Implementation of a Hierarchical Control System on a BBN Butterfly Multiprocessor: Initial Studies and Results

Mari Thune and Bir Bhanu

Computer Science Department
University of Utah
Salt Lake City, Utah 84112, USA

UUCS-87-008

31 March 1987

Abstract

This report discusses the introductory work with implementing a parallel version of a hierarchical control system named CAOS (Control using Action Oriented Schemata) on the BBN Butterfly Multiprocessor. First, an overview of the BBN Butterfly and how the control system can utilize the parallel processor is given, followed by a discussion of a partial implementation and future work. Finally, a more extensive overview of the Butterfly hardware and comments on the operating system Chrysalis and the Uniform System functions are included.

This work was supported in part by NFS grants DCR-8506393, DMC-8502115, ECS-8307483, and MCS-8221750.

Table of Contents

1 Introduction 1
2 Overview 2
3 Implementation 4
4 Implementation Difficulties 5
5 Future Work 6
6 The Butterfly 7
7 The Butterfly Configuration 7
7.1 The Processor Node 9
7.2 The Butterfly Switch 10
7.3 The Butterfly Clock 13
7.4 The Butterfly I/O Card 14
7.5 The Multibus Adapter 15
8 The Butterfly at The University of Utah 15
9 Technical Details 16
10 Butterfly Performance 16
11 Chrysalis - The Operating System 19
12 The Uniform System Approach 19
13 References 21
14 Appendix 22

List of Figures

Figure 1: A Possible Butterfly Set-up. 8
Figure 2: Processor Node Components and Main Interconnections. 9
Figure 3: Routing of Data Through a 2-input Switch Node. The data package arrives serially, the "head" being the routing bit. It is stripped off, and the package is routed down (1 - down, 0 - up). 11
Figure 4: Routing of Data Through 4-input Switch Node. Two routing bits are needed, since there are 4 possible outputs. 11
Figure 5: Resolving Path Contention. If a message arrives at a switch node, and the indicated output is busy, it is rejected. A message is sent back to the processor, indicating it has to try again. 12
Figure 6: Switch System for 4 Processors. B = 2, N = 4, $(4/2)*\log_2 4$ = 4 switch nodes, $\log_2 4$ = 2 columns of switch nodes. 13
Figure 7: Switch System For 16 Processors. B = 2, N = 16, $(16/2)*\log_2 16$ = 32 switch nodes, $\log_2 16$ = 4 columns of switch nodes. The routing of a data package is also shown. To go from a processor to a memory, four switch nodes are passed. Therefore, there are 4 bits used for routing. 14
Figure 8: "Cylinder" Model of the Butterfly Switch. 15
Figure 9: The Switch Network of the U of U Butterfly. 17
Figure 10: Results from Matrix Multiply. 400x400 matrices. Avg. 9.0 microsec. per 1 processor Multiply-Add. Uniprocessor code with local data is 6% faster. 4.8 sec. for 128 processors. 18
Figure 11: Results from Gaussian Elimination. 1200 equations. Avg. 40 microsec. per 1 processor. Uniprocessor code with local data is 7% faster. 206 sec. for 122 processors. 18

1 Introduction

This is the final report for the work done for professor Bir Bhanu during the summer of 1986 as part of a team project where we designed and implemented a hierarchical robot control system. The report will discuss the aspects involved in learning about and using the BBN Butterfly at The University of Utah to implement a parallel version of the hierarchical control system CAOS[13] (Control using Action Oriented Schemata).

Initially, we planned to have a complete parallel version of the control system running on the BBN Butterfly by the end of the summer. This was not realized, however, partially due to the machine's extreme tendency to crash when encountering errors in the code. This made development and debugging very time consuming because the Butterfly had to be rebooted almost after every program run. Code for accessing files on the host machine (VAX 11/780) and for reading in and setting up in memory all the needed information found in the knowledge base and data base was completed, however. This was running very well, but did not really feature any parallelism.

The purpose of the control system is to obtain high level goals given by the user. The control system consists of three main parts: the knowledge base, the data base and the inference engine. Using rules and metarules from the knowledge base and facts from the data base, the system can obtain the specified goals. The system is written in C, which also is supported by the BBN Butterfly, making the transportation easier. A later version was written in LISP (because of several inconveniences in C), but BBN does not have a reliable LISP compiler on the market yet, although it will be there soon.

This report discusses the part of the project which involved becoming familiar with all the available documentation on the Butterfly hardware to understand how the machine functions and can be programmed. The references include all the material read. Also, the report discusses the (partial) implementation of the parallel version of the hierarchical control system on the BBN Butterfly.

First an overview of the Butterfly and how the control system can utilize the parallel processor is given. Then a discussion of the implementation follows. Finally, there is a more extensive overview of the Butterfly hardware and comments on the operating system Chrysalis and the Uniform System functions. The report has the code written for the control system included as an appendix.

2 Overview

This section is an overview of the Butterfly and how the hierarchical control system can be able to utilize the possibilities of a parallel machine, as described in the paper we wrote for publication in IEEE's Proceedings on Robotics and Automation.

The BBN Butterfly: The serial C version of the robot control system is partially transported from a VAX 11/780 to a Butterfly Parallel Processor[12] (parallel version). It is a multiple instruction, multiple data (MIMD) machine, and is connected to a host machine which in our case is a VAX 11/780. The Butterfly may have up to 256 processor nodes interconnected by a switching network called the *Butterfly Switch*. Each processor node has a co-processor called the Processor Node Controller (PNC) which is responsible for all memory references and transfers. The local Butterfly at The University of Utah has 19 Motorola MC68020 processor nodes (one being a backup), each having a Motorola MC68881 co-processor and 1 Mbyte of memory, except two, which have 4 Mbyte of memory. The processors operate at 16 MHz, due to a frequency doubler. References over the Butterfly Switch, to remote memory, usually takes about 4 microseconds round trip.

All code for the Butterfly is developed and compiled on the host machine (VAX 11/780). The executable code is then downloaded to the Butterfly, where it is run. There are two approaches we used to program the Butterfly: *Chrysalis* functions[11] and *Uniform System* functions[8].

Each processor runs one copy of the operating system Chrysalis. This operating system is mainly written in C and supports communication and synchronization between processes running on different processors. This is done by means of dual queues which allow locking and passing of tasks between these processes, and an event mechanism used for synchronization (similar to *signals* in UNIX). Chrysalis does not provide automatic resource allocation, load balancing, or process migration, however[6]. Each user-developed program has to set up the data, create all necessary processes, and decide on which node(s) they will run. Five analogs to UNIX's seek-, open-, close-, read-, and write-functions enable access to files residing on the host machine.

Compared to Chrysalis, the Uniform System approach to programming the Butterfly provides the user with easier resource management. The Uniform System is built on top of Chrysalis and consists of several subroutines which take care of, for example, allocation of memory and processors, and generation of new tasks (processes). The user does not allocate memory space or processors explicitly, since the Uniform System takes care of the distribution of tasks on processors and provides special memory alloca-

tion routines. The Uniform System is especially suitable for homogeneous problems often found in low level computer vision programs.

Parallelism in the Control System: Exploiting parallelism in the hierarchical control system involves activating several independent parts of the hierarchical control at the same time, requiring complex communication and synchronization between various processes. Chrysalis is suitable for this. The control system uses the information available at each node in the hierarchically organized control tree, to decide if subgoals can be started up in parallel. This occurs when different alternative subgoals can achieve the same goal with approximately the same expectation of success. In addition, subgoals can be started up in parallel when all needed inputs are provided, and any use of end effectors will not result in conflicts.

One of the advantages of using multiple processors to simultaneously execute alternative goal paths, is to prevent time delay due to an alternative's failure to obtain the goal. If one of the alternatives fails, or the results are not satisfactory, the result of another can be used instead. If the alternatives were not executed in parallel, and the most promising one failed, it would take longer to achieve a goal; the next alternative would be executed only after the first had failed.

When the hierarchical control allows parallelism, the "parent" (the top most node, controlling the overall goal) has to check if there are any processors available on which to start up "child processes" (subgoals in the form of nodes on the next level in the control tree). If this is the case, the parent must also set up all the necessary data on the respective processors before it can initiate any child processes. The parent and child use an event mechanism for interprocess communication and can pass tasks over a dual queue. When a child is done, a message informs the parent[11]. If no processor is available, however, the child process must be started up on the same processor as the parent. Moreover, if there is only one way of obtaining a goal, the child will always be started on the same processor node as the parent, since there are no alternatives which can be started up in parallel.

The possibility of executing several alternative or independent subgoals simultaneously, can speed up the system considerably compared to executing it on a uniprocessor. How much faster it will actually run, depends on how well parallelism can be exploited in each particular case.

Parallelism in Programs: In addition to parallelism in the control system, inherent parallelism can be exploited in programs such as low level image processing. These programs deal with image data which requires extensive and time consuming operations. Implementing such programs has no complex control aspects because the processing is homogeneous, enabling the processors to run the same code on different

data. This implies that the Uniform System is the best programming approach. One example is edge detection. In this case, the data (the image) can be split into several "chunks" and put onto the available processors, which all run the same edge detection program on their part of the image[8] (a homogeneous problem). There is no complex control aspects involved, like starting up different programs on different processors and taking care of dual queues for tasks to be passed between the processes.

Processor Utilization: The two categories of parallelism in the hierarchical control system, discussed above, could use as many parallel processors as there are possible processes. However, there is a limit on the number of processors, 18 in our case, and therefore the problem of processor utilization arises. There has to be a balance between the number of processors the two categories are allowed to occupy. Obviously, the most time consuming processes should use the maximum number of processors, thus reducing the number of "bottle necks" in the system. Since programs such as low level image analysis will be the most expensive part with regard to execution time, it is preferable that these processes occupy most of the processors on the Butterfly, so as to prevent unecessary serial execution. The total execution time for achieving a goal will then be minimized. If the hierarchical control programs occupy just a few nodes, this will not hurt the overall performance significantly, since even the serial version of the control does not take much execution time.

3 Implementation

As the main program (in the appendix) shows, the control system first loads in all the information needed to obtain goals the user can ask about. It then allows the user to specify what s/he wants to do. The user interface permits the user to see what the knowledge base and data base contain, see what the syntax of a command is, store any newly acquired information into the knowledge and data bases, and exit the system. Other commands are just converted into upper case without any further action - they would be the interesting goals the system could obtain if it had been fully implemented.

To read from and write to files on the host machine (VAX 11/780), making use of a server process running on the host was necessary. The server process waits for the Butterfly to request access to a file using r_open, r_seek, r_read, r_write, and r_close. These are all routines which are analogous to the respective functions in UNIX. The Butterfly and the host machine had to access each other through specific ports, which had to be specified. If the ports were somehow out of function, they had to be "cleaned up" by resetting the machine before running the programs. Every time the program accessed a file on the host, it had to connect to the server running on the host before getting access, and disconnect

after the access.

A "catch - throw" statement wrapped around any code accessing files on the host machine or memory on the Butterfly was supposed to prevent crashing the machine totally, informing the user where the error occurred, and why, whenever an error was "caught". It did not prevent overwriting of memory though, which often happens if a pointer does not point where assumed. This often caused the system to crash because the operating system was overwritten due to errors in the programs.

The unimplemented part of the parallel version includes all the parallelism found in the hierarchical control system. It would require extensive use of Chrysalis, because it would not be possible to run uniform processes on all processors as the Uniform System functions require. The control system would have uniform processes only at the leaves in the control tree, where for example low level image processing would take place. At higher levels in the control tree the tasks would vary from node to node, requiring explicit control of each process and the memory it accesses. The low level programs would probably be the most time consuming ones (for example scene analysis), and should therefore occupy the largest proportion of nodes on the Butterfly in order to make the overall control system most time effective.

4 Implementation Difficulties

There were several obstacles when developing the code which slowed down the progress in implementing CAOS on the BBN Butterfly. The machine had little capability to prevent overwriting of system software, was time consuming to boot after crashing, and had incomplete/incorrect documentation making it hard to learn how to use the machine.

The lack of protection against overwriting the system portion of memory on the BBN Butterfly made debugging of the simplest programs very time consuming. If the code made an inappropriate memory reference, the machine would crash at once, usually because the ethernet connection was broken or the operating system was overwritten. The booting procedure could take from five to twenty-five minutes, and got very frustrating if the bugs were not found during the first try.

The documentation on the BBN Butterfly hardware was useful and quite easy to follow, giving a nice introduction on how the machine was built and how it worked. The documentation on how to get started with programming was not as nice, however. Out of date documents made introductory programming very hard to get through. As of this writing (March 1987) the documentation has been updated and local

conventions are added making it easier to work with the documentation.

5 Future Work

Since the current implementation of CAOS on the BBN Butterfly is only a partial implementation not really featuring any parallelism, the major part of the implementation is still to be done. Also, performance measurements of the complete parallel version compared to the serial versions need to be done when CAOS' parallel version is fully implemented.

The main aspect of completing the parallel version is to use the probability measures associated with each goal known to CAOS to figure out if two or more (sub)goals should be started up in parallel. If this is the case, the system needs to find idle processors and distribute the tasks onto these. Also, the problem of processor utilization has to be solved. The system needs to be able to identify if a goal contains a particularly time consuming subgoal and then preferably allow this subgoal to occupy a larger number of processors than a less time consuming subgoal.

The performance measurements need to include several types of problems, especially the two extremes; having lots of inherent parallelism in the programs at the lowest level in the hierarchically organized control tree, and having almost no inherent parallelism in the control tree. In addition, varying the number of running processors will be helpful in identifying how the parallel version is comparable to linear speedup.

6 The Butterfly

Most of today's computers are "uniprocessors" with only one central processor. This processor is connected to the computer memory via a single path which has a limited capacity, enabling operation on only one data element at a time. Now, however, there are several parallel processors on the market or under development. These computers, also called multiprocessors, have a collection of processors and many paths to memory. This allows manipulation of many data elements (one per processor) at the same instant in time, reducing the execution time of a program considerably. Ideally, a program run on a parallel computer with N processors should run N times faster than on a uniprocessor machine.

This part of the report will discuss "The Butterfly Multiprocessor", developed by Bolt, Beranek, and Newman (BBN) Laboratories, in general. Descriptions of both the Butterfly hardware configuration and possible programming approaches are included. In addition, the particular configuration of the 18 processor Butterfly at The University of Utah will be described, as it is at present (Winter 1987). Finally, two examples will illustrate how close to the ideal performance (N processors giving a factor of N speed up compared to a single processor) a 128 processor Butterfly comes.

7 The Butterfly Configuration

The Butterfly is accessed through a host machine, which typically is a SUN or a VAX. Furthermore, a separate input system (like the VICOM at the U of U) can be connected to the multiprocessor (a possible Butterfly setup is illustrated in Figure 1). This makes the machine well suited for manipulation of large amounts of data and real time computations. Computer Vision is a field which as a great need for computers like the Butterfly.

The Butterfly is a Multiple Instruction Multiple Data (MIMD) machine. This means the computer is able to execute several instructions at the same time, each instruction working on (possibly) different data. In contrast, a single processor machine can only execute one instruction at a time, on a single data element. Therefore, a multiprocessor, which can execute several instructions at once, is much faster than a uniprocessor which only can execute the same instructions sequentially.

All memory in a Butterfly is accessible by all processors (so called "shared memory" configuration). This allows every processor to access the same data, without having one copy of each piece of data per processor. Physically, memory is divided into N parts where N is the number of processors. Each processor has a local "chunk" of the total memory, but all other processors can access it.

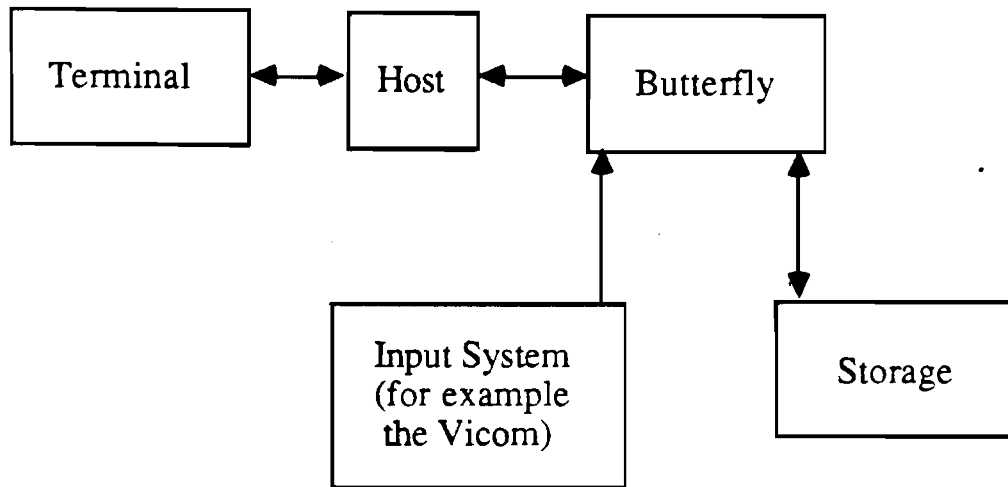

Figure 1: A Possible Butterfly Set-up.

The reason a Butterfly can execute many instructions at once is that it has several processors. However, there has to be interconnections between the processors also, to allow communication among them; Often, one processor needs the results of another processor's activity to be able to continue its own work. Last, but maybe most important, there must be a connection between each processor and the computer memory, so data can be accessed and stored.

In the Butterfly, a connection system called "The Butterfly Switch" provides the necessary communication link among any two processors and between any processor and memory.

In addition to the processors and the switching system, the Butterfly has some other components, which allow access to I/O devices and Multibus compatible devices. Finally, there is a global clock source for the entire multiprocessor. A listing of the Butterfly components follows below.

1. Processor nodes
2. MSI Switch nodes or VLSI Switch nodes
3. Butterfly Clock
4. Butterfly I/O boards
5. Multibus Adapter

I will now describe the processor node and the Butterfly Switch quite thoroughly, accompanied by a more brief description of the Butterfly clock, I/O boards, and Multibus adapter.

7.1 The Processor Node

The processors in the Butterfly do not reside on a board by themselves. Instead, each processor is the heart of a "processor node" card containing the elements listed below.

1. A (main) processor
2. Main memory
3. A co-processor (Processor Node Controller or PNC)
4. Memory management hardware (Memory Management Unit or MMU)
5. I/O bus (Butterfly Input Output link or BIOlink)
6. Interface to the Butterfly Switch

Figure 2 shows the organization of a processor node.

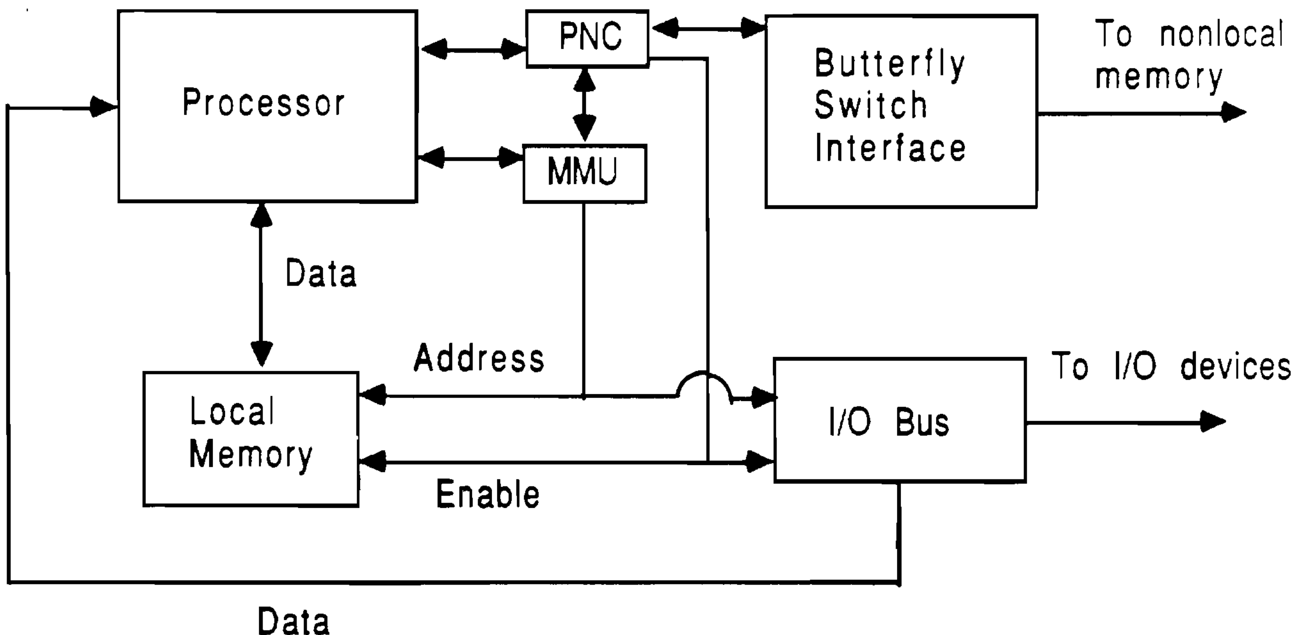

Figure 2: Processor Node Components and Main Interconnections.

Each processor runs a copy of the operating system, Chrysalis. Application programs also run on the processors. The local memory, physically on the board, is 1 Mega byte (Mb). By attaching daughter boards, it can be expanded to 4Mb. The MMU's responsibility consists of translating 24 bit virtual

memory addresses, which the processor generates, into 32 bit physical addresses in the memory of the Butterfly.

The processor can make both local and non-local memory references. It is the responsibility of the PNC to determine if any reference is non-local. If it is, the PNC sends a request through the Butterfly Switch to the correct processor (the one that has the requested memory physically on its board), asking for access to the memory address indicated by the requesting processor. The PNC then handles all transfers. If it is a local reference, however, the MMU just translates the virtual address to the corresponding physical address in local memory.

BIOlink (the I/O bus) has two modes of operation. It can either communicate directly with up to 4 I/O boards (per processor node), or it can communicate with Multibus devices through the Multibus Adapter. A processor node can only access its local I/O boards. No reference to another processor node's I/O boards is possible.

7.2 The Butterfly Switch

The Butterfly Switch is the communication link of the Butterfly. It provides communication only - no processing is done by the switch. It connects all the processors in the Butterfly to each other and to memory. It is responsible for the flow control and addressing, which is done as described below. The switch nodes, constituting the Butterfly Switch, are interconnected in a scheme resembling the Fast Fourier Transform (FFT). The number of inputs to a switch node (also called "switch element" or just "switch") corresponds to the "base" in a FFT. Two inputs therefore corresponds to a base of 2.

When a data package is to be sent through the Butterfly Switch to an address, a special routing scheme is used. For example, when it arrives at a switch node with two inputs (from two processors) and two outputs (to memory - see Figure 3), a packet of data might appear as shown. The switch node looks at the first bit and routes the rest of the data according to its value (0 - up, 1 - down). If the Butterfly Switch has switch nodes with four inputs and four outputs, the two first bits of the data package are used for routing. If we started with an incoming package of six bits, only four will come out at the indicated output (Figure 4). The P's in the figures refer to "processor" whereas the M's refer to "memory".

The number of routing bits a data package needs will increase according to the number of switch elements it has to go through. Figure 7 shows the routing of a data package through a Butterfly Switch which supports sixteen processors. The switch nodes has two inputs which means we need only one

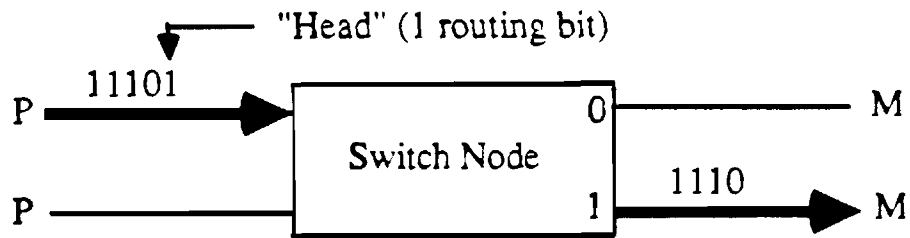

Figure 3: Routing of Data Through a 2-input Switch Node. The data package arrives serially, the "head" being the routing bit. It is stripped off, and the package is routed down (1 - down, 0 - up).

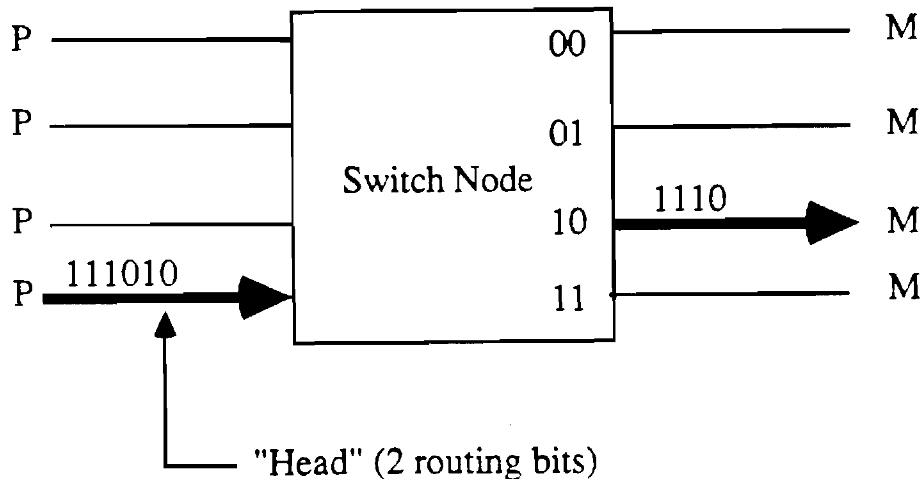

Figure 4: Routing of Data Through 4-input Switch Node. Two routing bits are needed, since there are 4 possible outputs.

routing bit per node. Since a path from any destination to any source contains four switch nodes, four routing bits must be included in the data package. In this scheme it does not matter which source the data came form. It will always end up at the same destination (if it has the same routing bits). All the bits in a data package arrives serially at one of the inputs of a switch element. The routing bits make up the "head" of the package.

The Butterfly Switch will not have some switch nodes with two inputs and some with four inputs. All the switch nodes in a particular Butterfly Switch will have the same number of inputs and outputs.

The FFT connection scheme used in the Butterfly Switch gives the result that there is not a unique path from every source to every destination (as in a fully connected graph) in the Butterfly. Path access conflicts can arise. Two messages might want to use the same path at the same time. If the head of a data package arrives at one input of a switch node, and the output port indicated by the routing bits is not available, there is a conflict (the output port is used by another data package, which is about to leave the switch node). This is resolved by rejecting the last arriving data package, and sending a signal back to the processor which sent it indicating that the processor has to try again. See Figure 5 for an illustration of this.

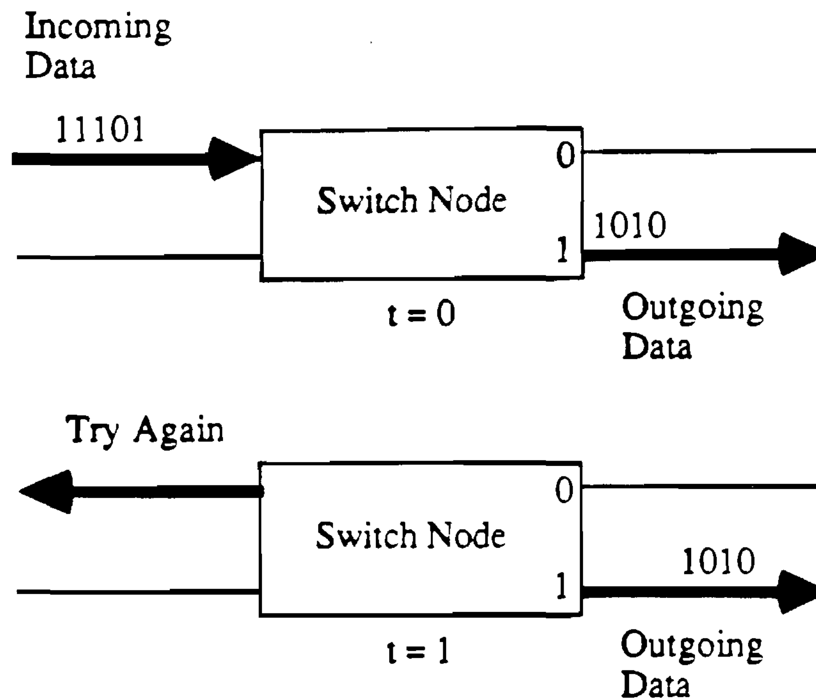

Figure 5: Resolving Path Contention. If a message arrives at a switch node, and the indicated output is busy, it is rejected. A message is sent back to the processor, indicating it has to try again.

The Fast Fourier Transform organization of the nodes in the Butterfly Switch results in a particular appearance of the switch node's arrangement. They are organized in $\log_B(N)$ columns. N is the number of processors, B is the number of inputs to a switch node (or equivalently the FFT base - in the MSI switch node, describe shortly, it is four). The total number of switch nodes are $(N/B)*\log_B(N)$. This can easily be checked with an example of, say, N = 4. This example, and another with N = 16, are illustrated in Figures 6 and 7. Both examples use switch elements with two inputs and outputs (B = 2).

So far, all the illustrations of the Butterfly Switch give the impression that a processor has to go through

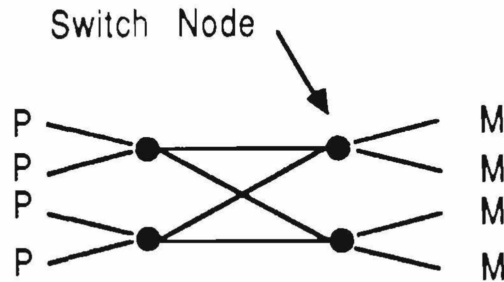

Figure 6: Switch System for 4 Processors. B = 2, N = 4, $(4/2)*\log_2 4$ = 4 switch nodes, $\log_2 4$ = 2 columns of switch nodes.

the switching network to access any part of memory. This is not true. Each processor has its own local memory which it can access without going through the Butterfly Switch. Essentially, this is obtained by "wrapping" the inputs of the leftmost column around, so they become adjacent to the outputs of the rightmost column. The network looks (conceptually) like a cylinder where processors and their respective local memories are connected at the adjacency. In real life, the processor and its local memory actually reside on the same physical board, as described in section 7.1. In addition to ensure direct access to local memory, the cylinder configuration allows bidirectional transfers through the Butterfly Switch. The cylinder concept is illustrated in Figure 8.

BBN Laboratories implemented two switch node cards: the Butterfly MSI Switch Node card (BSN) and the Butterfly VLSI Switch Node card (BVSN). The BVSN is the equivalent of eight BSNs (has eight switch nodes on one board). The BSNs are only used in small Butterfly systems.

Each switch node is a four-input, four-output crossbar switch. Its functions include routing of data, distribution of a clock signal, and distribution of a systemwide RESET.

7.3 The Butterfly Clock

The Butterfly is a fully synchronous machine at the bottom level. This requires a global clock source. The Butterfly Clock (BCLK) distributes the signal form this source (a crystal oscillator). The BLCK card is able to distribute the signal to a maximum of eight BSNs or BVSNs. This means that the BCLK has to operate in two modes. If there are eight or less switch cards in the system, it operates in a "master mode", just distributing the signal directly to the switch cards. If there are more than eight switch cards, however, another scheme is required. The clock signal is generated in a "root" BCLK (operating in master mode) and distributed through a tree of other BCLK boards operating in slave mode.

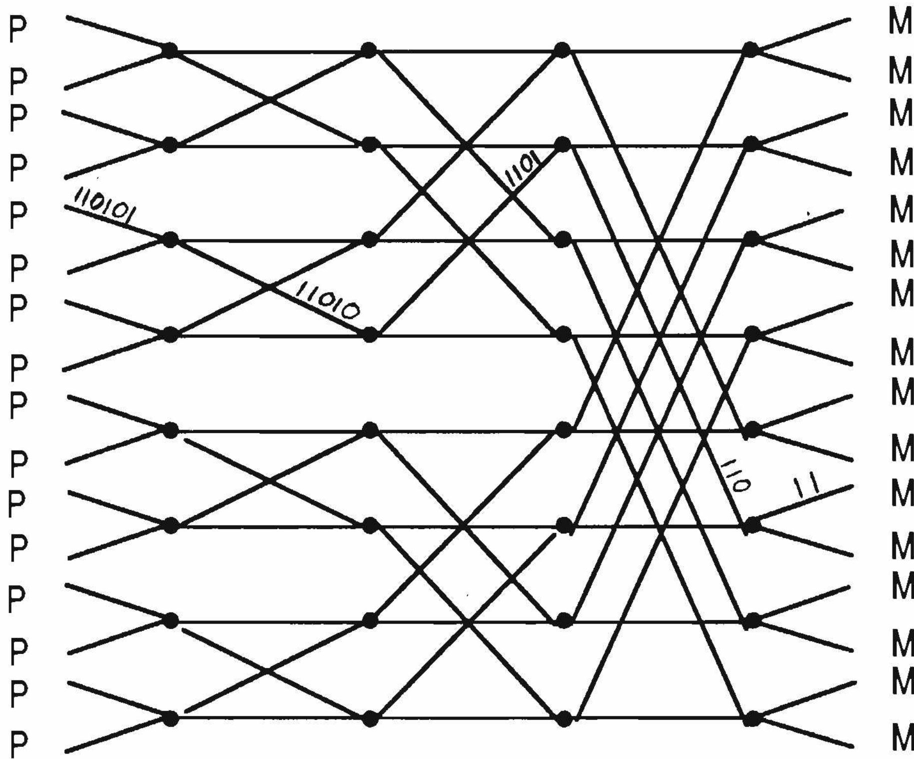

Figure 7: Switch System For 16 Processors. B = 2, N = 16, $(16/2)*\log_2 16 = 32$ switch nodes, $\log_2 16 = 4$ columns of switch nodes. The routing of a data package is also shown. To go from a processor to a memory, four switch nodes are passed. Therefore, there are 4 bits used for routing.

7.4 The Butterfly I/O Card

This card, also called the BI1, contains four character-asynchronous I/O channels and four synchronous I/O channels. It also contains an interface to BIOlink (I/O bus between BI1 and a processor node).

The BI1 uses a direct memory access mechanism for transferring data between synchronous I/O channels and the main memory of a processor node.

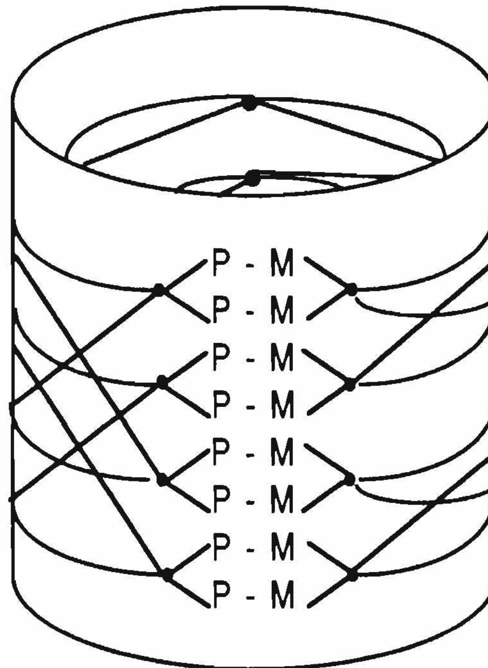

Figure 8: "Cylinder" Model of the Butterfly Switch.

7.5 The Multibus Adapter

Via the Butterfly Multibus Adapter (BMA), the processor nodes are allowed to communicate with Multibus I/O and memory devices. Multibus devices can access local memory of the processor node it is attached to (only). Devices attached to BIOlink can, in turn, access devices on the Multibus.

8 The Butterfly at The University of Utah

The local Butterfly is an eighteen processor configuration (with one additional node as backup). The processors are Motorola MC68020s and the co-processors are Motorola MC68881s. A frequency doubler brings the clock rate up to 16MHz (on the PNC boards only). Each node has 1Mb of memory, except for two, which have 4Mb each. Chrysalis, version 2.3.1, is the operating system currently being used. It supports the floating point abilities of the co-processors.

Due to conflicting information, it is hard to say if Figure 9 is absolutely correct for the local Butterfly. The machine has two boards containing processors and needs to have three BVSN cards to accommodate what Figure 9 illustrates. The current configuration allows two paths from a PNC to any part of memory

("odd switch configuration"), and the switching network can support up to 32 processors. Figure 9 should be correct for this maximal number of processors if P's and M's were added on the bottom half of the drawing. Then 24 switch nodes would be organized in 3 columns of 8 each (giving three BVSN card, each having eight switch nodes).

9 Technical Details

This section gives a short summary of some of the important technical details of the Butterfly configuration.

-The bandwidth in the Butterfly Switch is 32 Mbits/sec per switch path.

-The I/O bus (BIOlink) has a peak transfer rate of 16 Mbits/sec.

-The asynchronous RS-232 channels on BI1 transfer up to 38.4 Kb/sec each.

-The synchronous RS-422 channels on BI1 transfer up to 2 Mb/sec each.

-The asynchronous RS-232 channels on BMA transfers up to 38.4 Kb/sec each.

-The clock frequency is 8 MHz.

-The processors and co-processors operate at 16 MHz.

10 Butterfly Performance

In an experiment to test the performance of a 128 processor Butterfly, BBN Laboratories found that all the processors could be effectively used to obtain close to linear speed up. They tested the machine on matrix multiplication and Gaussian elimination. In Figure 10, the results of running matrix multiplication on different configurations form 1 to 128 processors are shown. Figure 11 shows the results from running the Gaussian elimination. Both figures show curves that are normalized to the time a single processor would use to complete the task (number of processors used plotted against the equivalent number of effective processors).

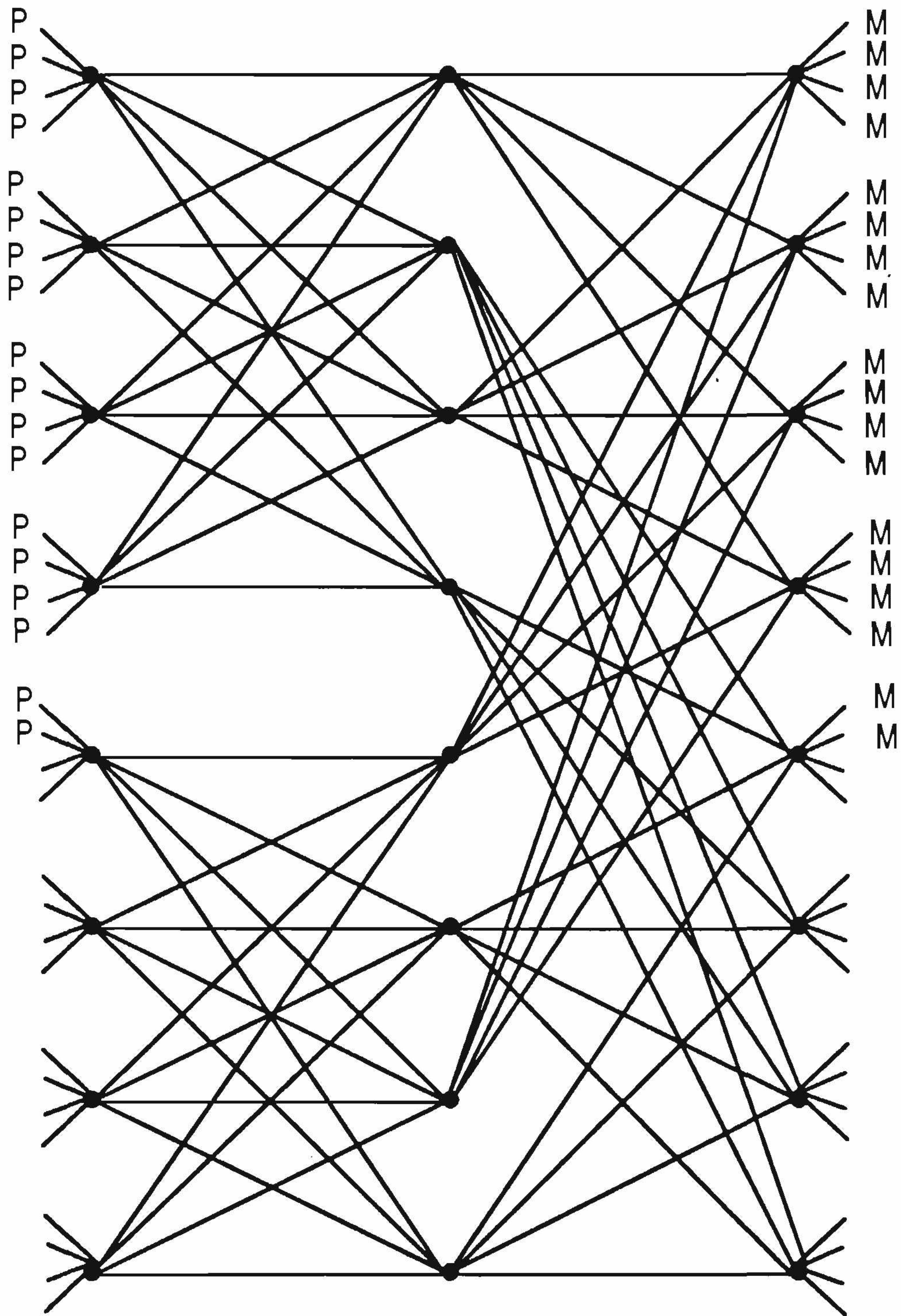

Figure 9: The Switch Network of the U of U Butterfly.

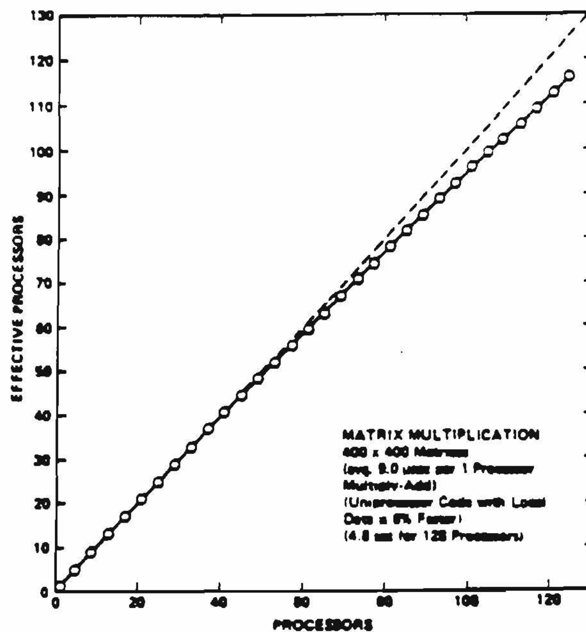

Figure 10: Results from Matrix Multiply. 400x400 matrices. Avg. 9.0 microsec. per 1 processor Multiply-Add. Uniprocessor code with local data is 6% faster. 4.8 sec. for 128 processors.

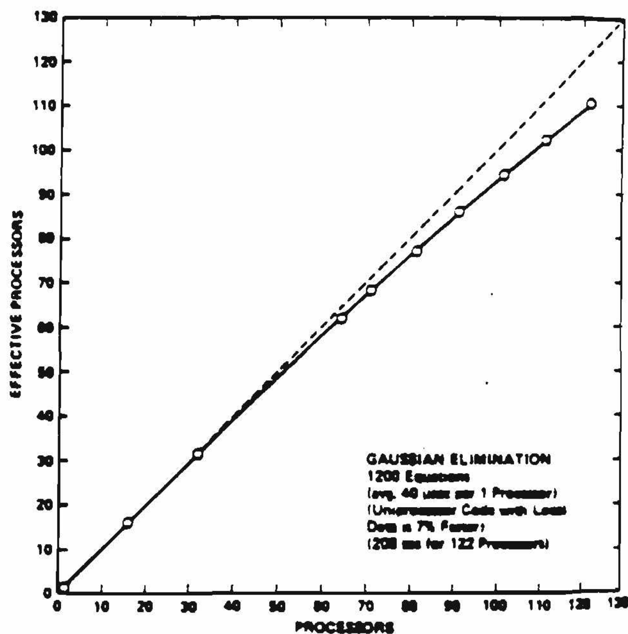

Figure 11: Results from Gaussian Elimination. 1200 equations. Avg. 40 microsec. per 1 processor. Uniprocessor code with local data is 7% faster. 206 sec. for 122 processors.

11 Chrysalis - The Operating System

Chrysalis is a real time, object oriented operating system. It provides configuration management, as well as management of virtual and physical address space. "Events" allow synchronization between processes. Furthermore, dual queues enable locking and passing of tasks between processes.

A collection of functions enable the user to explicitly manage the Butterfly resources (processors and memory) herself/himself. Each processor runs its own copy of Chrysalis. When the main program (being the "parent" process) is executed, it generates "child processes" (as specified by the user), each on a different node. The dual queues mentioned above allows the children and parent to lock and pass tasks. "Events allow interprocess communication. For example, a child can leave a message on the queue when it terminates. The user must ensure that the parent waits for all children to terminate before it terminates itself. Otherwise, the children will be terminated when the parent terminates, possibly leaving some of the tasks of the children uncompleted.

Even though Chrysalis easily supports development of application programs, it does not provide any automatic allocation of resources (processors). The user must specify herself/himself where and when a task is to be run, and where the data to be used can be found. It can be hard to correctly estimate the execution time of particular programs, thereby making it hard to use all the processors effectively.

12 The Uniform System Approach

In contrast to Chrysalis, the Uniform System approach (Us) emphases the computational tasks. It is built on top of Chrysalis and hides some of the details of resource management. A Uniform System library contains several subroutines which take care of, for example, allocation of memory and processors, and generation of new tasks. The user does not allocate memory space or processors explicitly. Us takes care of the distribution of tasks on processors and provides special memory allocation routines.

Memory management is simplified by the Us' view of memory: a large, shared memory, instead of a collection of small, local memories. Library functions allow scattering of data all over the globally shared memory, which reduces the frequency of memory contention.

Us treats all processors as identical workers. When an application program is to be executed, Us generates a copy of this program on every processor node. This enables every processor to do any task in the application program.

When the user writes an application program, there are two things s/he must include. First, the sub-

routines that make up what the program is going to do. Second, a special routine called the "generator" has to be included (there can, of course, be several generator routines, which is common). When a program is to be executed, the controlling part of the program (usually called "main") calls a "generator activator" routine (called UsDo). Its main arguments are one of the ordinary subroutines, a data structure, a generator, and a range. The subroutine is going to be executed on data in the given data structure over the specified range. The generator activator builds a task description according to the specifications of the generator. It then gives each processor a copy of this task descriptor and thereby activates them. When the task is finished, control is resumed to "main".

13 References

1. Anonymous, *Butterfly Multiprocessor*, BBN Laboratories, October 1986.

2. Anonymous, *Development of a Butterfly Multiprocessor Test Bed, The Butterfly Switch*, BBN Laboratories Incorporated, Report No. 5874, October 1985.

3. Anonymous, *Development of a Butterfly Multiprocessor Test Bed, Description of Butterfly Components*, BBN Laboratories Incorporated, Report No. 5872, March 1986.

4. D. Barry, W. Crowther, *The Butterfly Multiprocessor*, BBN Laboratories, May 1984.

5. M. Beeler, *Butterfly Parallel Processor Tutorial (for the C language)*, BBN Laboratories, November 8, 1985.

6. C.M. Brown, C.S. Ellis, J.A. Feldman, T.J. LeBlanc, and G.L. Peterson, *Research with the Butterfly Multicomputer*, Technical Report, BBN Laboratories Incorporated, 1986.

7. S.L. Chiu et. al, *Sensor Data Fusion on a Parallel Processor*, IEEE International Conference on Robotics and Automation, April 7-10, 1986, pp 1629 - 1633.

8. W. Crowther, *The Uniform System Approach to Programming the Butterfly Parallel Processor, Draft Version 1*, BBN Laboratories, November 14, 1985.

9. W. Crowther et. al, *Performance Measurements on a 128-node Butterfly Parallel Processor*, IEEE, 1985, pp 531 - 539.

10. R.M. Keller, and F.C.H. Lin, *Simulated Performance of a Reduction-Based Multiprocessor*, Computer, July 1984, pp 70 - 82.

11. W. Milliken, *Chrysalis Programmers Manual, Version 2.2.5*, BBN Laboratories, December 20 1985.

12. B. Thomas, R. Gurwitz, J. Goodhue, D. Allen, and M. Beeler, *Butterfly Parallel Processor Overview*, BBN Laboratories Incorporated, Report No. 6148, March 1986.

13. N. Thune, and B. Bhanu, *CAOS - An Approach to Robot Control*, Computer Science Dept., University of Utah, Technical Report No. UUCS-87-007, 31 March 1987.

14 Appendix

On the next pages follows a listing of the code written for the Butterfly. It runs, and takes care of input/output and setup of all necessary information in memory. It also includes examples of the knowledge base, data base, and the makefile used.

After the code for the partial implementation, the pseudo code and code for Hough's Transform follows (this code was never debugged entirely).

```
/**************************************************************************

Procedure Name   :main()
Part of          :CAOS - Control using Action Oriented Schemas
File Name        :main.c
Date             :July 18, 1986
Author           :Mari Thune

Purpose          :Main program of CAOS

**************************************************************************/

#include "defs.h"

Kelem   *Know_Table[27];

main()
{
 goalinfo *Goal_Information;             /* pointer to goal information */
 char           Main_Goal[COMMAND_SIZE],/* main goal from user */
                KnowBase[50],           /* Hold names of the two databases */
                ProgBase[50];

/*------------------------------------------------------------------------
 * Startup Section
 */

 PRINT_HEADING;

 printf("\nPlease specify the following : ");
 GET_FILE_NAME("Knowledge Base :", KnowBase);
 GET_FILE_NAME("Program Base :", ProgBase);
 printf("\n Thank's...\n");

 Load_Information(KnowBase,ProgBase);

/*------------------------------------------------------------------------
 *  Main Goal Achievement Section
 */

 for (;;)
 {
        User_Interface(Main_Goal,KnowBase,ProgBase);
/*      Goal_Information = Get_Goal_Information(Main_Goal);
        Assign_Inputs(Main_Goal,Goal_Information);
        Achieve_Goal(Main_Goal,Goal_Information);
        Print_Goal_Result(Main_Goal,Goal_Information); */
 }
}
```

```
/*************************************************************************

File Name       :defs.h
Date            :July 18, 1986
Author          :Mari Thune

Purpose         :Define all constants and include all needed files.

*************************************************************************/
#ifndef DEFSH
#define DEFSH


/* All needed include files */

#include "public.h"             /* Butterfly definitions */
#include "stdio.h"              /* Input/Output definitions */
#include "math.h"               /* Math definitions */
#include "macros.h"             /* Macro definitions */
#include "struct.h"             /* Structure definitions */
#include "func.h"               /* Function declarations */
#include "args.h"               /* Something with Butterfly */
#include "strings.h"            /* String definitions */
#include "ctype.h"              /* String and char. handling functions */
/* #include "match.h" */        /* Don't know what this is */
#include "chrys.h"              /* Chrysalis definitions */
#include <errno.h>              /* Don't know exactly what these are */
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include "rfs.h"                /* File server definitions */


/* Some constants needed for Butterfly stuff */

#define PORT 0
#define VAXADDR "128.110.4.21"          /* Net address of VAX */
#define BUTTERADDR "128.110.4.118"      /* Net address of BUTTERLFY */


/* Constants used in CAOS */

#define COMMAND_SIZE    100     /* Length of command size */
#define DONE            1       /* Is schema DONE ? */
#define GO              2       /* Schema is not DONE */
#define LEARN           3       /* Schema should learn from user */
#define QUIT            4       /* Quit trying to obtain a goal */
#define NOTOUTOK        -1


/* Some constants used when loading/writing data bases  */

#define ARG             '~'     /* Command ARGument     */
#define OUTP            ':'     /* OUTPut follows       */
#define EOC             '!'     /* End Of Command       */


#endif
```

```
/******************************************************************************

File Name       :func.h
Date            :August 1, 1986
Author          :Mari Thune

Purpose         :Define all the functions that CAOS uses.

******************************************************************************/
#ifndef FUNCH
#define FUNCH


void    Load_Information();
void    Make_Knowledge_Base();
void    Make_Program_Base();
void    Insert();
void    Store_Arguments();
void    User_Interface();
void    Store_Information();
void    Print_Know_Table();
void    Print_File();
char    *Load_File();
int     atohaddr();

#endif
```

```
/***********************************************************************

File Name        : struct.h
Date             : July 18, 1986
Author           : Mari Thune

Purpose          : To define all the structures for the CAOS system.

***********************************************************************/
#ifndef STRUCTH
#define STRUCTH


/*typedef      char    boolean;*/               /* Use for TRUE/FALSE values */

/*--------------------------------------------------------------------*/

typedef struct _kelem
{
        struct  _kelem  *Next;          /* Next goal in bucket */
        int             N;              /* # of trials */
        int             S;              /* # of successes */
        char            *Goal;          /* Name of current goal */
        struct goal     *Goal_Info;     /* Goal info of current goal */
} Kelem;        /* Element in hash table */

/*--------------------------------------------------------------------*/

typedef struct _delem
{
        struct  _delem  *Next;          /* Pointer to next data in bucket */
        char            *Data;          /* The actual data */
        int             Type;           /* Data type */
} Delem;        /* Element in hash table */

/*--------------------------------------------------------------------*/

typedef struct inp
{
        char            *Inp_Name1;     /* Name of input */
        char            *Inp_Name;      /* Name of input */
        char            *Value;         /* Actual input */
        struct inp      *Next;          /* Pointer to next needed input */
} input;        /* Holds input information for a goal */

/*--------------------------------------------------------------------*/

typedef struct outp
{
        char            *Out_Name1;     /* Name of output */
        char            *Out_Name;      /* Name of output */
        char            *Value;         /* Actual output */
        struct outp     *Next;          /* Pointer to next needed output */
} output;       /* Holds output information for a goal */

/*--------------------------------------------------------------------*/

typedef struct subg
{
        char            *SubGoal;       /* Holding the subgoal 'command' */
```

```
        int             Group;          /* Group # */
        struct goal     *Goal_Info;     /* Pointer to subgoal information */
        struct subg     *Next;          /* Pointer to next subgoal */
}subgoal;       /* Subgoal information about goal achievement */

/*-------------------------------------------------------------------*/

typedef struct goal
{
        boolean Null;                   /* If goal info was found or not */
        int     ProgNo;                 /* If program, this identifies it */
        boolean Done;                   /* Is the subgoal done ? */
        input   *Needed_Input;          /* Input needed to achieve goal */
        output  *Needed_Output;         /* Ouput needed to achieve goal */
        char    *Output;                /* Output string 8/
        boolean OutOK;                  /* Indicates if output is OK */
        subgoal *Subgoals;              /* Subgoals if goal is not program */
        int     SubgNo;                 /* Number of subgoals for the goal */
}goalinfo;      /* Holds information about goal achievements */

/*-------------------------------------------------------------------*/

#endif STRUCTH
```

```
/****************************************************************************

File Name       :macros.h
Date            :July 18, 1986
Author          :Mari Thune

Purpose         :Define macros for the CAOS system.

****************************************************************************/
#ifndef MACROSH
#define MACROSH


/**************************************************************************/
/*                 Macro to print main heading                            */
/**************************************************************************/

#define PRINT_HEADING   { printf("\t\tHierarchical Robot Control System\n");\
                          printf("\t\t\tRelease 1.30\n\n\n\n\n");}


/**************************************************************************/
/*                 Macro to allocate more memory                          */
/**************************************************************************/

#define MORE_MEMORY(num,type)   (type *)(calloc(num,sizeof(type)))

/**************************************************************************/
/*                 Macro to read in a file name                           */
/**************************************************************************/

#define GET_FILE_NAME(text,file)        { printf("\n%s",text);\
                                          scanf("%s",file);\
                                          printf("\n");}

/**************************************************************************/
/*                 Macro to print error messages                          */
/**************************************************************************/

#define ERROR_MSG(s1,s2)          { printf("\n\n%s %s\n\n",s1,s2);\
                                    printf("Error - Can't continue....\n\n");\
                                    exit(-1);}


/**************************************************************************/
/*                 Macro to close a file gracefully                       */
/**************************************************************************/

#define DISCONNECT(fd)  { r_close(fd);\
                          printf("\nDisconnecting - be patient...\n");\
                          disconnect_from_server();\
                          printf("\n");}

/**************************************************************************/
/*                 Macro to catch errors and report to caller             */
/**************************************************************************/


#define PROTECT(code,file,message)\
                 {      catch\
```

```
                                code;\
                        onthrow\
                                when(TRUE)\
                                  printf("\n%s:  %s\n", file, message);\
                        endcatch };


/***************************************************************************/
/*      Macro that initializes an array to the 'value' given              */
/***************************************************************************/

#define INITIALIZE(no,value,array) {int i;\
                                    for (i = 0; i < no; i++)\
                                        array[i] = value; }

/***************************************************************************/
/*      Macro that prints out menu                                         */
/***************************************************************************/

#define PRINT_MENU      {printf("\n\n\t\tM E N U\n\n");\
                         printf("1...Print Knowledge and Program files\n");\
                         printf("2...Print Knowledge Base\n");\
                         printf("4...Print Command Syntax\n");\
                         printf("5...Exit without storing anything\n");\
                         printf("9...Exit Control System\n"); }


/***************************************************************************/
/*      Macro that prints the command syntax                               */
/***************************************************************************/

#define PRINT_SYNTAX    printf("\nSyntax: letters~argument~:letters~argument~!\n")


/***************************************************************************/
/*      Macro which converts characters in a string to upper case          */
/***************************************************************************/

#define TOUPPER(array)  {int i = 0;\
                         while (array[i] != '\0') {\
                         if (array[i] == ARG) while (array[++i] != ARG) ;\
                         else if (islower(array[i])) array[i] = toupper(array[i]);\
                         ++i; } }


#endif MACROSH
```

```
/***************************************************************************

Procedure Name   :Load_Information()
Part of          :CAOS
File Name        :li.d
Date             :July 18, 1986
Author           :Mari Thune

Purpose          :Read information in Knowledge base, Program base,
                  Data base and Probability base into memory.

***************************************************************************/

#include "defs.h"

void Load_Information(KnowBase,ProgBase)
char    *KnowBase,      /* Hold names of the four databases */
        *ProgBase;
{
 char                   *string;        /* Holds the file read in */
 extern  Kelem          *Know_Table[27];

 INITIALIZE(27,NULL,Know_Table);

 string = Load_File(KnowBase);          /* Get info in Knowledge Base */
 Make_Knowledge_Base(string);           /* Store Knowledge Base in memory */

 string = Load_File(ProgBase);          /* Get info in Program Base */
 Make_Program_Base(string);             /* Store Program Base in memory */
}
```

```
/*****************************************************************************

Procedure Name  :Load_File()
Part of         :CAOS
File Name       :lf.c
Date            :July 18, 1986
Author          :Mari Thune

Purpose         :To read in contents of a knowledge base file.

*****************************************************************************/

#include "defs.h"

char    *Load_File(Base)
char    *Base;
{
int     fd;
int     size;
int     i,j;
char    *string;

catch
{
        if (connect_to_server(atohaddr("128.110.4.21"),PORT) < 0) /* Connect */
                ERROR_MSG("li.c: Can't connect to server","");

        if((fd = r_open(Base,0)) < 0)           /* Can we open the file ? */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: Can't open KB or PB file for reading !","");
        }
        if((size = (int)r_lseek(fd,0L,2)) < 1)          /* Find size of file */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Nothing in KB or PB file !");
        }
        if(r_lseek(fd,0L,0) != 0)       /* Rewind to beginning of file */
        {
                DISCONNECT(fd);
                ERROR_MSG("li.c: ","lseek didn't return to beginning of file");
        }
        if((string = MORE_MEMORY(size+1,char)) == NULL) /* Allocate for KB or PB */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Couldn't allocate space for KB or PB");
        }
        if(r_read(fd,string,size) < 0)          /* Read in contents */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Couldn't read from KB or PB file !");
        }
}
onthrow
        when(TRUE)
        {
                DISCONNECT(fd);   /* Any error -> disconnect from server */
                ERROR_MSG("lf.c: ","Error in filehandeling !");
        }
endcatch;
```

```
DISCONNECT(fd);          /* Filehandling finished, disconnect from server */

catch
{
        printf("\nHere's what I read:\n");
        printf("%s\n",string);
}
onthrow
        when(TRUE)
          printf("\nlf.c: Error when trying to print contents of string\n");
endcatch;

catch
{
        return(string);
}
onthrow
        when (TRUE)
          printf("\nlf.c: Error when trying to return string\n");
endcatch;
}
```

```
atohaddr (S)
char *S;
{
   char  Substr[20];
   char *ssp;
   int   temp = 0;
   int   i;

   for (i = 0; i < 4; i++) {
      ssp = Substr;
      while ((*S != '.') && (*S != '\0'))
         *ssp++ = *S++;
      S++;
      *ssp = '\0';
      temp = temp | ((atoi (Substr)) << ((3 - i) * 8));
   }
   return (temp);
}
```

```
/************************************************************************

Procedure Name  :Make_Knowledge_Base()
Part of         :CAOS
File Name       :mkb.c
Date            :August 1, 1986
Author          :Mari Thune

Purpose         :To get the knowledge base (for CAOS) into memory.

************************************************************************/
#include "defs.h"

void Make_Knowledge_Base(string)
char    *string;
{
char     com[COMMAND_SIZE+1],              /* Holds command */
         subcom[COMMAND_SIZE+1];           /* Holds subcommand */
int      no,Sno;                           /* Prognr, # of subgoals */
goalinfo *Goal_Info;                       /* Pointer to all info about goal */
subgoal  *Subg;                            /* Pointer to subgoal */
int      i,j,N,S;                                  /* Loop controls */

i = 0;

catch
{
        while(string[i] != '\0')          /* Get every command */
        {
         while (string[i] == 15) ++i;
         sscanf(&string[i],"%d%s!",&no,com);
         i = i + strlen(com);             /* Ensure correct offset for next scanning, */
         if(no <  10)  ++i;               /* when assuming # of subgoals < 100 */
         else   i += 2;                   /* and <return> at end of command */

         if((Goal_Info = MORE_MEMORY(1,goalinfo)) == NULL)
          ERROR_MSG("mkb.c", "Error when allocating Goal_Info");
         Goal_Info->SubgNo = no;

         if((Subg = MORE_MEMORY(1,subgoal)) == NULL)
                ERROR_MSG("mkb.c","Error when allocating Subg");
         Goal_Info->Subgoals = Subg;

         for(j = 0; j < no; ++j)          /* Get every subcommand */
         {
                while (string[i] == 15) ++i;
                sscanf(&string[i],"%d%s!",&Sno,subcom);
                i = i + strlen(subcom);          /* Ensure correct offset for next scannin
                if(Sno <  10)  i += 1;           /* when assuming # of subgoals < 100 */
                else    i += 2;                  /* and <return> at end of command */

                if((Subg->SubGoal = MORE_MEMORY(1+strlen(subcom),char)) == NULL)
                        ERROR_MSG("mkb.c","Error when allocating Sug->SubGoal");
                strcpy(Subg->SubGoal,subcom);
                Subg->Group = Sno;
                if(j < no-1)
                {
                        if((Subg->Next = MORE_MEMORY(1,subgoal)) == NULL)
                         ERROR_MSG("mkb.c","Error when allocatingSubg->Next");
                        Subg = Subg->Next;
```

```
                }
          }
          while (string[i] == 15) ++i;
          sscanf(&string[i],"%d%d",&N,&S);

          if (N < 10) ++i;
          else if (N < 100) i += 2;
          else i += 3;

          if (S < 10) ++i;
          else if (S < 100) i += 2;
          else i += 3;

          Insert(com,Goal_Info,N,S);
         }
        PROTECT(cfree(string), "mkb.c", "Error when freeing string");
}
onthrow
        when(TRUE)
        printf("\nmkb.c: Something happened when storing Knowledge Base\n");
endcatch;
}
```

```
/************************************************************************

 Procedure Name   : Make_Data_Base()
 Part of          : CAOS
 File Name        : mdb.c
 Date             : August 6, 1986
 Author           : Mari Thune

 Purpose          : To load the data base into memory

************************************************************************/

#include        "defs.h"                            /* My definitions */

void    Make_Data_Base(string)
char    *string;                                    /* Points to all data */
{
 char            data[COMMAND_SIZE+1];   /* Array holding 1 pice of data */
 int             Type;                   /* Program number */
 Delem           *delem;
 extern Delem    *Data_Table[27];
 int             i;                      /* Position control for reading of string */

 i = 0;
 while (string[i] != '\0')
 {
        sscanf(&string[i],"%d%s!",&Type,data);
        i = i + strlen(data) + 1;        /* Ensure correct pos. for next scanning*/
        if(Type < 10) i += 1;            /* assuming <return> at end of data line */
        else i += 2;                     /* and Program # < 100 */

        if((delem = MORE_MEMORY(1,Delem)) == NULL)
         ERROR_MSG("mdb.c","Error when allocating delem");
        delem->Next = Data_Table[data[0]-'A'];
        if((delem->Data = MORE_MEMORY(1+strlen(data),char)) == NULL)
         ERROR_MSG("mdb.c","Error when allocating delem->Data");
        if((strcpy(delem->Data,data)) == NULL)
         ERROR_MSG("mdb.c","Error when stringcopying to delem->Data");
        delem->Type = Type;
        Data_Table[data[0]-'A'] = delem;
 }
 PROTECT(cfree(string),"mdb.c","Error when freeing string");
}
```

```
/******************************************************************************

Procedure Name  : Make_Program_Base()
Part of         : CAOS
File Name       : mpb.c
Date            : August 7, 1986
Author          : Mari Thune

Purpose         : To load low level programs into the program. base

******************************************************************************/

#include        "defs.h"                /* My definitions */

void Make_Program_Base(string)
char    *string;                        /* String holding info */
{
 goalinfo       *Goal_Info;     /* Pointer to goalinfo */
 char   com[COMMAND_SIZE];      /* Character used to read file */
 int            Pno;            /* Program number */
 int            i;              /* Loop control */
 int            N,S;

 i = 0;

 while (string[i] != '\0')
 {
        while (string[i] == 15) ++i;
        sscanf(&string[i],"%d%s!%d%d",&Pno,com,&N,&S);
        i = i + strlen(com); /* Ensure correct position for scanning */
        if(Pno < 10) ++i;         /* assuming <return> at end of line and */
        else i += 2;              /* Pno < 100 */

        if (N < 10) ++i;
        else if (N < 100) i += 2;
        else i += 3;

        if (S < 10) ++i;
        else if (S < 100) i += 2;
        else i += 3;

        printf("%d %d\n",N,S);
        if((Goal_Info = MORE_MEMORY(1,goalinfo)) == NULL)
         ERROR_MSG("mpb.c","Error when allocating Goal_Info");
        Goal_Info->ProgNo = Pno;
        Insert(com,Goal_Info,N,S);
 }
 PROTECT(cfree(string),"mpb.c","Error when cfreeing string");
}
```

```
/*****************************************************************************

Procedure Name  : Insert()
Part of         : CAOS
File Name       : i.c
Date            : August 5, 1986
Author          : Mari Thune

Purpose         : To load in low level programs into the knowledge base

*****************************************************************************/

#include        "defs.h"                /* My definitions */

void Insert(com,Goal_Info,N,S)
char            com[];                  /* Command string */
goalinfo        *Goal_Info;             /* Pointer to goal information */
int             N,S;                    /* Prob */
{
 extern Kelem   *Know_Table[27];        /* Knowledge base */
 Kelem          *Curr_Elem;

 if((Curr_Elem = MORE_MEMORY(1,Kelem)) == NULL) ERROR_MSG("i.c: Error when allocating Curr_
 if((Curr_Elem->Goal = MORE_MEMORY(1+strlen(com),char)) == NULL) ERROR_MSG("i.c: Error whe
 if((strcpy(Curr_Elem->Goal,com)) == NULL) ERROR_MSG("i.c: Error when stingcopying","");

 Curr_Elem->N = N;
 Curr_Elem->S = S;
 Store_Arguments(com,Goal_Info);

 Curr_Elem->Goal_Info = Goal_Info;
 Curr_Elem->Next = Know_Table[com[0]-'A'];
 Know_Table[com[0]-'A'] = Curr_Elem;
}
```

```
/*****************************************************************************

Procedure Name   : Load_Data_Base()
Part of          : CAOS
File Name        : ldb.c
Date             : July 20, 1986
Author           : Mari Thune

Purpose          : To load the data base into memory

*****************************************************************************/

#include        "defs.h"                        /* my definitions */

void Load_Data_Base(fd)
FILE    *fd;                                    /* file descriptor for data file */
{
        char            data[COMMAND_SIZE+1];   /* character used to read file */
        int             Type;                   /* program number */
        delem           *Delem;
        extern  delem   *Data_Table[27];

        while (fscanf(fd,"%d%s!",&Type,data) != EOF)
        {
                Delem = MORE_MEMORY(1,delem);
                Delem->Next = Data_Table[data[0]-'A'];
                Delem->data = MORE_MEMORY(1+strlen(data),char);
                strcpy(Delem->data,data);
                Delem->Type = Type;
                Data_Table[data[0]-'A'] = Delem;
        }
}
```

```
/*****************************************************************************

Procedure Name  : Print_Data_Table()
Part of         : CAOS
File Name       : pdt.c
Date            : September 2, 1986
Author          : Mari Thune

Purpose         : To print the contents of the data base for the user

*****************************************************************************/

#include        "defs.h"                /* My definitions */

void Print_Data_Table()
{
        extern Delem    *Data_Table[27];
        Delem           *Curr_Elem;
        int             i;

        printf("\nData Base:\n\n");

        for (i = 0; i < 27; i++)
        {
                Curr_Elem = Data_Table[i];
                while (Curr_Elem != NULL)
                {
                        printf("%02d %s\n",Curr_Elem->Type,Curr_Elem->Data);
                        Curr_Elem = Curr_Elem->Next;
                }
        }
}
```

```
/*************************************************************************

 Procedure Name  : Print_File()
 Part of         : CAOS
 File Name       : pf.c
 Date            : Sepember 2, 1986
 Author          : Mari Thune

 Purpose         : To print the contents of a given data base file

*************************************************************************/

#include        "defs.h"        /* user defined goodies */

void Print_File(Base)
char    *Base;                  /* file name for knowledge base */
{
int     fd;
int     size;
int     i;
char    *string;

catch
{
        if (connect_to_server(atohaddr("128.110.4.21"),PORT) < 0) /* Connect */
                ERROR_MSG("li.c: ","Can't connect to server");
        if((fd = r_open(Base,0)) < 0)           /* Open the file */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Can't open KB or DB or PB file for reading !");
        }
        if((size = (int)r_lseek(fd,0L,2)) < 1)          /* Find size of file */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Nothing in KB or DB or PB file !");
        }
        if(r_lseek(fd,0L,0) != 0)       /* Rewind to beginning of file */
        {
                DISCONNECT(fd);
                ERROR_MSG("li.c: ","lseek didn't return to beginning of file");
        }
        if((string = MORE_MEMORY(size+1,char)) == NULL) /* Allocate for KB, DB or PB */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Couldn't allocate space for KB or DB or PB");
        }
        if((size = r_read(fd,string,size)) < 0)         /* Read in contents */
        {
                DISCONNECT(fd);
                ERROR_MSG("lf.c: ","Couldn't read from KB or DB or PB file !");
        }
}
onthrow
        when(TRUE)
        {
                DISCONNECT(fd);   /* Any error -> disconnect from server */
                ERROR_MSG("lf.c: ","Error in filehandeling !");
        }
endcatch;
```

```
DISCONNECT(fd);         /* Filehandling finished, disconnect from server */

catch
{
        printf("\nHere's what %s contains:\n",Base);
        i = 0;
        while(string[i] != '\0')
        {
                printf("%c",string[i]);
                if((string[i] == '!') || (string[i] == 015)) printf("\n");
                i++;
        }
}
onthrow
        when(TRUE)
          printf("\nlf.c: Error when trying to print contents of string\n");
endcatch;
}
```

```
/*****************************************************************************

Procedure Name  : Print_Know_Table()
Part of         : CAOS
File Name       : pkt.c
Date            : September 2, 1986
Author          : Mari Thune

Purpose         : To print the contents of the knowledge base for the user.

*****************************************************************************/

#include        "defs.h"                /* My definitions */

void Print_Know_Table()
{
        extern Kelem    *Know_Table[27];/* know. base */
        Kelem           *Curr_Elem;
        subgoal         *Subg;
        int             i;

        printf("\nKnowledge Base:\n\n");

catch
{
        for (i = 0; i < 27; i++)
        {
                Curr_Elem = Know_Table[i];
                while (Curr_Elem != NULL)
                {
                        printf("%s\n",Curr_Elem->Goal);
                        Subg = Curr_Elem->Goal_Info->Subgoals;
                        while (Subg != NULL)
                        {
                                printf("\t%02d %s\n",Subg->Group,Subg->SubGoal);
                                Subg = Subg->Next;
                        }
                        printf("Goal Prob = %f\n\n",Curr_Elem->S/(float)Curr_Elem->N);
                        Curr_Elem = Curr_Elem->Next;
                }
        }
}
onthrow
        when (TRUE)
                printf("ERROR: in pkt.c somewhere !!\n");
endcatch;
}
```

```
/******************************************************************************

 Procedure Name  : Store_Arguments()
 Part of         : CAOS
 File Name       : sa.c
 Date            : August 5, 1986
 Author          : Mari Thune

 Purpose         : To store the arguments of a goal

******************************************************************************/

#include        "defs.h"        /* My definitions */

void Store_Arguments(com,Goal_Info)
char            com[];
goalinfo        *Goal_Info;
{
 output         *Out;
 input          *Inp;
 char           Argument[100];
 int            no;
 int            i = -1;
 boolean        OUTPUT = FALSE;

 while (com[++i] != '\0' && com[i] != EOC)
 if (com[i] == OUTP)
 {
        OUTPUT = TRUE;
        if ((Goal_Info->Output = MORE_MEMORY(1+strlen(&com[i]),char)) == NULL)
                ERROR_MSG("sa.c: Error when allocating Goal_Info->Output","");
        if (strcpy(Goal_Info->Output,&com[i]) == NULL)
                ERROR_MSG("sa.c: Error when strcpy Goal_Info->Output","");
 }
 else if (com[i] == ARG)
 {
        no = 0;
        while (com[++i] != ARG)
        {
                if (com[i] == '\0') ERROR_MSG("sa.c:  Missing ~ when reading ", com);
                Argument[no++] = com[i];
        }
        Argument[no] = '\0';
        if (OUTPUT)
        {
                if (Goal_Info->Needed_Output == NULL)
                {
                        if((Out = MORE_MEMORY(1,output)) == NULL)
                         ERROR_MSG("sa.c","Error when allocating Out");
                        Goal_Info->Needed_Output = Out;
                }
                else
                {
                        if((Out->Next = MORE_MEMORY(1,output)) == NULL)
                         ERROR_MSG("sa.c","Error when allocating Out->Next");
                        Out = Out->Next;
                }
                if((Out->Out_Name = MORE_MEMORY(1+no,char)) == NULL)
                 ERROR_MSG("sa.c","Error when allocating Out->Out_Name");
                if((strcpy(Out->Out_Name,Argument)) == NULL)
```

```
                 ERROR_MSG("sa.c","Error when stringcopying to Out->Out_Name");
        }
        else
        {
                if (Goal_Info->Needed_Input == NULL)
                {
                        if((Inp = MORE_MEMORY(1,input)) == NULL)
                         ERROR_MSG("sa.c","Error when allocating Inp");
                        Goal_Info->Needed_Input = Inp;
                }
                else
                {
                        if((Inp->Next = MORE_MEMORY(1,input)) == NULL)
                         ERROR_MSG("sa.c","Error when allocating Inp->Next");
                        Inp = Inp->Next;
                }
                if((Inp->Inp_Name = MORE_MEMORY(1+no,char)) == NULL)
                 ERROR_MSG("sa.c","Error when allocating Inp->Inp_Name");
                if((strcpy(Inp->Inp_Name,Argument)) == NULL)
                 ERROR_MSG("sa.c","Error when stringcopying to Inp->Inp_Name");
        }
    }
}
```

```
/**************************************************************************

Procedure Name  : Store_Information()
Part of         : CAOS
File Name       : si.c
Date            : August 13, 1986
Author          : Mari Thune

Purpose         : To store all information

**************************************************************************/

#include        "defs.h"                /* My definitions */

void Store_Information(KnowBase,DataBase,ProbBase)
char    KnowBase[],
        DataBase[],
        ProbBase[];
{
 int    i,c;                            /* Loop controls */
 extern elem    *Know_Table[27];/* Knowledge base */
 elem           *Curr_Elem;
 subgoal        *Subg;
 FILE           *fd;            /* File descriptor */
 FILE           *fopen();       /* Open function   */

 printf("\nTrying to Store Knowledge Base . . . ");
 catch
 {
  if(connect_to_server(atohaddr("128.110.4.21"),PORT) < 0)      /* Connect */
        ERROR_MSG("si.c","Can't connect to server");
  printf("si.c: Connect to server - OK\n");

  if ((fd = r_open(KnowBase,1)) < 0 )           /* Open the file */
  {
        DISCONNECT(fd);
        ERROR_MSG("si.c","Can't open Knowledge Base");
  }
  printf("si.c: Knowledgebase opened\n");

  for (i = 0; i < 27; ++i)
  {
   Curr_Elem = Know_Table[i];
   while (Curr_Elem != NULL)
   {
        if (Curr_Elem->Goal_Info->ProgNo == 0)
        {
         sprintf(&string[c],"%d%s\n",Curr_Elem->Goal_Info->SubgNo,Curr_Elem->Goal);
         Subg = Curr_Elem->Goal_Info->SubGoals;
         while (Subg != NULL)
         {
                fprintf(fd,"%d%s\n",Subg->Group,Subg->SubGoal);
                Subg = Subg->Next;
         }
         fprintf(fd,"\n");
        }
        Curr_Elem = Curr_Elem->Next;
   }
 }
 fclose(fd);                            /* close file */
```

```
printf("Stored !\n");
}
```

```
/***************************************************************************

Procedure Name  : User_Interface()
Part of         : CAOS
File Name       : ui.c
Date            : August 12, 1986
Author          : Mari Thune

Purpose         : To get a command (main goal) from the user via the
                  keyboard

***************************************************************************/

#include        "defs.h"        /* My definitions */

void User_Interface(command,KnowBase,ProgBase)
char    command[],              /* Holds the comand string */
        KnowBase[],             /* Hold names of the knowledge bases */
        ProgBase[];
{
 int    i;                      /* Loop control */

 for (;;)
 {
        PRINT_MENU;
        printf("\nCommand: ");
        fflush(stdout);
        scanf("%s!",command);

        switch (command[0])
        {
                case '1':       Print_File(KnowBase);
                                Print_File(ProgBase);
                                break;

                case '2':       Print_Know_Table();
                                break;

                case '4':       PRINT_SYNTAX;
                                break;

                case '5':       printf("\nI am exiting now !\n");
                                exit(1);

/*              case '9':       Store_Information(KnowBase,ProgBase);
                                printf("\nBye bye . . .\n\n");
                                exit(1);*/

                default:        catch
                                TOUPPER(command);
                                onthrow
                                        while(TRUE)
                                         printf("\nui.c: Error when converting to upper case
                                endcatch;
                                printf("\n");
                                return;

        }
 }
}
```

MAKEFILE

```
BFLY    = /usr/butterfly
REL     = $(BFLY)/chrys/release
LOC     = $(BFLY)/local
US      = $(REL)/us
BIN     = $(BFLY)/bin
CFLAGS  = -c -O -I$(REL)
LFLAGS  =   -lm -g
CC68          =    $(BIN)/bfcc
CC68FLAGS     =    -c -O -DBFLY -h
LNK68LIBS     =    $(REL)/libtools.a

# Put your initials here
ID      = mt

CFLS    = pf.c ui.c main.c lf.c atohaddr.c li.c mkb.c i.c sa.c mkb.c\
          mpb.c pkt.c

OFLS    = pf.o ui.o main.o lf.o atohaddr.o li.o mkb.o i.o sa.o mkb.o\
          mpb.o pkt.o

O68FLS  = pf.o68 ui.o68 main.o68 lf.o68 atohaddr.o68 li.o68 mkb.o68 i.o68\
          sa.o68 mkb.o68 mpb.o68 pkt.o68

.SUFFIXES:
.SUFFIXES: .68 .o68 .o .c

.c.o68:
        $(CC68) $(CC68FLAGS) $*.c

default:
        @echo "You must specify target machine as 'make b' or 'make v'"

v:      $(CFLS) $(OFLS)
        cc -o caos $(OFLS) $(US)/us.o $(LFLAGS)

b:      $(CFLS) $(O68FLS)
        $(CC68) $(O68FLS) $(LNK68LIBS) -o caos.68
        cp caos.68 $(LOC)/public/$(ID)_caos.68
clean:
        rm *.o *.o68 *.68
```

KBASE

```
4CLEAR~OBJECT~!
1FIND_SPACE_FOR~OBJECT~:~X~~Y~~Z~!
1GRASP~OBJECT~AT~X~~Y~~Z~!
1MOVE~OBJECT~TO~X~~Y~~Z~!
1UNGRASP~OBJECT~:~STATUS~!
11 4

3GRASP~OBJECT~AT~X~~Y~~Z~!
1PRESHAPE_HAND_FOR~OBJECT~!
1MOVE_HAND_TO~OBJECT~AT~X~~Y~~Z~!
1GRIP~OBJECT~!
23 8

3GET~OBJECT~AT~X~~Y~~Z~!
1GRASP~OBJECT~AT~X~~Y~~Z~!
2CLEAR~OBJECT~!
2GRASP~OBJECT~AT~X~~Y~~Z~!
28 16

3MOVE~OBJECT1~FROM~X1~~Y1~~Z1~TO~OBJECT2~AT~X2~~Y2~~Z2~!
1MOVE~OBJECT1~TO~X2~~Y2~~Z2~!
2CLEAR~OBJECT2~!
2MOVE~OBJECT1~TO~X2~~Y2~~Z2~!
17 16

5PUT~OBJECT1~ON~OBJECT2~:OK~STATUS~!
1FIND~OBJECT1~:~X1~~Y1~~Z1~!
1FIND~OBJECT2~:~X2~~Y2~~Z2~!
1GET~OBJECT1~AT~X1~~Y1~~Z1~!
1MOVE~OBJECT1~FROM~X1~~Y1~~Z1~TO~OBJECT2~AT~X2~~Y2~~Z2~!
1UNGRASP~OBJECT1~:~STATUS~!
30 16
```

PBASE

```
5FIND_SPACE_FOR~OBJECT~:~X~~Y~~Z~!
2 1

1FIND~OBJECT~:~X~~Y~~Z~!
2 1

4GRIP~OBJECT~!
2 1

6MOVE~OBJECT~TO~X~~Y~~Z~!
2 1

3MOVE_HAND_TO~OBJECT~AT~X~~Y~~Z~!
2 1

2PRESHAPE_HAND_FOR~OBJECT~!
2 1

7UNGRASP~OBJECT~:~STATUS~!
2 1
```

DBASE

POLY1
POLY2
POLY3
POLY4
POLY5
POLY6

CYLINDER1
CYLINDER2
CYLINDER3
CYLINDER4

KBASE2

```
3WHAT-IS-IN~IMAGE_FILE~:FOUND~OBJECT~!
1BUILD-FEATURES~MODEL_FILE~:~FEATURE_TABLE~!
1HYP_GEN~IMAGE_FILE~:~HYP_RES~!
1HYP_VER~HYP_RES~:~OBJECT~!
2 1
```

PBASE2

```
1BUILD-FEATURES~MODEL_FILE~:~FEATURE_TABLE~!
2 1

2HYP_GEN~IMAGE_FILE~:~HYP_RES~!
2 1

3HYP_VER~HYP_RES~:~OBJECT~!
2 1
```

PSEUDOCODE FOR HOUGH'S TRANSFORM

```
Hough_Transform()

int *INPUTIMAGE, *OUTPUTIMAGE
int *ACCUMULATOR_ARRAY

BEGIN
    Split INPUTIMAGE into ROWS;
    Convolve each ROW in  parallel,  using  Laplacian  mask,  and
    Put_Results in OUTPUTIMAGE;

    Find_Max_Min gray values in OUTPUTIMAGE;
    Spread_Results to every node;

    Split OUTPUTIMAGE into ROWS;
    Normalize eahc ROW in parallel and Put_Results into
    OUTPUTIMAGE;

    Compute_Histogram;
    Spread_Results to each node;

    Ask_User for TRESHOLD PERCENTAGE;
    Caclulate_Treshold_Value;
    Split OUTPUTIMAGE into ROWS;
    Do_Tresholding on each ROW in parallel and Put_Results in
     OUTPUTIMAGE;

    Allocate_Room for ACCUMULATOR_ARRAY;
    Split OUTPUTIMAGE into ROWS;
    Calculate_Value in ACCUMULATOR_ARRAY, for all pixels in
     ROWS, in parallel, and Put_Results in ACCUMULATOR_ARRAY;

    Split ACCUMULATOR_ARRAY into ROWS;
    Check_For_Lines of length >= 10 in ROWS, in parallel;
    If_Line_Found, Print_Out THETA, RHO, END and START
     coordinates for line.
END
```

```
@device(x2700)

#include <us.h>          /* Uniform System routines */
#include <stdio.h>

int Rows,Cols,End;       /*# of rows,cols in image + # elements in subparts*/
int **Image,**Oimage;    /* Input and output images */
int **acc;               /* Accumulator array */
int max,min;             /* Max and min gray values in image */

main()                   /* Parent process */
{
        int x;           /* Treshold percentage */
        int READY = 0;   /* Loop control */
        InitializeUs();  /* Initialize Uniform System on all nodes */
        Allocate_Images();      /* Allocate memory for the images */

/* The following routine executes DoConvol in parallel for index */
/* values between 0 and Rows-3.  InitforProc1 is an initialization */
/* routine which allocates space for needed variables on all nodes */
/* that will run DoConvol.  It also gives variables a value. */
/* FinalforProc1 cleans up any variables that will not be used later. */
/* The 0 holds the place of a problem description data structure */
/* which is not needed in this program */

        GenOnIndexA(InitforProc1,DoConvol,FinalforProc1,0,Rows-2);

        FindMaxMin();    /* Find the max,min gray values in the image */
        Share(&max);     /* Let all nodes get a copy */
        Share(&min);

/* Since I already have all the needed variables on each node, I do */
/* not need to do any initialization.  I don't want to clean up any- */
/* thing either.  Therefore, a call to GenOnIndexA is not needed. */
/* GenOnI does the same thing, except for initialization and clean-up. */

        GenOnI(Normalize,Row-2);

        Histogram();     /* Compute the histogram of the image */

        system("clear");/* Ask user about the tresholding percent */
        printf("\n\n\n                  Tresholding\n\n\n");
        printf("        Input treshold value X (0 - 100):  ");
        while(!READY)
        {
                scanf("%d",&x);
                if((x<0)||(x>100)) printf("\n\n Try again (0<=x<=100): ");
                else READY = 1;
        }

        Treshold(x);     /* Calculate the treshold value */
        GenOnI(Binary,Row-1); /* Do the tresholding */
        AllocateAcc();  /* Allocate memory for acc. array.  Set up vars. */
        GenOnI(Accumulator,Row-1); /* Create the accumulator array */

        /* Find lines and print them */
```

```
        GenOnIndexA(InitforProc2,LineFind,FinalforProc2,0,Max-1);
}



Allocate_Images();      /* Allocate memory for images and scatter them */
{                       /* all over memory */
        int i,j;
        Rows = 64;      /* Size of image */
        Cols = 64;
        Iimage = (int **)AllocScatterMatrix(Rows,Cols,sizeof(char));
        Oimage = (int **)AllocScatterMatrix(Rows,Cols,sizeof(char));

        for(i = 0;i < Rows;i++) /* Create a simple image */
           for(j = 0;j < Cols;j++)
               Iimage[i][j] = i%2;

        Share(Rows);    /* Let all nodes get a copy of variables */
        Share(Cols);
        Share(&Iimage); /* Let all nodes get a copy of the pointers */
        Share(&Oimage);
}



InitforProc1(dummy)     /* Routine for initializing variables on nodes */
int dummy;
{
        int *Row;
        int *Prev;
        int *Next;
        int *Result;
        End = Rows-1;   /* # of elements in subparts of image */
        Row = (int *)malloc(Cols*sizeof(int));  /* Subpart of image */
        Prev = (int *)malloc(Cols*sizeof(int)); /* Prev. subpart */
        Next = (int *)malloc(Cols*sizeof(int)); /* Next subpart */
        Result = (int *)malloc(Cols*sizeof(int));/* Holds result */
}



FinalforProc1()
{
        free(Prev);     /* Clean up variables not needed anymore */
        free(Next);
        free(Result);
}



Do_Convol(dummy,r)      /* Convolution function: works on one subpart, */
                        /* which is a row of the image */
int dummy,r;            /* Dummy, range of suparts on which to do convol */
{
        int i;          /* Loop control */
        if (r&1) r = Rows-r-2;  /* Start on top or bottom ? */

/* I use the Laplacian matrix in my convolution.  It is a 3x3 matrix. */
```

```
/* Therefore, each process will reference 3 rows.  However, since */
/* processes working on neighbouring rows will have to reference 2 */
/* shared rows, there will be memory contentions.  To solve this, */
/* processes working on even numbered rows will start at the top of the */
/* image.  Processes working on odd numbered rows will start at the */
/* bottom.  The "if" statement ensures this.  Memory contention will */
/* then occur only when the two processes access data in the middle. */

/* Do_bt, below, is a procedure which transfers a copy of the */
/* specified size (last argument) from the start of the first */
/* argument into the second argument */

        Do_bt(Iimage[r++],Prev,Cols*sizeof(int)); /* Let each node get */
        Do_bt(Iimage[r++],Row,Cols*sizeof(int));  /* the subpart (row) it */
        Do_bt(Iimage[r--],Next,Cols*sizeof(int)); /* is going to work on */

        for(i = 1;i < End;i++)  /* Calculate the result pixel value */
           Result[c] = -Prev[i]-Row[i-1]+(4*Row[i]-Row[i+1]-Next[i];

/* Transfer a copy of the results into the output image */

        Do_bt(Result,Oimage[r],Rows*sizeof(int));
}



FindMaxMin()    /* Finds max and min gray values in the image */
{
        int i,size,v;   /* Loop control, size of image, gray value */
        size = Rows*Cols;       /* Calculate size of image */
        max = Oimage[0];        /* Initialize max and min */
        min = Oimage[0];

        for (i = 0;i < size; i++)       /* Find max and min.  This is */
        {                               /* done in serial, which must */
                        v = Oimage[i];  /* be changed */
                if (v > max)  max = v;
                if (v < min)  min = v;
        }
}



Normalize(dummy,r)      /* Normalizes all gray values to 0-255 */
int dummy,r;            /* Dummy, range of subparts to execute on */
{
        Do_bt(Oimage[r],Row,Cols*siezeof(int)); /* Each node gets a row */

        for(i = 0;i <= End; i++)                /* Normalize values in row */
                Row[i] = (Row[i] - min)*255/(float)(max - min);

        Do_bt(Row,Oimage[r++],Cols*sizeof(char));/* Put results in Oimage */
}



Histogram()     /* Calculates histogram of image */
{
```

```
        int Hist[256];  /* Holds histogram */
        int i,v;        /* Loop control, gray value */

        for(i = 0;i < size; i++)        /* Do the calculation. This is */
        {                               /* done in serial, which must be */
                v = Oimage[i];          /* changed */
                Hist[v] = Hist[v]+1;
        }

        Share(Hist);    /* Let all nodes get a copy of the histogram */
}



Treshold(x)     /* Find treshold value */
int x;          /* Treshold percentage */
{
        int t,i,P;      /* Treshold value, loop control, current percentage */
        long SUM;       /* The number of pixels added so far */

        SUM = 0;        /* Initialize */

        for(i = 255;i >= 0;i++) /* Add # of pixels and find treshold */
        {                       /* Done in serial, must be changed */
                SUM = SUM + Hist[i];
                P = (SUM*100)/size;
                if (P > x) break;
        }

        if (i != 0) t = i - 1   /* Ensure correct treshold value */
        else t = i;

        Share(&t);      /* Let all nodes have a copy */
}



Binary(dummy,r) /* Do the tresholding by making a binary image */
int dummy,r;    /* Dummy, range of subparts (rows) to execute on */
{
        int i;  /* Loop control */

        for(i = 0;i <= End;i++) /* Make the row binary */
                if Row[i] < t  Row[i] = 0
                else Row[i] = 1;

        Do_bt(Row,Oimage[r++],Cols*sizeof(char)); /* Results to Oimage */
}



AllocateAcc()   /* Allocate memory for accumulator array */
{
        int num = 18;   /* Number of different angles */
        int grad = 10;  /* Interval between angles */
        int Max;        /* Max # of rows in acc. array */

        Max = (int)(Rows/sqrt(2.0) + Cols/sqrt(2.0));
```

```
        Share(num);     /* Let every node get a copy of variables */
        Share(Max);
        Share(grad);
        Acc = (int **)AllocScatterMatrix(Max,num,sizeof(int));
        Share(&Acc);    /* Copy of pointer to acc. array (scattered */
}                       /* all over memory) to all nodes */



Accumulator(dummy,r)    /* Calculate values in acc. array */
int dummy,r;            /* Dummy, range of rows to calculate for */
{
        int i;          /* Loop control */
        int step;       /* Counts through all values of theta */
        int theta;      /* Holds angle */
        int rho;        /* Holds distance */

        Do_bt(Oimage[r++],Row,Cols*sizeof(int)); /* Get row for nodes */

        for(i = 0;i < End;i++)  /* For each pixel found in edge detection, */
           if(Row[i] == 1)      /* calulate theta,rho + add to acc. arrray */
           for(step = 0;step < num;step++)
           {
                theta = (step*grad)*PI/180.0;
                rho = (int)(i*cos(theta) + r*sin(theta)); /* x=i,y=r */
                rho = rho + Max - 1;
                Acc[rho*num+step] = Acc[rho*num+step] + 1;
           }
}



InitforProc2(dummy)     /* Give all nodes necessary variables */
int dummy;
{
        int x,y;        /* X,Y coordinates in Oimage */
        int START;      /* Indicates start and end of line to be printed */
        int END;
        int *AccRow;    /* A row of the accumulator array */
        AccRow = (int *)malloc(num*sizeof(int));
        int length = 10;/* Minimum lenght of a line to be printed */

}



FinalforProc2()         /* Last clean-up */
{
        free(Row);
        free(Accrow); free(Acc);
        free(Iimage); free(Oimage);
        free(Hist);
}


LineFind(dummy,r)       /* Finds and prints lines with lenght >= 10 */
int dummy,r;            /* Dummy, range of rows in acc. array */
```

```
{
int i;          /* Loop control */

Do_bt(Acc[r],AccRow,num*sizeof(int)); /* 1 row of Acc. array */
                                      /* per node */
for(i = 0;i < num;i++)  /* Find lines */
   if(AccRow[i] >= lenght)      /* Found line >= 10 */
   {
    START = 1;
    END = 0;
    for(x = 0; x < Cols;x++)
    {
     y = ((r-Max+1-x*cos(i*grad*PI/180.0))/sin(1+i*grad)*PI/180.0);
     if (Oimage[y*Cols+x] == 1)
       if(START == 1)   /* Print rho,theta + start coordinates of line */
       {
        printf("\nrho=%d theta=%d (X1,Y1)=(%d,%d)",r-Max+1,i*num,x,y);
        START = 0;
        END = 1;
       }
     else if (END == 1);/* Print end coordinates of line */
       {
        printf(" (X2,Y2)=(%d,%d)\n",x-1,(r-Max+1)*cos(i*num*PI/180.0)
                                          /sin(1+i*num)*PI/1);
        END = 0;
        START = 1;
       }
    }

    /* End point is on boarder of image.  Print it */

    if (END = 1) printf("(X2,Y2)=(%d,%d)\n",x-1,(r-Max+1-(x-1)
                                  *cos(i*grad*PI/num);
   }
}
```