The Architecture of the Butterfly Plus Parallel Processor

David Kotz

Duke University TR CS-1988-6, January 1988

A⁻

The Architecture of the Butterfly Plus Parallel Processor

David Kotz

December 16, 1987

Abstract

This paper investigates the architecture of the Butterfly Plus Parallel processor, an MIMD shared-memory machine based on the Motorola MC68020 microprocessor and a multi-stage interconnection network. The primary emphasis is on the interaction of components of the system rather than on the details of its components, especially the standard Motorola components. However, particular attention is paid to the memory management issues since this is the significant difference between the Butterfly Plus and its predecessor, the Butterfly Parallel Processor.

1 Introduction

The Butterfly Plus^{TM} Parallel Processor[2] is an MIMD machine with asychronous processing nodes based on the Motorola MC68020 microprocessor and a multi-stage interconnection network. The Plus is an extension of the original Butterfly Parallel Processor (the Butterfly I), incorporating the MC68851 memory management unit to improve the virtual memory capabilities of the Butterfly. The focus of this paper will therefore be on the memory management aspects of the architecture, including a description of its use by the current operating system for the Plus, Chrysalis Plus^{TM}. The emphasis throughout the paper, however, is not to describe any portion of the architecture in detail but instead to investigate the architecture as a careful construction combining standard and custom parts to form a high-performance multiprocessor.

The Butterfly Plus is made up of up to 256 independent processing nodes, each with a significant amount of memory (4-16 Mbytes). The MC68020 processor ([9],[11]), the core of each processor node, is assisted by two coprocessors, the MC68881 floating-point coprocessor ([8],[13]) and the MC68851 programmable memory-management unit (PMMU) ([7],[12]). Since all communication between processors is performed through shared memory, access to remote memory is a central part of the architecture. The Processor Node Controller (PNC) on each node automatically determines whether a physical memory reference is local or remote and processes the reference in a manner that is completely transparent to the processors. The PNC accesses all remote memory through a multi-stage switch that connects the nodes. The processor nodes and the switch are constructed to allow for a scalable architecture that may be configured with any number of processors from one to 256.

The description of the architecture will begin in Section 2 with a discussion of the components of each processor node, including the processor, the coprocessors, and the PNC. Following this, the Butterfly Switch interconnection network is described by Section 3. Section 4 discusses the use of the memory-management hardware by the Chrysalis Plus operating system. Finally, Sections 5 and 6 mention the I/O capabilities and the performance, respectively.

2 The Butterfly Plus Processor Node

Each processor node of the Butterfly Plus, comprising the CPU, coprocessors, memory, switch interface, and limited I/O interfaces, is contained on a single card. The components of the processor node are shown in the block diagram in Figure 1. Coupled tightly to the MC68020 microprocessor are its coprocessors, described below, which do floating-point and memory-management operations. The processors run at a clock speed of 16MHz, their maximum rated speed, and the PNC and switch operate at 8MHz.

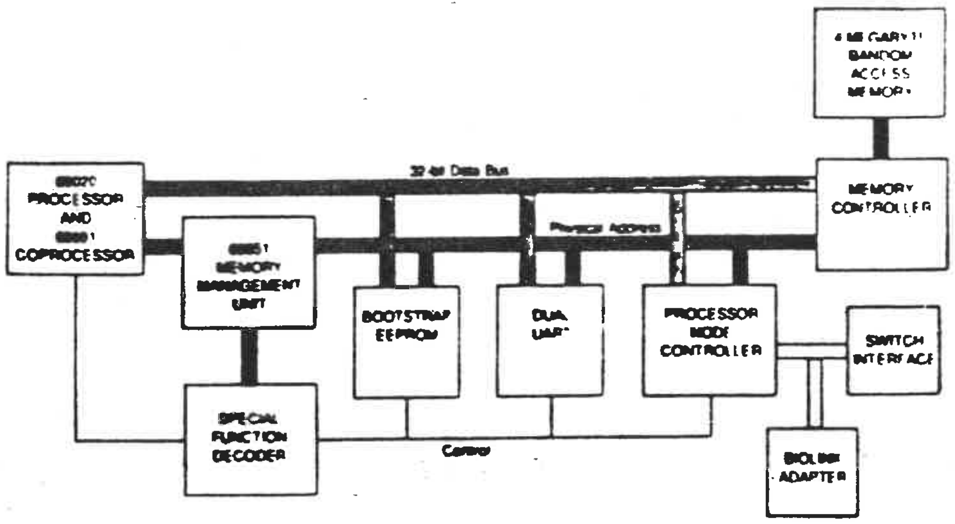

Figure 1: Processor Node block diagram[2].

2.1 Overview of the Microprocessor

The MC68020 processor is currently an industry-standard general-purpose microprocessor. It has full 32-bit internal and external address and data paths. It contains an on-chip, 64-set, direct-mapped instruction cache, containing four bytes of instruction. The cache mapping is based on logical addresses, using low-order bits for indexing and high-order bits for the tag. It is therefore likely that a several-word segment of instructions for a process may reside fully in the cache, eliminating instruction fetches and allowing for a complete overlap of operand fetches with the fetch of the next instruction[9].

The microprocessor uses several pipelining and parallel techniques internally to improve its performance, although it is not externally a pipelined or parallel processor. The micromachine is pipelined, using three levels, with one word in each level: control generation, instruction decode, and execution[9]. Limiting the depth of the pipe to three words reduces the penalty for branch instructions to a reasonable level, while obtaining the performance benefits of a pipelined instruction unit. The execution unit itself is parallelized, allowing simultaneous calculation of the instruction address, operand address, and data operations. It is this parallelism that allows the data fetches to be overlapped with instruction fetches that hit in the cache, since the instruction and data addresses are ready simultaneously. The bus controller portion of the chip operates autonomously with respect to the micromachine, so instructions that do not require the bus (*i.e.*, they reside in the instruction cache and do not require any operands) may execute completely concurrently with the bus operations from preceding instructions.

The MC68020 supports up to eight coprocessors using a special interface protocol based on normal bus cycles. This allows for nearly transparent extensions to the instruction set of the microprocessor ([9], pp. 105-106). The MC68020 recognizes certain instruction formats as coprocessor instructions and sends these instructions to the appropriate coprocessor. The coprocessor in turn may request service from the processor for evaluation of effective addresses, operand fetching, synchronization, branching, and so on. This coprocessor interface is the key to the use of the floating-point and memory-management coprocessors.

2.2 Floating-point coprocessor

The MC68881 floating-point coprocessor performs many complex floating-point operations for the processor. In addition to providing the full range of IEEE floating-point specifications, it computes square-root, transcendental functions, and trigonometric functions. Performing these operations on a separate processor allows for some degree of concurrent instruction execution: while the MC68881 only allows one floating-point instruction to be executed at a time (*i.e.*, there is no pipelining, at least at this level), the main processor will continue executing non-floating-point operations concurrently ([13], p. 1-8). The coprocessor may instruct the main processor to wait if a floating-point instruction is issued before a previous one has finished execution.

In the Butterfly Plus, the MC68881 is connected to the logical address bus, and since all memory references by the coprocessor go through the processor, these references are resolved exactly as they would in any other instruction. To the programmer, the floating-point instructions seem as if they were a part of the main processor.

2.3 Memory-management coprocessor

The addition of the MC68851 memory-management coprocessor to the original Butterfly architecture is the single most significant change evident in the Butterfly Plus, and allows for a much more sophisticated and flexible control over virtual memory in the multiprocessing environment. The operating system ~~as~~ is free to choose its own style of memory management, rather than being limited to the fixed style of the original Butterfly system. The MC68851 PMMU, like the MC68881, is a coprocessor serving the main processor on the bus. Its role, however, is generally more subtle; other than occasional control instructions from the processor, it automatically translates addresses from the logical address bus, where the main processor and the floating-point processor live, to the physical address bus, where the memory and PNC live. All addresses generated by the processor are translated by the memory-management coprocessor.

The PMMU supports a flexible virtual memory environment. Up to four levels of page tables are permitted, selected by software, and a variety of formats allow for the use of simple or complex page table structures. The physical architecture of the Butterfly Plus does not force any particular structure on the virtual memory; the operating system is free to select its own.[1] In Section 4 the use of the memory-management unit within the Chrysalis Plus operating system is described.

A logical address is presented to the coprocessor as a 32-bit address accompanied by a four-bit *function code*. The function code defines several separate 32-bit logical address spaces: user text, user data, supervisor text, supervisor data, and a supervisor *CPU space*. Addresses in CPU space are passed through without change so the processor may access physical memory addresses directly.

The coprocessor contains a 64 entry fully associative *Address Translation Cache (ATC)*, a

[1]Exception — see the discussion of the *special space* in Section 4.2.

translation-lookaside-buffer, which can translate logical addresses that hit in the cache in one cycle time. The cache entries are selected by function code, logical address, and a ***task alias*** number, which is related to the current "task", or process, executing in the main processor; this is support for timeshared operating systems. Up to eight separate tasks may share the cache. This removes the need to flush the cache after a process context switch: if the same task alias can be reassigned to this process, no entries need be invalidated (see comments below regarding root pointers). Otherwise, only the entries for the assigned task alias are invalidated. In addition to the physical address, each cache entry contains copies of relevant bits from the page table entry. If the page represented by this entry is modified and the cached "modified" bit is not set, the "modified" bit is set in the cache and in the page descriptor ([12], p. 5-24).[2]

explain how it could

If the logical address mapping can not be found in the ATC, the coprocessor follows the chain of page tables in main memory — "tablewalks" — to find the appropriate physical address for the requested logical address. There is a ***root pointer*** data structure kept in the coprocessor for user accesses, supervisor accesses, and DMA accesses that contains the physical address of the root of the current page table tree for each type of access. The eight most-recently-used user root pointers are kept in the coprocessor to allow automatic reassignment of the same task alias last used with that root pointer, so that the entries in the ATC for that task alias may still be used.

The depth and characteristics of the page table tree are controlled by the ***translation control register*** of the coprocessor. This register defines the way the 32-bit logical address is broken down into the indices for the various levels; its format is given in Figure 2. The TIA, TIB, TIC, and TID fields define the width of the fields of the logical address used for the four levels of the page table tree, respectively; any field that is zero indicates that that level of the tree does not exist. Thus, for example, a two-level mapping will have zero in the TIC and TID fields. The PS (page size) field specifies the page size (as a number of bits in the logical address), which may be anything from 256 bytes to 32 Kbytes[7]. The IS (initial shift) defines the number of high-order bits to skip in the address before extracting the other fields. Figure 3 shows a typical four-level tablewalk to translate a logical address into a physical address.

The page descriptors and table descriptors at any level may be shared by several trees, which allows (for example) shared text segments, possibly at different logical addresses for different processes. In addition, pages and tables may be marked as ***shared global***, indicating that the specified range of logical addresses is the same for all tasks in the system. Only one entry in the ATC is

[2]The PMMU walks through the page table tree, setting the M (modified) bit at every level[1].

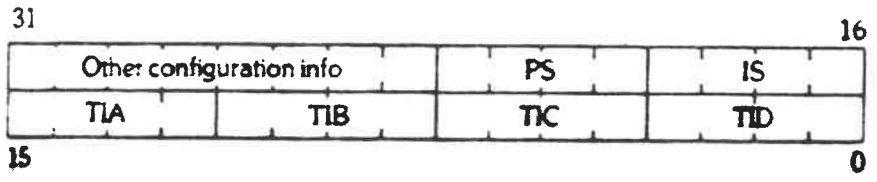

Figure 2: The MC68851 PMMU Translation Control Register[7].

maintained for shared global pages, regardless of task alias.

Pipelining is used extensively within the coprocessor to improve performance, particularly in the microcontroller. The address calculations used in tablewalking are also pipelined to produce two-bus-cycle-per-level translations. In addition, some operations in the coprocessor are overlapped; for example, bus arbitration is started while the first tablewalk microinstructions are executed[7].

2.4 Multiprocessing Considerations and the PNC

The MC68020 microprocessor is used with its coprocessors in the Butterfly in a standard configuration. Clearly, additional considerations must be made for the multiprocessing environment of the processor node, particularly access to the memory of other processing nodes. This is accomplished transparently by the *Processor Node Controller (PNC)*, which resides on the physical bus along with the memory. In the original Butterfly architecture, the PNC sat between the logical and physical busses, performing all address translation as well as local and remote memory access (see Figure 4). The responsibility for virtual memory support has been relieved by the addition of the PMMU, leaving the PNC responsible for remote memory access. Once a logical address has been converted by the PMMU and placed on the physical bus, the request is serviced by either the PNC (for remote access) or the memory module (for local memory access).[3] Since the first byte of all physical memory addresses is the processor node number, any address whose node number does not match the local node number is considered remote.

The PNC services remote memory references by sending a message to the PNC on the remote processor through the Butterfly Switch (Section 3). The remote PNC then does the memory access in its memory module, and responds if necessary. It is significant to note that all references to remote memory are handled entirely by the PNCs at either end of the transaction: the transaction is transparent to the two processors, requiring no software support. The only difference noted by the processors in accesses to local and remote memory is the delay: local memory (32-bit) references

[3]Or by the bootstrap EPROM or by the dual UART. Other memory-mapped I/O passes through the PNC to the BIOLINK adapter. See again Figure 1.

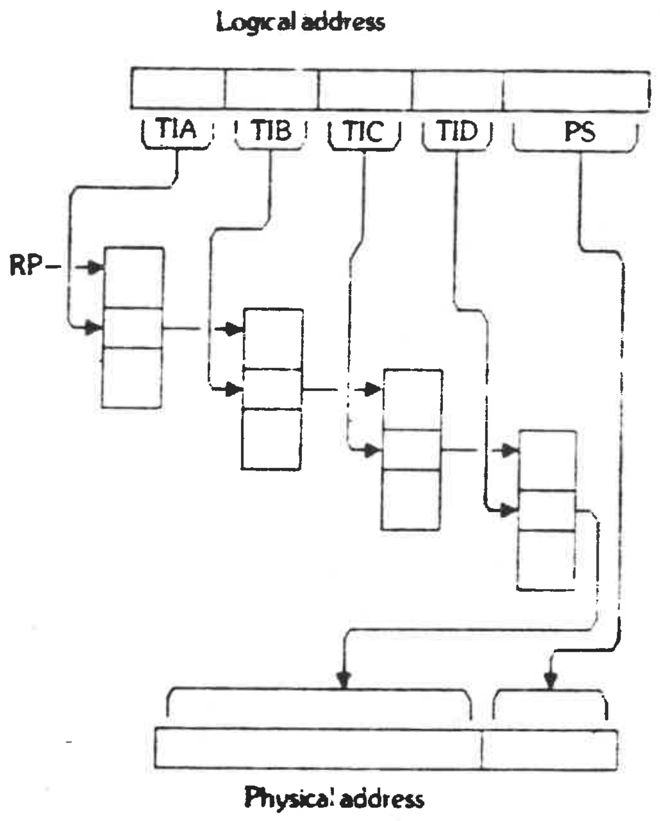

Figure 3: A typical four-level tablewalk[7].

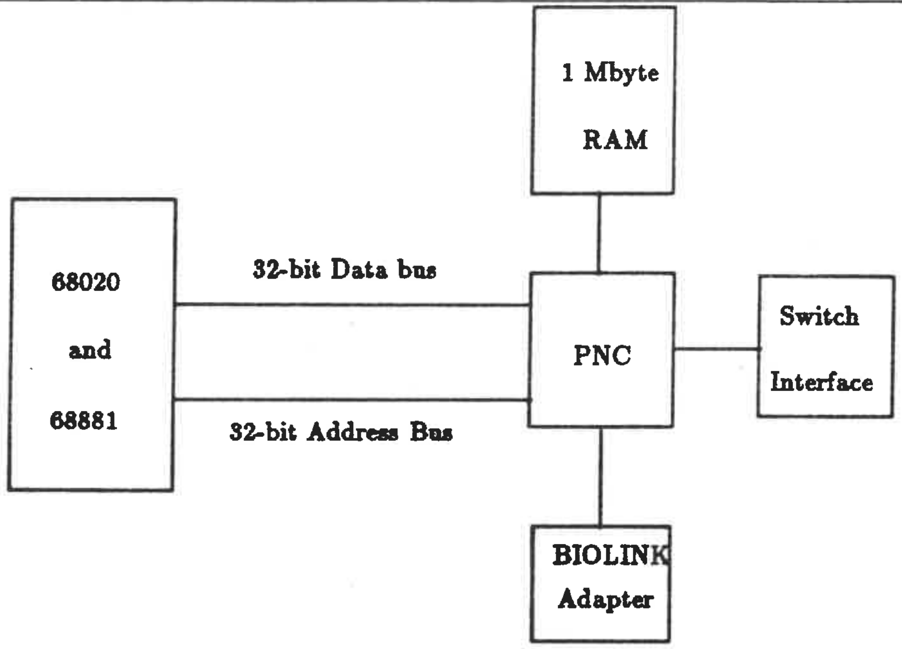

Figure 4: Block diagram of the Butterfly Plus predecessor.

require about 312 ns, and remote memory (32-bit) accesses require about 5000 ns, a factor of about 16 longer[1].

In addition, the PNC supplies several functions necessary or useful for supporting the multiprocessing environment. These include atomic reads and writes of 32-bit words, atomic arithmetic and logical operations, block transfer, and support for queuing and event posting operations. It also supplies the real-time clock and an interval timer, and assists in some scheduling operations. Since it can control the physical bus and is the sole connection to remote processors, it can stop all local memory transactions when a request for an indivisible (atomic) remote memory reference arrives and do the operation without interference from any processor.

The special operations performed by the PNC are invoked by writing to a special memory location that traps to the PNC microcode to handle that request. The PNC microcode runs on an AM2901 bitslice processor operating at 8MHz. This processor is pipelined to the extent that each microinstruction fetch is overlapped with the execution of the previous microinstruction. The microcode resides in a 4096 word by 64-bit static RAM, loaded at boot time from the bootstrap EPROM. This allows the microcode to be changed by the operating system if desired, as well as allowing greater flexibility in microcode debugging.

2.5 What is missing

The microprocessor at each node has an on-chip instruction cache, which certainly improves the performance of each processor node by reducing the number of logical and physical bus accesses and thereby avoiding contention at its memory module. However, although there is significant support in the processor and the memory-management coprocessor, the Butterfly processor nodes do not have data caches. Clearly, data caches are much more difficult to maintain than instruction caches in any environment, particularly a multiprocessor environment. If the cache were set up close to the processor, so that data from several nodes could reside in one cache, the cache consistency problem arises[14]. If, however, the cache were close to the memory module, all access to the local memory by both the processor and the PNC would be through the cache. There would therefore be no problem with cache consistency, as all data in the cache would be from the local memory only and all processors would go through the same cache. The potential advantages for a data cache — decreased average access time — would be limited in this scenario because no switch transactions would be avoided. It is probable that data caches were not included in the Butterfly processing nodes for these reasons, as well as complexity and board space limitations.

Interleaved memory might be expected in a high-performance architecture. BBN decided not to use an interleaved memory because of complexity and board space limitations[1]. In addition, Tom Blackadar mentioned that the PNC could not access memory as quickly as an interleaved design would supply it, and thus the simpler, cheaper single-module memory was chosen. ✓

could have used faster PNC
(i.e. faster bit slice Processor

3 The Butterfly Switch

The interconnection of the processor nodes is, of course, what makes a multiprocessor and characterizes its architecture. The name of the Butterfly parallel processor comes directly from its interconnection strategy, a multi-stage network that, when drawn, looks something like a butterfly. The network connects every node to every other at a uniform distance that grows as the logarithm of the number of processor nodes. Each node has one connection to the switch, through which it transmits all messages to other processors. No processing is done within the switch itself. As implemented, the network provides an interface that is flexible and transparent to the processor at each node. The Butterfly switch is described in further detail in [2].

The Butterfly switching network is a form of shuffle-exchange network, described in [14]. Each switching node in the network is based on a custom-designed VLSI chip that can switch four input lines to four output lines, each four bits wide, using an internal crossbar switch. Eight switch nodes are combined as in Figure 5 to form a two-column, 16-processor switch on one circuit board. Messages originating at at the left-hand side are switched through to the right-hand side. Each processor is really connected to both sides of the switch in this diagram, allowing for bidirectional communication between processors (although all messages flow the same way through the switch). Figure 6 illustrates this idea by representing the switching network as a cylinder.

This eight-node switch, on a single board, represents the smallest switch for the Butterfly — larger switches are formed by combining several switch boards. For example, a 64-node switch can be constructed with eight boards, doubling the width of the switch to four columns and the breadth of the switch to 64 connections. This flexible design allows for an arbitrary number of processor nodes (up to 256) to be configured in a Butterfly Plus system, and allows for simple expansion of existing systems.

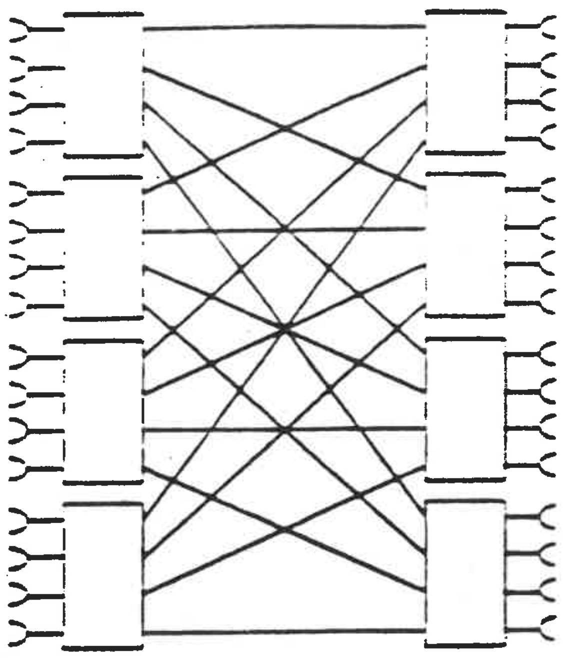
Figure 5: An 8-node, 16-processor switch[2].

There is one path through the switch for each pair of processors, although portions of this path are shared with other paths. The complexity of the switch (in terms of the total number of connections), grows as $N \log_4 N$ for N processor nodes, whereas a fully connected switch grows as

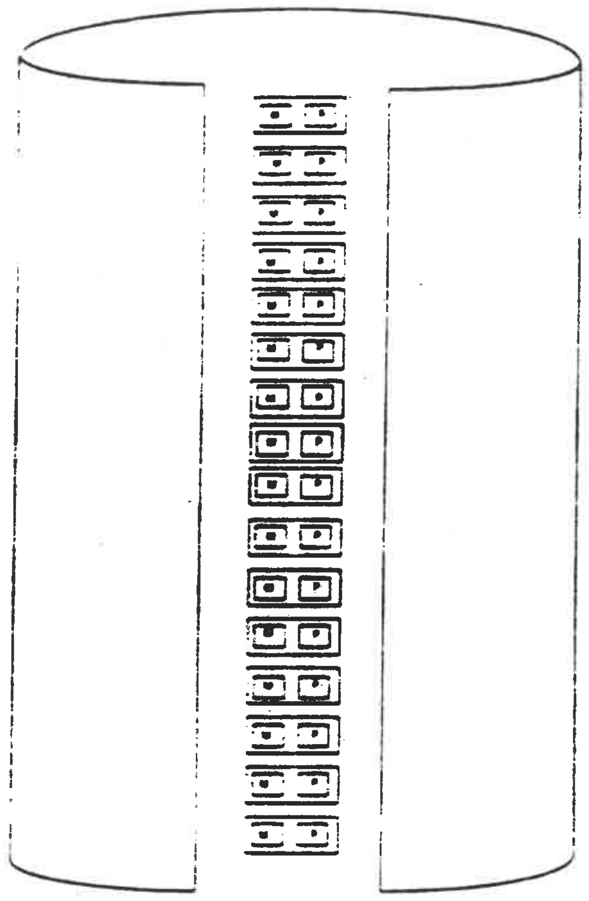

Figure 6: The same switch represented as a cylinder[2].

N^2. The width of the switch, which determines the delay for messages sent through the switch, grows as $\log N$.

Memory references and other transactions are accomplished with messages sent between the PNC at one node and the PNC at another node. Thus the switch itself need only support the transmission of messages from one processor to another. Each message contains the processor number of the destination, used as an address to select the output line to be used at each switch node along the path (see Figure 7). The packet is switched to one of the four output lines based on the front two bits of the address, regardless of the input line. In this example, in the first column the packet is switched to the third output line since its first two bits are 11. In the second column it is switched to the second output line since the next two bits are 10. The entire path is held open until the PNC at the destination has accepted the message and the entire message has been sent, so the scheme is circuit-switched for each packet[10]. The message is spread out along its path through the switch, so the nibbles (four-byte chunks) of the message are, literally, pipelined through the switch from source to destination. All messages are clocked through the switch one column at a time, using a clock speed of 8MHz, the same speed at which the PNC and the processor node switch interface operate. At this rate each path has an effective bandwidth of 32 Mbps, for a total bandwidth in a 256-node system of 8 Gbps[2].

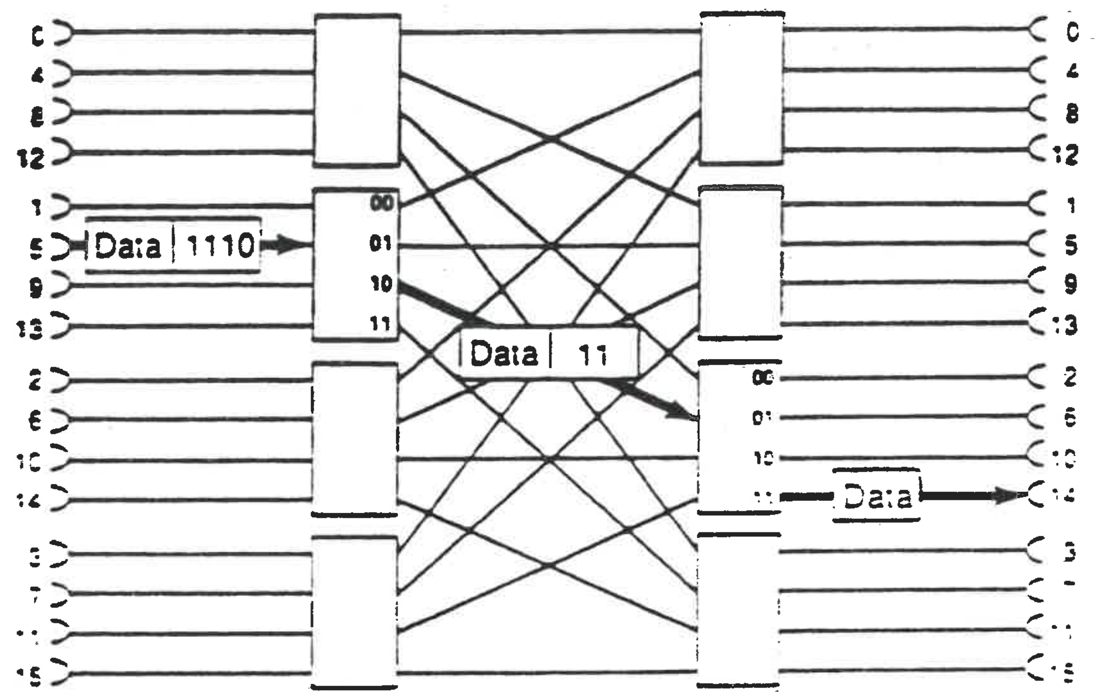

Figure 7: A packet for processor 14 moves through the switch[2].

Several messages may be simultaneously active in the switch. A conflict occurs, however, if more than one message desires the same output line from a switch node. Conflicts are resolved by permitting one message to continue and by rejecting the other message. In addition, a processor's switch receiver may reject a message if its buffers are full. A rejected message *retreats* by signaling back along its path to its sender, which will retransmit the message after a short random delay.

Short messages are used for such operations as single-word reads and writes, atomic operations, and so on. Typically, the PNC on the requesting processor node forms a message requesting the service from another node, and sends the message to that node. When the service is complete, and return data is required (*e.g.*, in a read operation), the remote PNC forms a response message and returns it to the sender. The components of the path between them may be used by several other

short messages have significant overhead, reducin bandwith by a factor of 4 or more messages in the interval.

In contrast, longer messages, used for block transfer, take better advantage of the pipelined operation of the switch since the initial delay for the head of the message to reach the destination, as well as the overhead of the address and checksum information that is sent with each message, is amortized over the length of the message. The concept of request and response is the same as for short messages. Since long messages may increase the delay for short messages, the operating system typically breaks requests for long block transfers into short transfers of 256 bytes ([2], p. 3-14, 3-28).

To provide an interconnection network that tolerates failure of individual switching nodes, alternate paths between pairs of nodes may be added by using additional switch columns. The multiple paths are represented by several switch addresses for the same destination processor. The switch interface hardware uses a different path for each transmission to a given destination. If the transaction fails, the next alternate path is used[10]. Another effect of this technique is to reduce conflict in the switch by spreading out the traffic in the network.

Any network must address the issue of deadlock. Deadlock is avoided in the Butterfly Plus switch with a retreat strategy in conflict resolution and by assuring that the switch interface at each node can always accept an acknowledgement packet. The latter condition is satisfied by including two input and two output buffers in each switch interface[2].

4 Memory Management

The addition of the MC68851 memory-management coprocessor is the single most significant change made between the original Butterfly processor node and the Butterfly Plus processor node. This has provided a more standard, more flexible virtual memory system than that used on the Butterfly I. Because of the programmable versatility of the MC68851 coprocessor, the particular style of the virtual memory used on the Butterfly Plus becomes an operating system issue. As an example, to demonstrate the capabilities of the architecture and some of the issues in designing a virtual memory structure on a multiprocessor system, this section describes the use of virtual memory by the Chrysalis Plus operating system. First, however, we define the physical memory layout.

4.1 Physical memory structure

The architecture defines the layout of the physical memory address space. Physical addresses on the Butterfly are 32 bits wide, derived from the processor set's use of 32-bit addresses and the need for a large address space. The first byte of each physical address is the processor node number, thus limiting the multiprocessor to 256 processors. This leaves 24 bits of address to specify the memory location within a node, or up to 16M of memory on each node. The minimum configuration has 4M of memory per node. Figure 8 shows the physical memory layout. Note the bottom 64K is allocated to object (memory, process, *etc.*) headers; this is the ***header segment*** of this node.

In addition, access to the memory mapped I/O is obtained by the use of an additional signal (PA32) obtained from the high byte of the logical address. When this byte is a hex FF, the signal is high and the resulting physical address is used for access to the Multibus, PNC, UART, and so on ([2], Appendix B). This, of course, places a hardware restriction on the use of this segment of virtual memory. It does, however, avoid a restriction on the amount of RAM that may be on each node by not reserving physical address space for memory-mapped I/O.

4.2 Virtual memory structure

The virtual memory, as seen by a Chrysalis process, is outlined in Figure 9. There are some notable features of this virtual memory layout. It is rather unusual to have the entire physical memory, the whole operating system, several operating system tables, and all of the special I/O space mapped into *every* process, but Chrysalis uses segments FC-FF for these purposes. Chrysalis is a very open system, providing very little protection, since users obtain control of entire nodes for their own use. This scheme also allows a number of operating system functions to be included in a library and performed without trapping to supervisor mode. The ***header segment*** (the bottom 64K of physical memory) of every processor node is mapped into segment FC, making access to objects convenient. Objects in Chrysalis have an identifying number that processes may use to request that the memory for that object be mapped into their own space. This is the mechanism that allows several processes to share memory. The identifier is of the form *WWXXYYZZ*, where *WW* is the processor node number where that object is located. A descriptive header for that object (which contains physical pointers to its TIC-level descriptors) is located in that node's header segment, and is found at logical address *FCWWYYZZ*.

System text and data structures are found in segment FD, and the entire local physical memory is mapped into segment FE. To access physical memory locations for this node, the operating system accesses the corresponding logical address in segment FE. The top segment, segment FF, is the ***special space***, which allows access to the EPROM, Multibus, PNC functions, and so on. Access to this segment uses the special PA32 signal mentioned above, in addition to the usual logical to

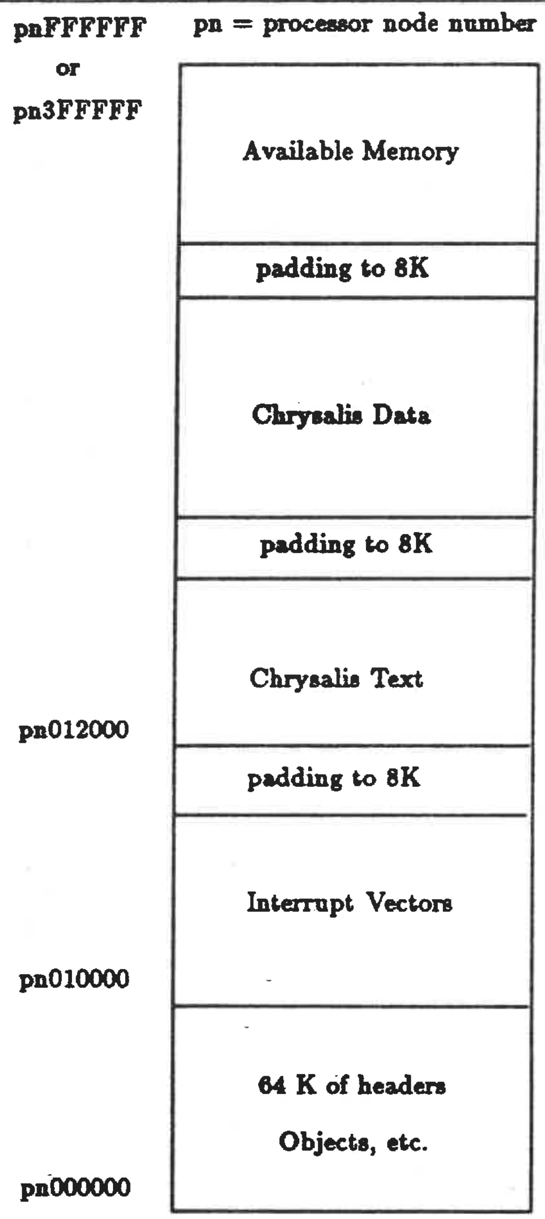

Figure 8: The physical memory on each node of the Butterfly Plus[3].

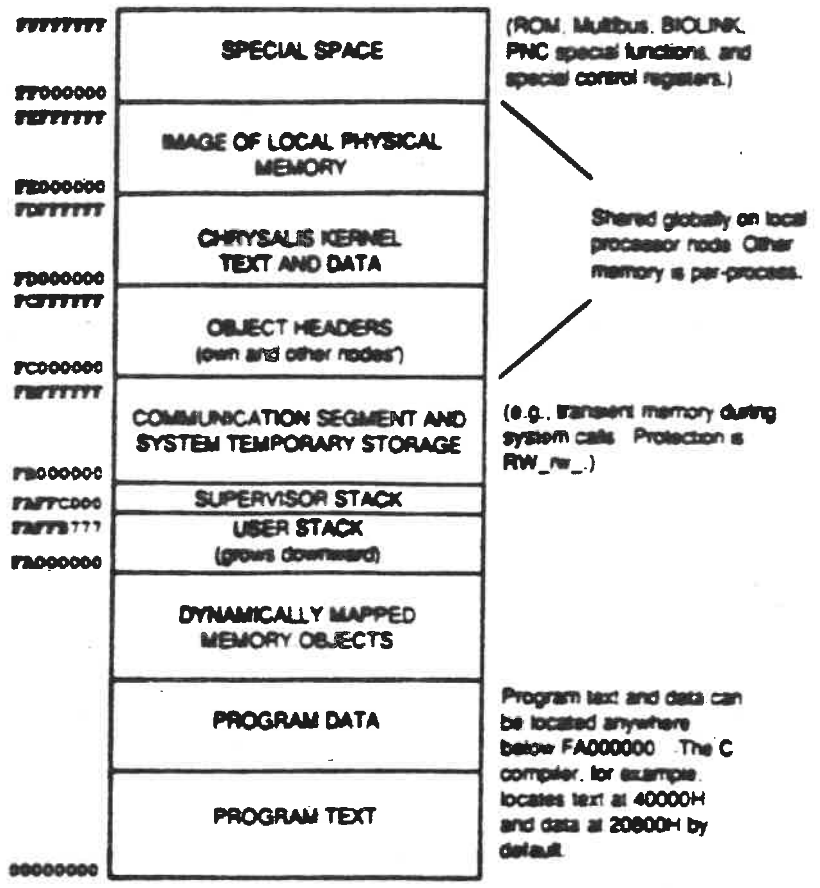
FFFFFFFF
SPECIAL SPACE
(ROM, Multibus, BIOLINK, PNC special functions, and special control registers.)
FF000000
FEFFFFFF
IMAGE OF LOCAL PHYSICAL MEMORY
FE000000
FDFFFFFF
CHRYSALIS KERNEL TEXT AND DATA
Shared globally on local processor node. Other memory is per-process.
FD000000
FCFFFFFF
OBJECT HEADERS (own and other nodes')
FC000000
FBFFFFFF
COMMUNICATION SEGMENT AND SYSTEM TEMPORARY STORAGE
(e.g., transient memory during system calls. Protection is RW_rw_.)
FB000000
FAFFC000
SUPERVISOR STACK
FAFFB777
USER STACK (grows downward)
FA000000
DYNAMICALLY MAPPED MEMORY OBJECTS
PROGRAM DATA
Program text and data can be located anywhere below FA000000. The C compiler, for example, locates text at 40000H and data at 20800H by default.
PROGRAM TEXT
00000000

Figure 9: The virtual memory seen by a process under Chrysalis Plus[6].

physical mapping provided by the translation tree.

4.3 The Chrysalis Page Table Tree

The page table tree for the virtual memory is derived from a logical address that is broken down according to the following scheme:

Logical Address

68851 field:	TIA	TIB	TIC	PS
32 bits:	8	8	3	13

Thus Chrysalis Plus uses 8K pages, with a three-level page table tree. The lowest level page descriptor table maps eight pages, or 64K; this is a historical connection the the Butterfly I 64K segments. The next level maps 256 segments, or 16 Mbytes. The highest level maps 256 of these units, or 4 Gigabytes (enough for the entire physical memory on a fully-configured Butterfly Plus). Beyond this, the function code level, there are four function codes used by Chrysalis: supervisor text, supervisor data, user text, and user data. Each function code space has its own TIA-level page table tree (although portions of the tree may be shared). The full page tree, therefore, looks like Figure 10.

Note that the TIA and TIB levels for a process reside entirely on the node where that process is running. In addition, all the text and data for that process reside on that processor node, along with their TIC-level tables. This is not strictly necessary, but provides much faster access to instructions. The TIC level for shared memory objects, mapped in as described above, lives on the node where the object is created, and is then included in the TIB-level table of processes that use the object. The PMMU will therefore cross node boundaries during a tablewalk, but transparently so, since the remote memory addresses look like any other physical address. The simplicity of this design is possible since the pages and tables do not move once created.

Since there is no secondary memory on the Butterfly Plus (at least at the current time), Chrysalis Plus does not consider paging, and therefore its maintenance of the virtual memory tree is vastly simplified. Of the status and control bits available in the page and table descriptors, Chrysalis only uses the WP (write protect) and SG (shared global) bits. The SG bit (only available in the long format descriptors) indicates that a portion of the logical address space is shared by all processes; it causes the PMMU to maintain only one translation cache entry for these logical addresses[12]. The globally shared portion of virtual memory under Chrysalis Plus is the FC-FF segments, where the function codes refer to different page table trees only at the TIA level, which

why?

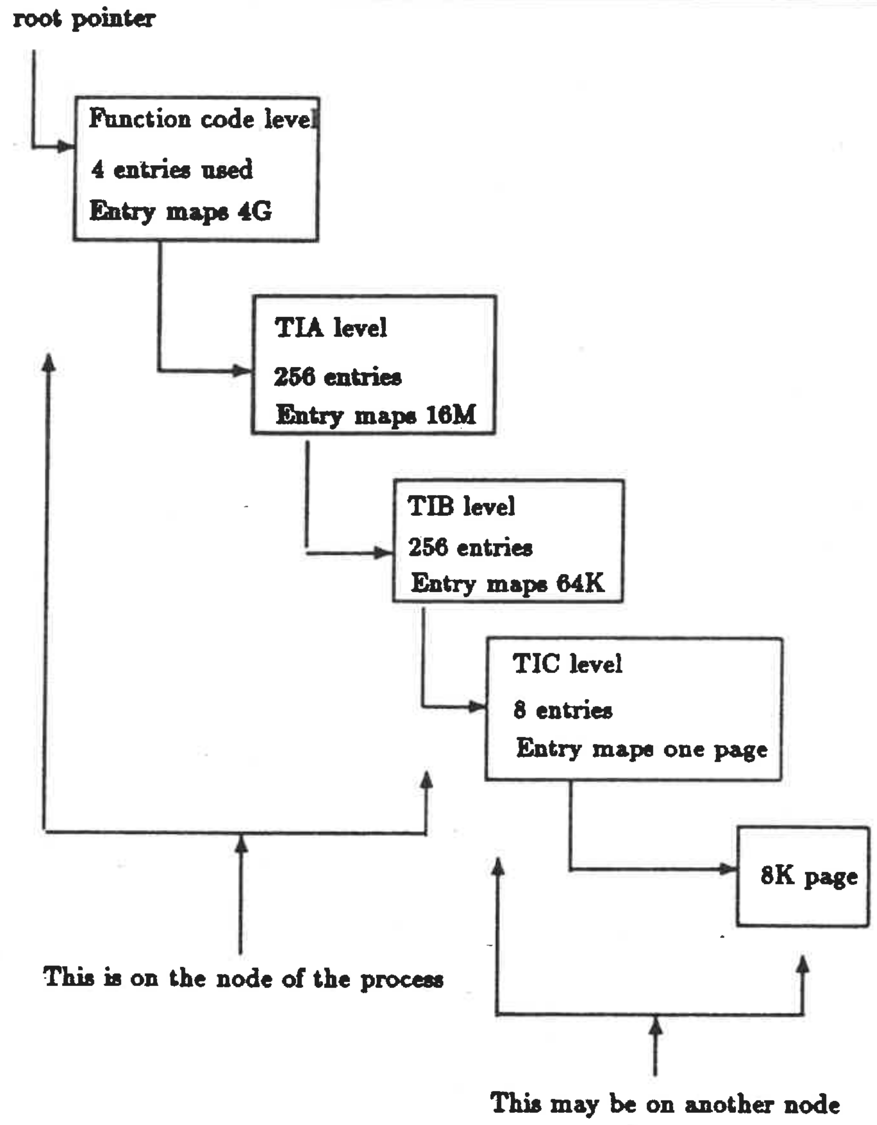

Figure 10: A portion of the page table tree for a Chrysalis Plus process[3].

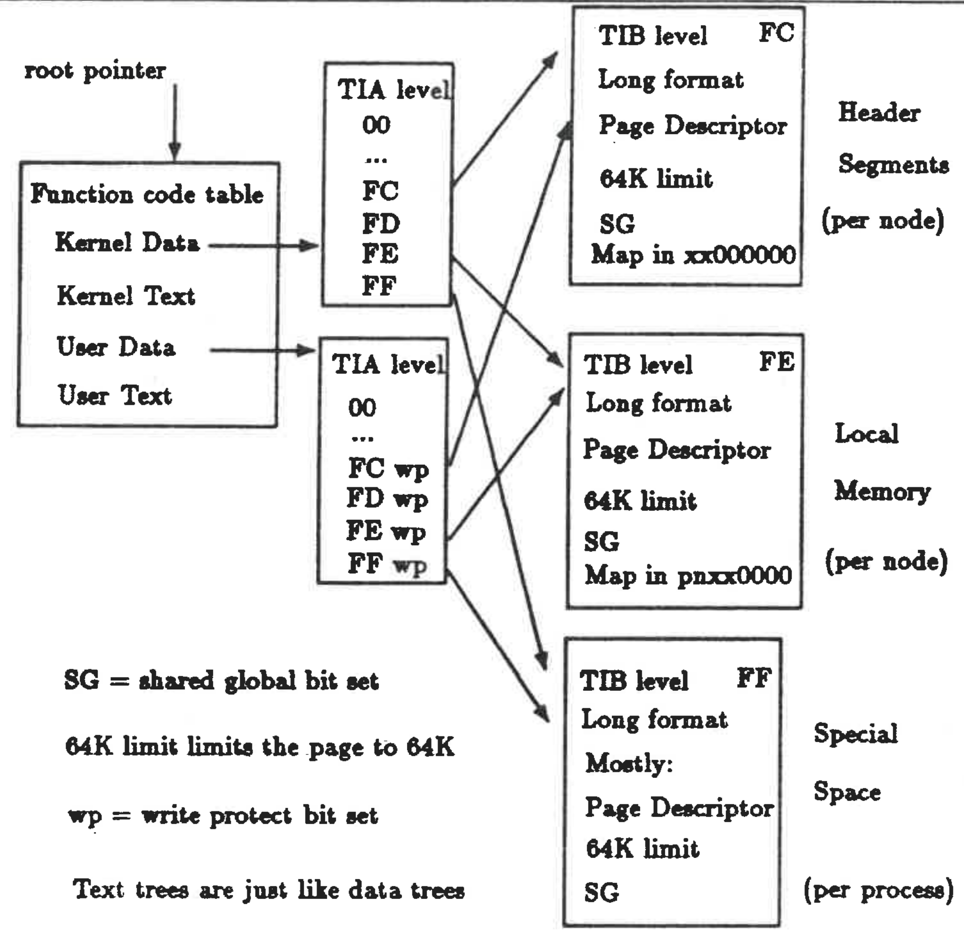

Figure 11: The operating system portion of the logical address space[4].

then all join at the TIB level. At the TIA level, the **WP** (write-protect) bit is set on the user side to protect the system from user processes[4].[4] At TIB level, the long page descriptors are used and the SG bit is set. No TIC level is used here. This is summarized by Figure 11.

The operation of the PMMU is controlled primarily by the setting of the translation control register (TC) and the root pointer, along with a few instructions regarding the address translation cache (ATC). The PNC microcode contains a function that sets the TC and root pointer given their values. The TC is set once, at startup, to reflect the mapping shown above. The root pointer is reset on every context switch to point to the page table tree for the current process. Chrysalis uses the PFLUSHA instruction, which invalidates all entries in the ATC[7], whenever the local page table tree structure is modified by the operating system. The PFLUSH instruction, which invalidates only entries of a particular function-code space in the ATC, is used for address translation before

[4]The **WP** bit is also used for text segment tables.

calling the PNC microcode for special functions[5].

5 Connections to the Outside World

The Butterfly Plus is provided with several ways to communicate with the outside world, typically through one or two particular nodes. Since the normal configuration of the Butterfly Plus does not include a disk, communications are very important. In addition, many applications may require high-speed I/O for processing. A brief summary of each of the options follows.

Access to a Multibus is made possible through an adapter card that plugs into a Multibus card cage and connects to the BIOLINK adapter of a processor node. A variety of peripherals may be used from the Multibus, the standard ones including serial lines, an ethernet controller, and a RAMboot card ([2], section 4).

The serial lines are used for the console, which is used for rebooting the system, and for the initial downloading of the operating system and network software. These lines are included directly on the Multibus adapter card. Additional serial lines, usually for user terminals, can be added with more Multibus cards.

The ethernet adapter is the most important peripheral on a running Butterfly Plus system, since all program downloading is done and (typically) all user connections are made through the ethernet. The Chrysalis Plus operating system software supports the usual TCP/IP network protocols.

The RAMboot card stores common programs, particularly those that are loaded over the serial line at system boot time, to speed access when the system needs to be rebooted and software must be reloaded to the processor node memory, where it is kept during normal operation.

In addition to the Multibus, an interface to a VME bus may be installed, providing high-bandwidth I/O (up to ten times more than with a Multibus) for applications that may require it, for example, high-speed graphics displays or disk controllers and buffer memory ([2], section 5). One or two interface boards are substituted in the place of processor nodes, connecting directly to the Butterfly Plus switch. About 5.5 Mbps of I/O bandwidth is available when both switch ports are used.

6 Performance

Although the performance of the Butterfly Plus is yet to be fully determined (since it is so new), it is clearly a high-performance machine, like its parent the Butterfly I. Each processor is rated at 2.5 Mips, for a total computational power of 640 Mips on a 256-processor system[1]. The local memory access time is 312 ns, half of what it was on the previous Butterfly model, and switch access time is around 5000 ns[1]. With 16 Mbytes of RAM available on each node, it is possible to have 4 Gbytes of RAM on the system. The switch can handle a peak rate of 32 Mbps per path, allowing a data transfer rate of 8192 Mbps with the full 256-processor switch[2].

7 Conclusion

The Butterfly Plus parallel processor represents an architecture that has been proven successful by its parent the Butterfly I, and that contains numerous improvements in speed and in functionality, particularly in the memory-management facilities. One notable feature of the architecture is its scalability: it works well with only a few processors yet scales smoothly (physically and financially) to a very large system of 256 processors. High-bandwidth I/O is available in the form of standard busses like the Multibus and VME bus. The processor set, based on the recent members of the MC68000 family, is well known and used in a standard configuration. The unique Butterfly switch allows for a remarkably elegant interconnection between processors that provides high bandwidth and a high level of transparency to the processor set. The memory management hardware allows for a variety of virtual memory implementations, chosen by the operating system designer.

To go beyond the Butterfly Plus, expanding to larger memories and a larger number of processors, will require a new design — perhaps microprocessors with larger address busses and an upgraded switch — but will not, I believe, need a significantly different general architecture.

References

[1] Tom Blackadar and Tom Clarke, BBN ACI, personal communication.

[2] BBN ACI, *Butterfly Parallel Processor: Inside the Butterfly Plus*, Preliminary Version of October 16, 1987.

[3] BBN ACI, Chrysalis 3.9.1 source file `include/mmu_plus.h`.

[4] BBN ACI, Chrysalis 3.9.1 source file `kernel/mmu_BF_PLUS.c68`.

[5] BBN ACI, Chrysalis 3.9.1 source file `kernel/traps.a68`.

[6] BBN ACI, "Chrysalis Plus in a Nutshell", October 20, 1987.

[7] Brad Cohen and Ralph McGarity, "The Design and Implementation of the MC68851 Paged Memory Management Unit", IEEE Micro, pp. 13-28, April 1986.

[8] Clayton Huntsman and Duane Cawthron, "The MC68881 Floating-point Coprocessor", IEEE Micro, pp. 44-54, December 1983.

[9] Doug MacGregor, Dave Mothersole, and Bill Moyer, "The Motorola MC68020", IEEE Micro, pp. 101-118, August 1984.

[10] Walter Milliken, BBN ACI, personal communication.

[11] Motorola, Inc., *MC68020 32-Bit Microprocessor User's Manual*, Second Edition, Prentice-Hall, Engelwood Cliffs, NJ, 1985.

[12] Motorola, Inc., *MC68851 32-Bit Paged Memory Management Unit User's Manual*, Prentice-Hall, Engelwood Cliffs, NJ, 1986.

[13] Motorola, Inc., *MC68881 Floating-Point Coprocessor User's Manual*, Prentice-Hall, Engelwood Cliffs, NJ, 1985.

[14] Harold S. Stone, *High Performance Computer Architecture*, Addison-Wesley, Reading, MA, 1987.