various interconnection topologies gives this new architecture an added characteristic. This is that algorithms created for these topologies can easily be ported to this architecture. In general, techniques developed for a given topology can be directly applied to this reconfigurable organization when it is configured for that topology. This is also true of the LAN logical topology.

It has already been mentioned that the LAN topology is logically equivalent to the new architecture. For this reason, the distributed processing techniques that have been developed and continue to be developed can be applied to this system.

Above, three results of the characteristics of the new architecture were presented. First the organization was described as being a good middle ground between LANs and shared memory multi-processors. The second result is that it is more flexible than the other high performance communications based multi-processor systems. Finally, it was shown how existing software techniques could be applied to this architecture. In addition to these, the example implementation in the next chapter demonstrated that this reconfigurable architecture can be realized easily with current, proven technology and techniques. It, furthermore, demonstrated that the system is also a good middle ground in the complexity and therefore cost between LANs and shared memory systems. Because of these results, the reconfigurable architecture presented in this chapter represents a new, useful approach to multi-processor computer organization.

CHAPTER V

An Example Implementation

...something like the computer existed, in the form of two partially assembled prototypes. ... They called this part of the project "debugging."

– Tracy Kidder
The Soul of a New Machine

In order to establish the feasibility of the new architecture and to provide insight into implementation details, a design and construction project was undertaken in association with the thesis. The first section of this chapter will provide the details of the implementation. Next a section describing an example application implementation for the machine will be presented. Finally, the chapter will close with an evaluation of the machine and the example program developed for it.

5.1. An Example Machine

5.1.1. The Processor Modules: For this implementation, microprocessors were chosen as the CPUs. A number of alternatives were considered. Among these were the use of minicomputers as the component processor modules. However, their use was rejected due to the financial limitations on the project. Likewise, most of the component processor modules are not existing microcomputer systems. Two of the three processor modules were constructed especially for the implementation and the other was an existing system.

In this system two types of CPUs exist. There is one DEC LSI-11 microprocessor and two Motorola 68000 microprocessors. The actual choice of processors was influenced by several factors. First it was desired that the processors not all be of the same type in order to demonstrate the use of heterogeneous processors in the system. Second, familiar and readily available processors were selected. Also the the existence of readily available and inexpensive peripherals for the LSI-11 influenced its selection for the system. Because the LSI-11 has a somewhat lower performance than the 68000s and since its choice was influenced by the set of available I/O devices, it is primarily intended for use as an I/O processor. Both of the 68000s run at an 8MHz clock speed and are used as user programmable processor modules. In addition, one of the 68000 modules is used to run the non-I/O portions of the operating system.

As discussed earlier, all memory in this system is local to the processor modules. Therefore, the memory design in this system is very conventional. For the LSI-11, a commercially available memory subsystem, the MXV-11 multi-function card, was employed. This board has 32kB of memory and two external serial I/O ports. Since the LSI-11 is intended to be an I/O processor, this amount of memory was considered adequate. It would be desirable for a substantial memory space to be available as I/O buffering space (primarily for disk cacheing), but the addressing limitation of 64kB on the LSI-11 makes further expansion limited. The choice of memory for the 68000s was also made to create a simple and adequate (though not large) memory system. Consequently, static memory devices were chosen to avoid the complications with dynamic memory refreshing. One of the 68000s has 32kB of memory. This is the processor which runs the operating system. The other has 128kB of memory. In addition to this static RAM, each 68000 processor module has 16kB of EPROM for bootstrapping and other control functions. In accordance with the overall simplicity of the implementation, no memory management was included to support multiple processes on each processor.

As mentioned earlier, most of the external I/O goes through the LSI-11. It has two serial RS-232 ports and an RL01 5MB disk drive. In addition, the 68000 with the smaller memory space has one RS-232 port for use with a console terminal. This complement of external I/O was included due to its cost and ease of integration. It is considered to be adequate for this early incarnation of the machine.

Figure 5.1 illustrates the processor modules used in the example implementation.

5.1.2. The IPCN: The network controller in this implementation is an Intel 8085 microprocessor with some memory and I/O ports. It has 2kB of EPROM which contains its control program. There are two 8155 RAM–I/O chips that provide temporary storage and communications to the systems processor modules. The cross-point switch chip's configuration registers are mapped into the controller's address space.

The network controller, then, operates by taking requests from the processor modules in the system, translating the requests into a mapping through the cross-point and configuring the switch accordingly. Four operating system requests are implement-

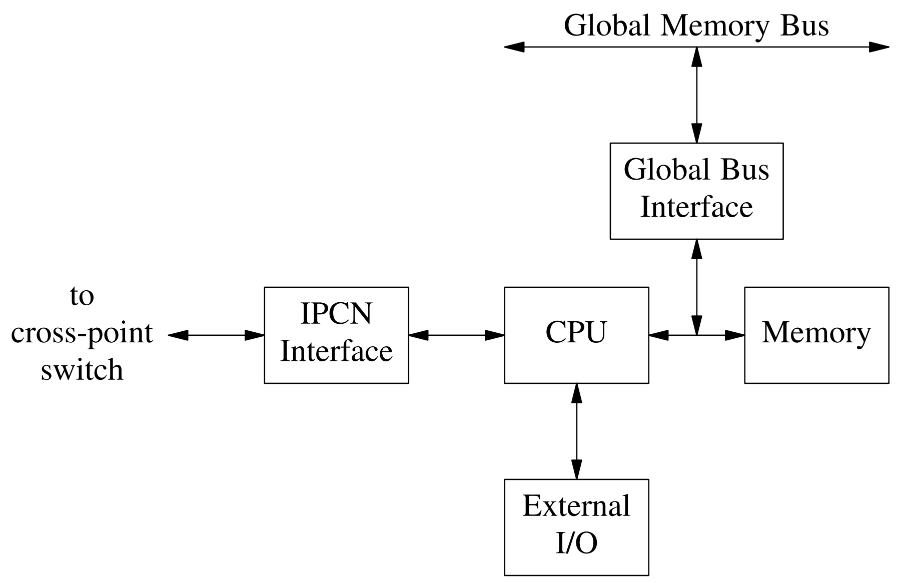

Figure 5.1

Processor Modules

ed in the network controller control software.

1) Reset
2) Query
3) Connect
4) Disconnect

The reset command initializes the IPCN to its power-up state with all ports disconnected. A query command is used to determine the connection status of a port. The port for which the status is being requested is passed to the network controller as an argument following the request byte. This request produces a result byte which is the input port to which the requested output port is connected or the value 128 if the port is disconnected. The connect request takes two arguments: an output port and an input port. Since there is only one port per processor module and since the modules cannot support multiple processes, the process to processor port translation is a direct one where process number and port number are equal. For output port n, input $2n$ to the network is connected to the transmitter output of the port and output $2n+1$ is connected to the transmit handshake. Likewise, input port m has receive data coming from output $2m$ and receive handshake going to input $2m+1$. The action of establishing a connection between output port n and input port m is to establish a path between network inputs $2n$ and $2m+1$ and outputs $2m$ and $2n+1$ respectively. A disconnect request is used to break the paths established by the connect request. For reasons discussed below, the disconnected network outputs are connected to input 31.

The interprocessor communications network is based on a Fairchild 32×32 cross-point switch chip. This device provides 32 digital inputs and 32 outputs. It is a multiplexer implementation as pictured in Figure 1.6. Along with some buffering, this chip is the reconfigurable communications network. Since the number of processors is small, a full cross-point can be used without undue complexity.

The organization of the IPCN module is pictured in Figure 5.2.

The use of a global bus to effect bulk I/O transfers was discussed earlier. This approach to bulk transfers was chosen for this implementation. All processor modules have a bus interface that couples the local memory bus to this global bus. The 68000 module with the smaller memory space was chosen as the bus master in accordance with its purpose as host for the operating system. Its bus interface, therefore, is the only one capable of driving the address and control lines on the global bus. When accessed via the global bus, the memory spaces of all other processors are in the addressing space of the bus master. The bus interfaces for the other processor modules all provide a DMA path between the bus and the local memory.

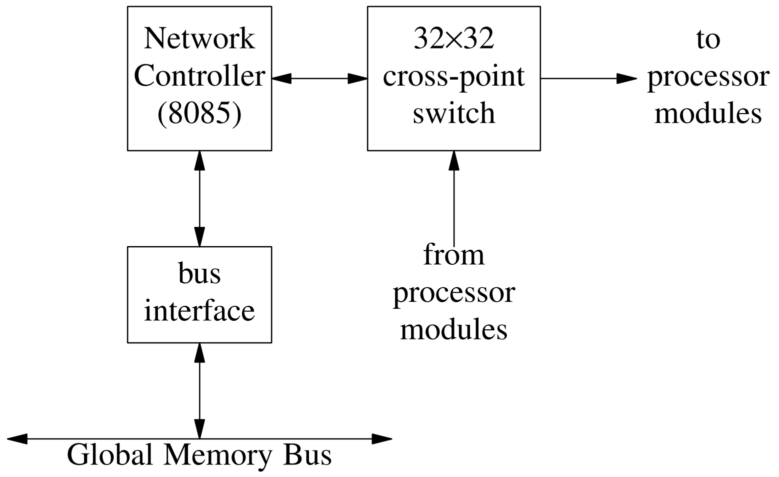

Figure 5.2

IPCN Module

To summarize the organization described above, Figure 5.3 shows the overall organization for the example implementation.

5.1.3. Interfacing the Processor Modules to the IPCN: Each of the processor modules has one bi-directional serial port connecting it to the IPCN. Serial data transmission was chosen because only two of the cross-point switches were available at the time of construction. These ports operate at 10Mbit/sec from a central clock distributed from the IPCN. Each port is designed as a memory mapped parallel-to-serial and serial-to-parallel converter set. These full-duplex ports can transfer one byte in each direction in 1μsec. In order to synchronize the transmit and receive state machines, a start bit is used. Sometime between zero and one bit times after the memory cycle loading the transmitter is completed, the start bit is sent followed by the eight data bits. When the start bit has been shifted through the receiver serial-to-parallel converter, the receiver is stopped. A read from the receiver resets its state machine.

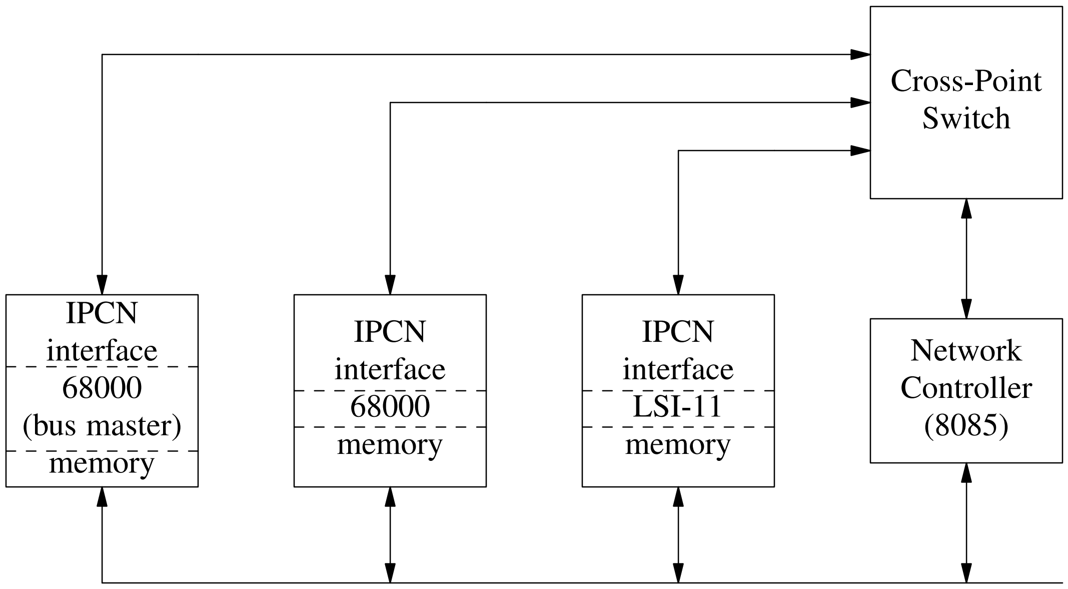

Figure 5.3

An Example Implementation

In order to provide information regarding the reception of characters, a handshaking signal is generated by the receiver hardware and sent to both the transmitting and receiving processors. The active level of this signal indicates that a byte has been received by the destination processor module and not yet read by its CPU. Conversely, the inactive state is used by a transmitting process to indicate that the connected receiver is ready to receive a byte. In this implementation, these handshaking signals must be polled by the software on the processor modules. Because this handshaking signal is sent back to the transmitting processor module, each half-duplex data connection established through the IPCN must actually be a full-duplex connection pair. One is needed for the data going from transmitter to receiver, and one for the handshaking signal going from the receiver back to the transmitter.

The general form of the IPCN interfaces is shown in Figure 5.4.

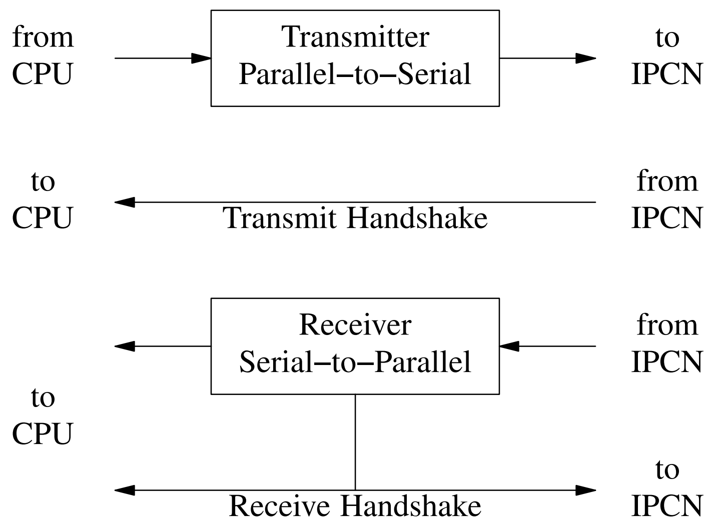

Figure 5.4

IPCN Interface

The design of these interfaces brings out a requirement on the communications network. Of concern, here, is the case of input ports that are left unconnected. Since a start bit is used to indicate the reception of a byte, unconnected inputs should be tied to the logic level opposite of the start bit. This is the same value that a transmitter sends between transmissions. Likewise, unconnected transmitter should have their handshake lines tied to the active level. This is done in order to indicate that no receiver is on the other end ready to receive a byte. To provide this disconnected state, two of the inputs to the cross-point switch are connected to the two logic states. When a connection is broken, both the transmit handshake and the receive data of the ports being disconnected are connected by the network controller to the appropriate logic state.

The interfacing between the processor modules and the network controller consists of the configuration channel operating over the global memory bus between the 68000 intended for the operating system and the controller. This channel is implemented as a mailbox in the 68000s address space and as one of the I/O ports in the 8085 I/O space. The operating system then requests half-duplex connections and the network controller allocates the full-duplex connection for the data connection requested and for the handshaking signal.

5.2. An Example Application

To supplement the example hardware implementation, a small example application program has been developed. The purpose of this portion of the project is to demonstrate the functionality of the hardware and to illustrate the details of using this system.

5.2.1. The Algorithm: One of the most commonly studied and programmed tasks in computer science is sorting a group of items. The sorting of a group of numbers is the task chosen for this sample application.

Since this application is intended to illustrate the use of a multi-processor system, a sorting algorithm that makes use of the available parallelism must be chosen. The algorithm chosen here makes use of the parallelism by dividing the list of numbers among all but one of the processors available. Each of the processors receiving a portion of the list then sorts its sub-list using some conventional sorting algorithm. The remaining processor takes the streams of sorted data from the other processors and performs a merge sort on them to produce the final sorted list. This division of the task works best when each of the sub-list sorts produces a steady stream of output as the sorting process continues. In this way, the merging takes place in parallel with the other sorting operations. An alternative approach is to divide the sorting task among all of the available processors and let them completely sort the component sub-lists. Following this phase of the program, one of the processors then performs the merging of its sub-list with those of the other processors. Throughout the remainder of this discussion of the algorithm, the sort will be taken to be in descending order. The remainder of this discussion will be for the approach implemented (the one using a separate processor for the merging).

For the merging process, the sorted sub-lists of the other processes can be viewed as sorted queues with the largest value at the front. Further, it is taken that the merging process has access only to the front value on this queue and that it can determine when the queue is empty. The empty state can be determined with a count of the number of items removed or by a marker at the rear of the queue. There should also be an output list on which the sorted result of the whole algorithm will appear. With these mechanisms in place the merging process can be described with the following algorithm:

```
repeat until all queues are empty
    find the largest value of those at the fronts of the queues
    remove that value from its respective queue
    place that value on the end of the output list
```

When all of the input stacks to the merging process are empty, the output list will have the sorted results of all of the data that was divided among the various processors.

The performance enhancement that can be expected from using multiple processors in this application is dependent on the merging process. The sorting sub-processes all execute in parallel without any communications among them. Therefore, if the merging process can take the data as fast as they can produce it, a close to linear speed-up should be seen in the number of processors used for this portion of the task. If the overhead in dividing the data among the various sub-processes can be ignored (such as when all processes are directly loaded from I/O) then the speed-up will be linear. Since the merging process depends on the other processes to supply it's data, it's effect on the overall performance is based on how well it is matched to the number and speed of the other processes. If the process cannot take the data as fast as the other processes are generating it, then the other processes will be made to wait causing an unacceptable delay. This places an effective limit on the number of sub-processes that can be supported by one merging process. To extend that limit, multiple processors can be used to host the merging process. If the merging process is exactly matched to the number and speed of the other sorting processes, then none of the processes will experience delay and the speed-up will be completely linear. The third possibility is that the processor hosting the merging process will be faster than needed. In this case the speed-up will be less than linear due to this process waiting on the others. For n processors each hosting one sub-process, the speed-up will be between $n-1$ and n.

5.2.2. Details of the Implementation: Since the implementation discussed in this chapter has three processors, the sorting algorithm will be divided into two conventional sorting sub-processes and one merging process. The two 68000s in the system host conventional straight selection sorts and the LSI-11 hosts the merge sort.

The basic form of the algorithm running on this LSI-11 has already been described. However, a number of other details have not been considered. In particular the loading of the queues has not been described. Since the sorted sub-lists come from processes residing on other processors, the queues are loaded from the IPCN receiver on the LSI-11 processor module. Because there are two processes suppling these sub-lists, there are two queues to hold this data.

Another point that must be considered in the implementation of this algorithms is the management of the queues. Several options are available. One approach to this is for the sub-processes to keep the sorted queues and return sorted elements on request from the LSI-11. However, because of the lack of interrupts on the IPCN ports and the unpredictable nature of the requests, this implementation is ill suited to this approach. Instead queues capable of storing the entire lists will be kept on the LSI-11 module. Sorting sixteen bit integers with 32kB of memory means that there is space for 16384 elements. Because of the size of the queues and the output list, the maximum number of elements to be sorted is 8192. Some space must also be reserved for stack and temporary storage. For these reasons 7168 elements will be selected as the maximum number for the example application.

The complete algorithm for the merge sort running on the LSI-11 processor module is then as follows:

```
get the number of elements
repeat until output list is full
    if the sub-sorts have new results
        add them to the queues
    if both queues have some elements
        if value at head of queue 1 is greater than at queue 2
            remove head of queue 1 and place it on the output list
        else
            remove head of queue 2 and place it on the output list
    else
        if all of the elements from one of the sub-sorts have been
        entered in the output list
        and there is an element in the other queue
            remove head of the other queue and place it on the output list
    notify processor 0 that the sorting is complete
```

The first task for the sorting sub-processes is to generate the random numbers that are used for the sorting run. These numbers are generated with the familiar “middle squares” algorithm. The overall algorithm for generating random numbers on both 68000 processor modules is as follows:

```
Running on Processor 0: (other processor is #1)

request a connection between output port 0 and input port 1
request a connection between output port 1 and input port 0
get the number of elements to be generated from the user
get two random number seeds from user
start random number generator on processor 1
pass it its seed
$last\ element \leftarrow local\ seed$
$old\ seed \leftarrow local\ seed$
repeat for half of the number of elements
    if $last\ element = 0$
        $new\ seed \leftarrow (old\ seed \times 2 + 643) \bmod 65535$
        $last\ element \leftarrow new\ seed$
        $old\ seed \leftarrow new\ seed$
    $last\ element \leftarrow \frac{(last\ element)^2}{256} \bmod 65535$
    place $last\ element$ in the list
wait until processor 1 has completed its random number generation

Likewise on Processor 1:

read the seed from processor 0
$last\ element \leftarrow local\ seed$
$old\ seed \leftarrow local\ seed$
repeat for half of the number of elements
    if $last\ element = 0$
        $new\ seed \leftarrow (old\ seed \times 2 + 643) \bmod 65535$
        $last\ element \leftarrow new\ seed$
        $old\ seed \leftarrow new\ seed$
    $last\ element \leftarrow \frac{(last\ element)^2}{256} \bmod 65535$
    place $last\ element$ in the list
notify processor 0 that random number generation is complete
```

In the random number generation portion of the example application a bi-directional channel of communications was established between the two 68000 modules (processors 0 and 1). For the actual sorting process, however, the system topology needs to be different. Since both of the 68000 modules must send information to the LSI-11 module, it would be desirable that the LSI-11 have two IPCN ports which could be connected to the two 68000s. This is not the case. To simulate this, the IPCN could be used to switch the LSI-11 input port between the two 68000 output ports. This is not very efficient due to the centralized configuration control. The simplest approach that fits into the implementation at hand is to configure the processors in a pipeline with the output port of processor 1 connected to the input of processor 0 and the output of processor 0 connected to the input of processor 2 (LSI-11 module).

The complete processor 0 part of the sorting implementation is as follows:

```
request a connection between output 1 and input 0
request a connection between output 0 and input 1
get the number of elements to be sorted from the user
start the sorting program on processor 1
pass the number to processor 1
request a connection between output 0 and input 2
start the merge sort on processor 2
pass the number of elements on to processor 2
let $n$ be the number of elements
let $x$ be the array of elements
$i \leftarrow 0$
repeat until $i = n$
    if processor 1 has information ready
        pass it on to processor 2
    $y \leftarrow i$
    $max \leftarrow -\infty$
    repeat until $y = n$
        if $x_y > max$
            $max \leftarrow x_y$
            $p \leftarrow y$
        $y \leftarrow y + 1$
    $x_p \leftarrow x_i$
    $i \leftarrow i + 1$
    send $max$ to processor 2
if all elements from processor 1 have not been passed to processor 2
    repeat until all elements have been passed
        wait until processor 1 has an element ready
        pass it on to processor 2
request a connection between output 2 and input 0
wait until completion message arrives from processor 2
```

Likewise, the straight selection sort on processor 1 goes as follows:

```
get the number of samples from processor 0
let $n$ be the number of elements
let $x$ be the array of elements
$i \leftarrow 0$
repeat until $i = n$
    if processor 1 has information ready
        pass it on to processor 2
    $y \leftarrow i$
    $max \leftarrow -\infty$
    repeat until $y = n$
        if $x_y$ > max
            $max \leftarrow x_y$
            $p \leftarrow y$
        $y \leftarrow y + 1$
    $x_p \leftarrow x_i$
    $i \leftarrow i + 1$
    send $max$ to processor 1
send a completion message to processor 0
```

To facilitate the measuring of the performance of this system, a single processor version of this algorithm is also available on processor 0.

5.2.3. Results: During the course of developing the sorting algorithm to run on the example implementation, the LSI-11 was revealed to have a heat related malfunction. The exact problem was intermittent, so small sorting runs could be made to demonstrate the correctness of the implementation. However, only a few larger sorts ran successfully. These indicated that the performance of the three processor configuration was about twice that of the single processor version of the algorithm. It is believed that the observed speed-up is significantly smaller than linear because the merging process on the LSI-11 spend most of its time waiting on the other sorting processes to produce results. By matching this process to more sub-sorts, the speed-up could be pushed closer to linear.

5.3. Evaluation of the Implementation

The example implementation was undertaken in order to gain insights into implementational details as well as establish the feasibility of the architecture. Because of this, it is important to review the results of this undertaking and to enumerate the lessons learned from it.

Overall, the system operated as designed and illustrated the basic principles of the architecture as described in this chapter. However, during the course of designing and constructing a detailed implementation, a number of insights into the issues raised in this chapter did arise.

5.3.1. The Processor Modules: The CPU−memory−I/O portion of each processor module is of conventional design. Although no insights into the design of such systems was gained from this project, it was determined that the use of these conventional computers as component processors works well. Furthermore, the implementation demonstrated that heterogeneous processors can be easily integrated using the new architecture. The use of the LSI-11 processor module also demonstrated the facility with which existing computers can be integrated into this system as processor modules.

In contrast to the results of the previous paragraph, the use of the global bus and DMA interface on each module demonstrated that it it not generally desirable. It was believed that providing one DMA path into each memory space under the control of one processor would be more cost effective than individual DMA channels for the IPCN interfaces. The design and construction experience has proven otherwise. The provision of this portion of the system was at least as complicated and hardware intensive as individual DMA ports, and it provides a limiting factor on the extendibility.

5.3.2. The IPCN: The design of the communications network and controller can be considered successful. Aside from desirable expansion, no lack of functionality could be found.

5.3.3. The Process Module to IPCN Interface: A review of the overall complexity of the resulting system shows that if enough of the cross-point devices had been available, then using several of them to switch parallel data would have been a better implementation. Since two input/output pairs of the cross-point are used for each half-duplex connection, sixteen such connections are possible with the one switch used. A parallel interface would use eight data lines and one handshake and require nine input/output pairs per half duplex connection. Sixteen of these could be implemented with five of the switch chips.

In planning the software implementation for this example realization, it has become clear that the use of polled I/O for the IPCN can cause an undesirable overhead for many applications. It was originally foreseen that the implementational details of the IPCN ports could follow those for I/O subsystems. For these ports, though, delays are more serious since another processor rather than an I/O device is waiting. Interrupt and DMA driven system seem to be the only reasonable approaches to these communication ports. It is for these reasons that new processor modules for this implementation are expected to utilize interrupt or DMA driven interfaces.

It has also become clear that the use of a single processor to make configuration decisions is too restrictive for many applications. This is particularly true if the number of IPCN ports per processor is small as in this implementation. For most general purpose environments, it will be necessary to provide a mechanism, such as the one described earlier, to allow all processor modules to request configuration changes.

Despite the deficiencies discussed above, this implementation has shown the new architecture to be one that is realizable. By making informed and intelligent decisions regarding some of the implementational details, a designer can create an incarnation of this new architecture which realizes the beneficial characteristics discussed earlier.

CHAPTER VI

An Analysis of Multi-Processing

Aristotle could have avoided
the mistake of thinking that
women have fewer teeth than men
by the simple device of asking
Mrs. Aristotle to open her mouth.

– Bertrand Russell

In Chapter Three, a mechanism of characterization was developed that measured the potential degree to which the processors can communicate. Taken as a performance measure, this degree of coupling indicates the extent to which cooperative processing can take place. An architecture with a degree of coupling close to one, will demonstrate few delays due to inter-processor communications. On the other hand, systems with small degrees of coupling cannot quickly pass cooperative information among the processors.

It is well known that no single measure of performance can completely characterize the relative speed that any given computer system can perform tasks. For this reason, another system of evaluation will be discussed. In particular, this characterization of computational power will be used to relate the architecture presented in this thesis with some of those discussed in Chapter Two.

6.1. The Janakiram and Agrawal Evaluation

At the North Carolina State University Center for Communications and Signal Processing, Janakiram and Agrawal have described a set of five characteristics pertaining to processor interconnection networks.[38]

The first of these is the average distance between processors. This distance is the average number of inter-processor links that messages must pass through. They give the following expression for average distance:

$$D = \frac{\sum_{d=1}^{r} dN_d}{N - 1}$$

where:

N_d = the number of processors a distance d apart

r = the maximum number of links between any two processors

N = the total number of processors

The smaller D is, the more efficiently cooperative information can be transferred. Furthermore, they define a normalized average distance to be the average distance multiplied by the number of ports per processors as follows:

$$D_n = D \times \frac{Ports}{Computers}$$

The second characteristic described by Janakiram and Agrawal is the total number of communications ports in the system. When a system has more communications ports, there is more opportunity for inter-processor cooperation.

Next the ease of routing messages has been described as the third characteristic. It is desirable for there to exist a routing algorithm that will find an exact path between any two processors. Additionally, it is advantageous for the algorithm to produce the shortest path.

The fourth characteristic which they described is a measure of the system's fault tolerance. If one of the processors in the system fails, then the preservation of the existence of a path between any two remaining processors indicates some degree of fault tolerance.

The last characteristic described by Janakiram and Agrawal is the ease of expansion. It is desirable for a network of processors to be expandable without necessitating a complete re-design of the system. Likewise, if the addition of processors causes large growth in the complexity of the system, then there is a practical limit on the size that such a system can achieve.

6.1.1. Application to the Reconfigurable Architecture: The architecture presented in this thesis can be described according to the characteristics listed above. First the average distance is a measure of the delay contained in transmissions between processors. For the reconfigurable architecture, the IPCN can be configured such that all messages traverse only one hop. Unless the network can be configured as a fully connected mesh, this characteristic cannot hold for all times. However, since it can be reconfigured upon demand, a one hop path can always be configured.

The total number of ports is not a fixed characteristic of the architecture, but is one of the particular implementation. Nevertheless, it can be stated that in general there will be at least as many ports as there are processors. More typically, each processor will have $\log_2(n)$ ports for an n processor system to accommodate a hypercube topology. This gives a total of $n \log_2(n)$ ports in a typical implementation.

Both the third and the fourth characteristics (well-behaved routing and fault tolerance) arise from the direct connection nature of the IPCN. In any cross-point or tree of cross-point switches, there will exist as many paths between two processors as the minimum number or ports on the two processors. Also since the routing does not go through any of the processors, the system communications will be tolerant of failed processors.

The final characteristic of expandability is also possessed by this architecture. It is clear that the complexity of the IPCN will grow with the addition of more processors, and it will do so at a rate greater than unity. But since the nature of the IPCN is much simpler than individual processors, a substantial number of processors may be added before the complexity and cost of the IPCN as a whole exceeds the processing components of the system. This is particularly true if the IPCN takes a hierarchical form rather than a pure cross-point switch.

6.1.2. Application to Other Architectures: In order to put this new architecture in the proper perspective, it is important to also evaluate other machines according to this criteria.

The average distance for the commercial shared memory systems and for C.mmp is one. For the Cm* shared memory configuration, the average distance depends on the exact configuration. Memory accesses to modules in the same cluster take place at a distance of one. For those to modules in clusters that are connected to one of the same intercluster busses, the distance is two. The exact configuration of an implementation determines the value of D. As an example, a system with twelve processors per cluster and three clusters connected to the same two intercluster busses. This implementation has a $D = 1.69$. Likewise, each quadrant of the ILLIAC IV has an $r = 7$ and the following table lists the number of processing elements at each distance.

Table 6.1

Frequency of Connection Distances in the ILLIAC IV

d	N_d
1	4
2	8
3	12
4	15
5	12
6	8
7	4

For the ILLIAC IV, then, $D = 4$. In the case of the MPP with the edges open,

$$D = \frac{\sum_{d=1}^{r} dN_d}{N-1} = \frac{\sum_{x=1}^{127}\sum_{y=1}^{127}(x+y)}{16383} = \frac{\sum_{x=1}^{127} 127x + \sum_{y=1}^{127} 127y}{16383} = \frac{254\sum_{i=1}^{127} i}{16383} = 126.02$$

For all of the shared memory systems discussed in Chapter Two, the total number of ports is equal to the total number of processors. The ILLIAC IV has four ports per processing element, or a total of 256 ports per quadrant. There are also four ports per processing element in the MPP for a total of 65,536 ports of which 256 are not used when the edges are disconnected.

For all of these machines, there exists a straightforward method for determining shortest path routing. Likewise, a single processor could fail in any of these systems and the system would still be able to pass messages between any other pair of processors (except of course in two processor implementations).

Finally, the ease of expandability varies widely among these architectures. The commercial shared memory systems could be expanded with little increased hardware, but the performance degradation due to memory contention limits the extent to which this can be carried. C.mmp can conceptually be expanded and well designed programs can minimize the memory contention. However, the cost of the cross-point switch grows as the square of the number of processors in the system, placing an effective limit on the number of processors in the system. The Cm* design allows for a high degree of expandability without severe performance degradation. This is because contention for each Kmap is limited to requests generated by twelve processors. The ILLIAC IV is designed to be expandable in the number of quadrants. The architecture also could be designed around any square array of processors. This is also true of the MPP array.

The following table summarizes the machines described in this section. The starred entries indicate that the parameter is dependent on the particular implementation. Entries of *n* indicate that the parameter is equal to the number of processors in the system.

Table 6.2

Summary of Evaluation on Several Multi-Processor Systems

Machine	*D*	Ports	Routing	Fault Tolerance	Expandability
Reconfigurable Architecture	1	*	yes	yes	medium
Commercial Shared Memory	1	*n*	yes	yes	low
C.mmp	1	*n*	yes	yes	low
Cm*	~1.6*	*n*	yes	yes	high
ILLIAC IV	4	256	yes	yes	medium
MPP	126	65,536	yes	yes	medium

6.2. Summary

Despite the evaluations described in this chapter, the only meaningful statement about the relative performance of two systems is the running times of the appropriate applications programs. This has led to the use of benchmarks to characterize the performance of computer systems. Even these benchmarks, though, have proven to be of limited value in comparing various computer systems. The comparison of multi-processor systems has been particularly ill served by performance measures due to the wide variation in software implementation techniques for multi-processing systems. Furthermore, time constraints in this project have not allowed for any extensive comparative benchmarking of the proposed architecture.

CHAPTER VII

Conclusion

The beginnings and endings
of all human undertakings
are untidy.

– John Galsworthy

In this, the last chapter, it is time to review the project. The previous chapters presented a new multiple processor computer architecture. This was done in the perspective of previous work in the same area and in the context of a model of multi-processing that emphasizes the system's degree of coupling. Finally, some analysis was presented in Chapter Five.

7.1. Retrospective

The last section of Chapter Four evaluated the architecture and the last section of Chapter Five the example implementation to determine what was learned and how well they met the objectives. To summarize the results of those sections, the implementation project revealed that several aspects of the example implementation did not have the expected advantages. These include the use of a global memory bus, the use of polled IPCN interfaces and the single processor configuration control structure. It was also indicated that parallel data transmission through the IPCN is generally preferable from both a performance and a complexity viewpoint.

The general architecture was also shown to be a good high performance system that is extendible and appropriate to a wide range of applications. It was also described as a good compromise between the flexibility and expandability of a LAN and the high performance of a shared memory system. In addition the architecture is more flexible than other high performance communications based multi-processors.

7.2. Future Directions

Clearly technology does not stay still. It is also clear that this work will not be the final word on multi-processing. For these reasons, it is appropriate to consider "Where do we go from here?" Determining the system's applicability with systems and applications software must certainly be considered a top priority. The software implications of the architecture will be considered in the next section. From a strictly organizational and hardware viewpoint, the success of the example system leads to the feasibility of a more extensive implementation. Due to their high availability and their low cost on the used market, PDP-11s or LSI-11s would make good candidates for such a project. These general purpose systems could also be combined with some of the recent powerful digital signal processor chips for the numerically intensive portions of applications.

Reconfigurability, as an organizational concept, appears to be a powerful one. Since it is so general, it is difficult to assess improvements to the general idea. However, one possible extension to the particular architecture presented here comes to mind.

Consideration of the model in Chapter Three leads to a generalization by providing reconfigurability of CPU-memory interconnections as well as CPU-CPU connections. Such a system would be of great advantage to the multi-programming environments since processes could be moved among processors with ease. Unfortunately though, this added flexibility is not free. This kind of extension would make it more difficult to use existing general purpose computer systems and designs. It would also be significantly more complex and therefore more expensive. The decision to use such a system would be predicated upon an expected improvement in performance that would compensate for the added complexity and expense. One can also generalize the model and architecture to SIMD computers. This would allow for a reconfigurable interconnection of control units and processing elements.

7.3. Software Aspects of the Architecture

No discussion of any computer architecture could be complete without a discussion of its impact on the software that it is to run. This is especially true of multi-processor systems in which the architectural characteristics can have such a great effect on all aspects of the software environment. In this section, three broad areas of that software environment will be discussed. These areas are the operating system, the programming languages and compilers, and the applications programs.

7.3.1. The Operating System: The implications that this architecture has for the operating system are far reaching, although not radical. Most notable is the opportunity for distributing the operating system over several processors. This fits in well with the modern conception of the operating system as a cooperating collection of processes. In this system, any subset of the processes may be run on any of the processors. The only restrictions arise from any differences that may exist in the processors. Clearly, the I/O device management portions of the system need to run on the processors to which the devices are attached. It also makes sense for the file management processes to run on the processors to which storage devices are attached. The design of these portions of the operating system can be taken directly from single processor systems.

In any multi-processing system where processors may be working on different processes, the operating system level of memory management is not as clearly defined as in a single-processor system. If the component processors don't have the ability to host more than one process simultaneously, then the concept becomes directly tied into process management. However, such a system is likely to be too restrictive in practice. Each processor should have some degree of hardware memory management instead. This opens up two options to the software management: local and global memory management. If there exists a processor that has access to all of the system's memory (as in the example system), then an opportunity exists for global control of the memory resources. In general, it would be more desirable for some degree of memory management to exist at each processor.

Of course, the process mangement portion of the operating system is also affected by the system organization. It is most appropriate to consider each stream of execution on each processor to be a separate process, regardless of how many may be cooperating on a given task. With this in mind, the problem of process scheduling becomes one of keeping processors busy and of keeping processes from waiting. Clearly, if the processors are only capable of hosting one process each, then the mechanism is simple.

Each time a process is created, it is made to run on a free processor—if one exists. If no processors are idle, then no new processes can be created. Practical systems would, of course, require that the individual processors be capable of time-sharing several processes. In this case, a load balancing problem comes into play.

A number of interesting aspects of multi-processor operating system design have been ignored in this discussion. This is because it would be all too easy to embark on a research project into such operating systems here.

7.3.2. Languages and Compilers: Like the operating systems, the field of languages and compilers for multi-processing systems is an interesting topic of study in its own right. Nevertheless, there must be some features in any language system for multi-processing in support of multiple execution streams. One approach has the programmer using the pre-existing multi-tasking features. For instance, in a UNIX environment, one could use the *fork* system call to create a new process that executes on another processor, as discussed above in operating system process creation. In addition, the pipe facilities could be used in controlling the connections through the IPCN. Another type of support of IPCN connections might be similar to the Berkeley UNIX socket inter-process communications support.

An academically more interesting approach is to use the compiler to detect parallelism and to generate the code to control the processes and communications facilities. Since this is a relatively new area of research, it is inappropriate to delve into much detail about it here. However, there is reason to believe that a reconfigurable architecture would be well-suited to such an approach. The compiler could assume any model of processor interconnection that is appropriate for the task and then generate the configuration requests at the initialization of the process.

7.3.3. The Applications Software: As discussed above, the applications programmer has the freedom to take advantage of multiple processors at a relatively high level with existing and familiar facilities. At a lower level of parallelism, the reconfigurability of the system lends itself well to any application that analysis has shown to fit well into any architecture. For image processing, the system could be configured as an array, but for FFTs, the system might be better configured as a hypercube. In general, if there are enough ports on the processors, any application that has been implemented for a specific architecture may be transferred with little effort to this architecture.

The conversion of programs written for shared memory systems is more complex. For these programs, a generally applicable (though not very efficient) approach is to create a shared memory management process. This process runs on one processor that acts as the repository for shared memory. All accesses to this memory space take place through requests passed to the shared memory management processor through the IPCN. While this is not as efficient as true shared memory, it does fit in well with the current methods of interlocking memory locations.

In general, then, this multiple processor computer architecture provides a high-performance, extendible computing system. It is well-suited to being constructed as a collection of existing general purpose single processor systems. Finally, the architecture is applicable to a wide range of environments. It can be used in traditionally single processor environments or as an alternative for existing multi-processor architectures.

REFERENCES

[1] John von Neumann. "First Draft of a Report on the EDVAC," *Papers of John von Neumann on Computing and Computer Theory.* (The MIT Press, 1987) pp. 17-82.

[2] Arthur W. Burks, Hermann H. Goldstine and John von Neumann. "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," *Papers of John von Neumann on Computing and Computer Theory.* (The MIT Press, 1987) pp. 97-142.

[3] Nancy Stern. *From ENIAC to UNIVAC: An Appraisal of the Eckert-Mauchly Computers.* (Digital Press, 1981) pp. 116-136.

[4] R. W. Hockney and C. R. Jesshope. *Parallel Computers: Architecture, Programming and Algorithms.* (Adam Hilger Ltd., 1983) p. 9.

[5] C. Gordon Bell, J. Craig Mudge and John E. McNamara. *Computer Engineering: A DEC View of Hardware Systems Design.* (Digital Press, 1978) pp. 162-164.

[6] M. Satyanarayanan. *Multi-Processors: A Comparative Study.* (Prentice-Hall, Inc., 1980) pp. 5-17.

[7] Kai Hwang and Fayé A. Briggs. *Computer Architecture and Parallel Processing.* (McGraw-Hill Book Company, 1984) pp. 687-690.

[8] ibid, pp. 690-693.

[9] Satyanarayanan, pp. 77-92.

[10] Ibid.

[11] John Fu, James R. Keller and Kenneth J. Haduch. *Digital Technical Journal.* "Aspects of the VAX 8800 C Box Design." (Digital Equipment Corporation, Feb. 1987) pp. 47-50.

[12] Stuart J. Farnham, Michael S. Harvey and Kathleen D. Morse. *Digital Technical Journal.* "VMS Multiprocessing on the VAX 8800 System." (Digital Equipment Corporation, Feb. 1987) pp. 113-116.

[13] Hwang and Briggs, pp. 714-717.

[14] Satyanarayanan, pp. 95-116.

[15] Ibid.

[16] Ibid.

[17] Satyanarayanan, pp. 120-141.

[18] Ibid.

[19] Hwang and Briggs, pp. 398-404.

[20] Ibid.

[21] Ibid.

[22] Ibid, pp. 422-430.

[23] Ibid.

[24] Nicolas Mokhoff. *Computer Design.* "Hypercube Architecture Leads the Way for Commercial Supercomputers in Scientific Applications." (PennWell Publishing Company, May 1, 1987) pp. 28-30.

[25] Ibid.

[26] W. Daniel Hillis. *Scientific American.* "The Connection Machine." (Scientific American, Inc. June 1987) pp. 108-115.

[27] C. Gordon Bell and Allen Newell. *Computer Structures: Readings and Examples.* (McGraw-Hill Book Company 1971) pp. 477-485.

[28] Ibid.

[29] Hwang and Briggs, pp. 32-35.

[30] Ibid.

[31] Ibid.

[32] Ibid.

[33] Hwang and Briggs, pp. 35-37.

[34] Ibid, pp. 37-40.

[35] Hockney and Jesshope, pp. 29-31.

[36] Ibid, pp. 31-43.

[37] Ibid, pp. 43-47.

[38] Virendra K. Janakiram and Dharma P. Agrawal. ''Multicomputer Architecture Evaluation.'' (Center for Communications and Signal Processing, 1983) pp. 2-4.

TABLE OF CONTENTS