3 Communication Micro-Architectures

Designing a communication architecture requires a good understanding of the available building blocks. While one might argue that a good knowledge of digital design and of VLSI architecture is sufficient, it turns out that understanding the structures used in current communication micro-architectures is highly beneficial and provides insight into the issues that the communication architecture must address to make effective use of the hardware.

This chapter presents the communication micro-architectures of the two machines on which Active Messages has been implemented. The presentation is in sufficient detail to support Chapter 4 which discusses design alternatives and performance results for Active Messages. In addition, the systematic description of the two micro-architectures brings out a set of four key issues that must be addressed as part of the design of the communication architecture.

The development of Active Messages on existing micro-architectures has proven an invaluable first step. The implementations on the nCUBE/2 and the CM-5 demonstrate the efficiency of Active Messages and identify the major shortcomings of current micro-architectures. These efficient implementations are enabled by the fact that the hardware in these machines essentially exposes the micro-architecture directly to the software, relying on the kernel to abstract and virtualize the low-level functionality for the user process. In this respect, the situation for innovation in communication architectures is more favorable than it was in the late 70s for instruction set design. Unlike in the early days of RISC architectures it is possible, at least to some degree, to develop new communication architectures without developing new hardware. Obviously such a strategy is only useful as a first step and the feedback from the characteristics of the new architecture into the micro-architecture design is desirable sooner or later.

The presentation in this chapter assumes familiarity with conventional RISC and CISC processor micro-architectures and, to a lesser degree, with common communication micro-architectures. Unfortunately, the treatment of the latter in the literature is not very complete. To start with, one often finds a thorough confusion between the communication architecture and the micro-architecture to the point that many papers present experimental performance data as pertaining to the "machine", i.e., the hardware, and completely overlook that such experiments also measure the performance of substantial layers of communication software. The vendor's hardware manuals usually reveal most of the communication micro-architecture, but obtaining them is often difficult and then they do not contain full information or performance data.

The nCUBE/2 communication micro-architecture is described in Section 3.1. It is a memory-to-memory architecture with DMA to transfer messages between memory and the network. The micro-architecture is exposed to the kernel which is responsible for virtualizing it, e.g., enforcing protection boundaries is fully the responsibility of the message layer and no network access from user level is possible.

Section 3.2 describes the CM-5 communication micro-architecture. The major innovation is the incorporation of a simple protection scheme into the network interface which allows protected user-level access to the network, reducing the communication overhead considerably. Unlike the nCUBE/2, the CM-5 has a register-to-register communication micro-architecture optimized for small messages.

The descriptions of these two machines begin with a summary of the current implementation technology and of the overall node micro-architecture. The nCUBE/2 instruction set architecture is briefly described because it is non-standard and its understanding will help in the performance analysis of Active Messages in Chapter 4. The descriptions of the micro-architectures of both machines follow the schema sketched in Subsection 2.1.4.

Section 3.3 concludes by contrasting the two micro-architectures with each other to identify four key issues that these (and other) micro-architectures require the communication architecture to address.

3.1 The nCUBE/2 Micro-Architecture

The nCUBE/2 is a massively parallel processor consisting of small custom processing nodes interconnected in a binary hypercube. Each node consists of the custom NCUBE 6400 processor and DRAM. The processor integrates the CPU, memory interface and network interface on one single chip. The resulting nodes with the processor and a bank of 10 DRAM chips occupy a tiny 1.25" by 4" printed circuit board and are mounted 64 to a 16" by 20" motherboard without requiring any glue logic.

The best reference for the nCUBE/2 micro-architecture is the NCUBE 6400 processor manual [NCU89]. The information herein is additionally based on experiments with the hardware[1].

3.1.1 Node micro-architecture

The NCUBE 6400 VLSI processor forms the core of the processing nodes used in nCUBE/2 systems. As depicted in Figure 3-1, it integrates a 64-bit CPU (integer unit and IEEE floating-point unit), a simple memory management unit, a DRAM memory interface, an on-chip wormhole cut-through router, and 14 DMA ports into the hypercube interconnect.

The micro-architecture of the processor is not particularly novel: it is in essence a “reduced VAX.” The instruction fetch state machine decodes at most one operand per clock cycle and is relatively decoupled from the micro-coded execution unit. In an nCUBE/2 system, the processor runs at 20MHz yielding roughly 5 VAX MIPS or 2Mflops.

The memory interface connects the processor to an external 32-bit wide memory system with ECC. The interface allows direct connection of a bank of page-mode DRAM memory chips and typically 10 4-bit wide DRAM chips are used to provide the required 39-bit wide path.

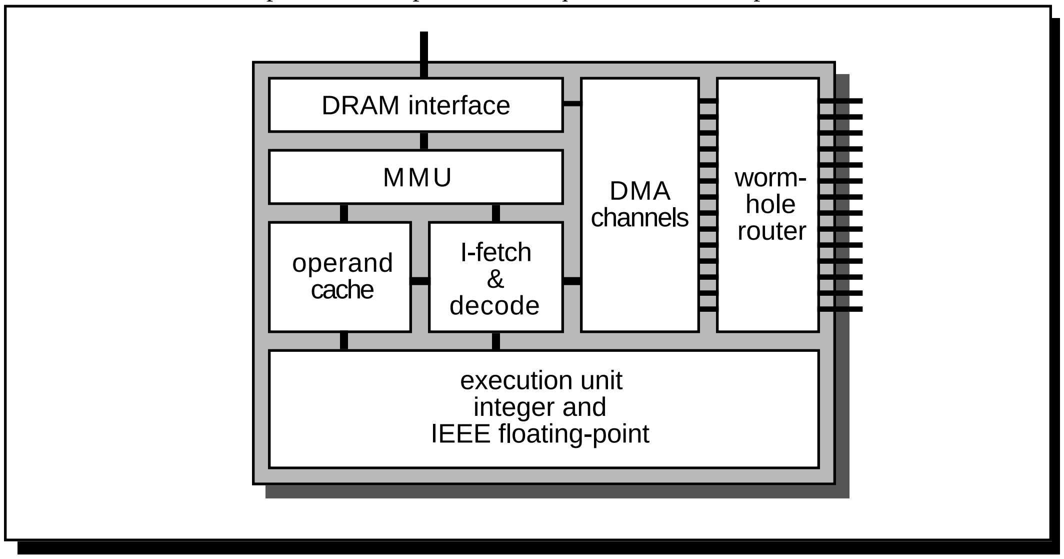


Figure 3-1: NCUBE 6400 processor block diagram.

The NCUBE 6400 processor chip integrates a complete nCUBE/2 node, with the exception of DRAM, onto a single chip. The full node occupies a tiny 1.25" by 4" daughter board which holds the processor chip and 10 DRAM chips (double-width versions with more DRAM also exist).

[1] Note that all nCUBE/2 performance data presented in this dissertation was obtained on nCUBE/2 systems. Newer nCUBE/2S and nCUBE/2E systems use a 25% faster clock and a somewhat better optimized micro-architecture. While these improvements should affect the performance noticeably, none of the fundamental aspects of the micro-architecture change.

The processor does not have traditional on-chip caches. However, the instruction fetch unit has a 128 byte prefetch buffer and the execution unit maintains a fully associative 8-entry operand cache.

The nCUBE/2 memory hierarchy is summarized below:

level	access	transfer
registers	1 cycle	64 bits
operand cache	2 cycle	32 bits
DRAM page	2 cycle	32 bits
DRAM	5 cycle	32 bits

The on-chip memory management unit (MMU) provides a simple segmented protection scheme. Each user process has one code segment and four data segments, each relocated via a base register and limited by a size register.

3.1.2 Instruction set architecture summary

The instruction set is reminiscent of the VAX ISA and encoded in the same style with operands of variable byte length. The main characteristics of the instruction set are:

- 16 64-bit general-purpose registers (joint integer and floating-point registers) and separate stack pointer and program counter registers,
- 11 data types: signed/unsigned byte/half-word/word/double-word integers, single/double floats and BCD,
- numerous addressing modes including auto-increment and memory indirection,
- an orthogonal 3-instruction memory-memory instruction set with instructions encoded in VAX-style using an op-code byte optionally followed by operand bytes, each followed by the variable-sized operand,
- repeat-mode allowing repetition of a single instruction to emulate vector instructions (e.g., repeated memory-memory add with auto-incremented addresses), and
- special privileged instructions to control the 14 DMA ports.

A summary of the instruction set is shown in Figure 3-2. The cycle counts shown for each instruction reflect its execution time after all operands have been fetched. The operand fetch times are indicated for each addressing mode, but do not include DRAM access time discussed above.

Interrupts and kernel traps

The nCUBE/2 has individually vectored interrupts and kernel traps. Each one is quite expensive: saving and restoring the PC, the PSW, and registers used by the kernel takes many memory accesses and the trap or interrupt itself costs a dozen of cycles because the change of privilege level breaks the instruction prefetch and decode pipeline.

Calling convention

The calling convention used on the nCUBE/2 treats registers R0-R5 as caller-saves, registers R6-R13 as callee-saves, and R14-R15 are reserved. Up to four arguments are passed in registers: the *n*-th argument is passed in a register provided that argument *n* fits into a single register, and that all previous arguments are passed in registers. Arguments not passed in registers are pushed onto the stack and the return address is also pushed onto the stack by the `call` instruction.

Instruction format::

```
        <label>:    <op><size>   <src1>,<src2>,<dst> ! comment
```

Examples:

```
        l1:         movb         R0,R1               ! move byte from R0 to R1
```

Regular op-codes†		cycles	
		int	fpu
mov	move	1	
mvu	move unsigned	1	
add	add	1	7
sub	subtract	1	7
neg	negate	1	1
sbr	subtract reverse	1	7
cmp	compare	1	6
bit	bit test (and)	1	
mul	multiply	4–11	6–10
div	divide	14–38	7–31
rem	remainder	13–38	6–69
sft	shift logical	4	
sfa	shift arithmetic	4	
rot	rotate	3	
ffo	find first one	3	
and	bitwise and	1	
or	bitwise or	1	
xor	bitwise excl. or	1	
not	bitwise negate	1	

†. Not all opcodes are listed.

Addressing modes		cycles
r*N*	register	2
(r*N*)	reg indirect	4
-(r*N*)	auto decrement	5
(r*N*)+	auto increment	5
o(r*N*)	reg+offset indirect	4
o(sp)	stack relative	4
stk	push/pop stack	5
#*N*	immediate	2
N	absolute	4

Control-flow†	
b	branch always
be	branch equal
bne	branch not equal
bg	branch greater
bge	branch greater equal
bl	branch less
ble	branch less equal
jmp	jump
call	call (push PC on stack)
ret	return (pop PC from stack)
trap	trap (push PC&PSW)
reti	ret from trap (pop PC&PSW)

†. cycle costs not specified

Operand sizes	
B	signed byte
UB	unsigned byte
H	halfword
UH	unsigned halfword
W	signed word
UW	unsigned word
D	signed doubleword
UD	unsigned doubleword
X	BCD
R	32-bit IEEE float
L	64-bit IEEE float

Figure 3-2: NCUBE 6400 instruction set summary.

The NCUBE 6400 instruction set is very similar to the DEC VAX11 instruction set. High regularity allows almos all op-codes to be combined with all operand types and all addressing modes. The instructions are encoded in byt tokens very similar to the VAX; every operand requires a descriptor byte followed by the required offsets or values Most instructions are available in two-address and three-address formats. The cycle times shown are approximate the execute unit operates in parallel with the instruction fetch and decode unit.

3.1.3 Communication micro-architecture

The nCUBE/2 communication micro-architecture is built around a bit-serial binary hypercube network. Figure 3-3 shows the organization of the hardware support for the network found on each processor: 14 bidirectional network ports which are directly connected to neighbor processors in the hypercube, a router which forwards messages among network ports, and 14 input/output DMA engine pairs connected to the respective network ports.

Network topology

The 14 bidirectional network ports are designed to be directly connected to the ports of neighbor processors in the hypercube. Of these ports, 13 are used to interconnect all computational nodes and the 14th port is reserved for connections to I/O processors[2]. The 13 ports allow a maximal system size of 8192 processors, in systems with fewer processing nodes the high-order connections are simply left unconnected. The 14 network ports are connected to a wormhole router which forwards messages automatically to the appropriate output port and to 14 input/output DMA engine pairs, one for each network port. The 28 DMA engines operate independently but share access to the DRAM interface with the processor.

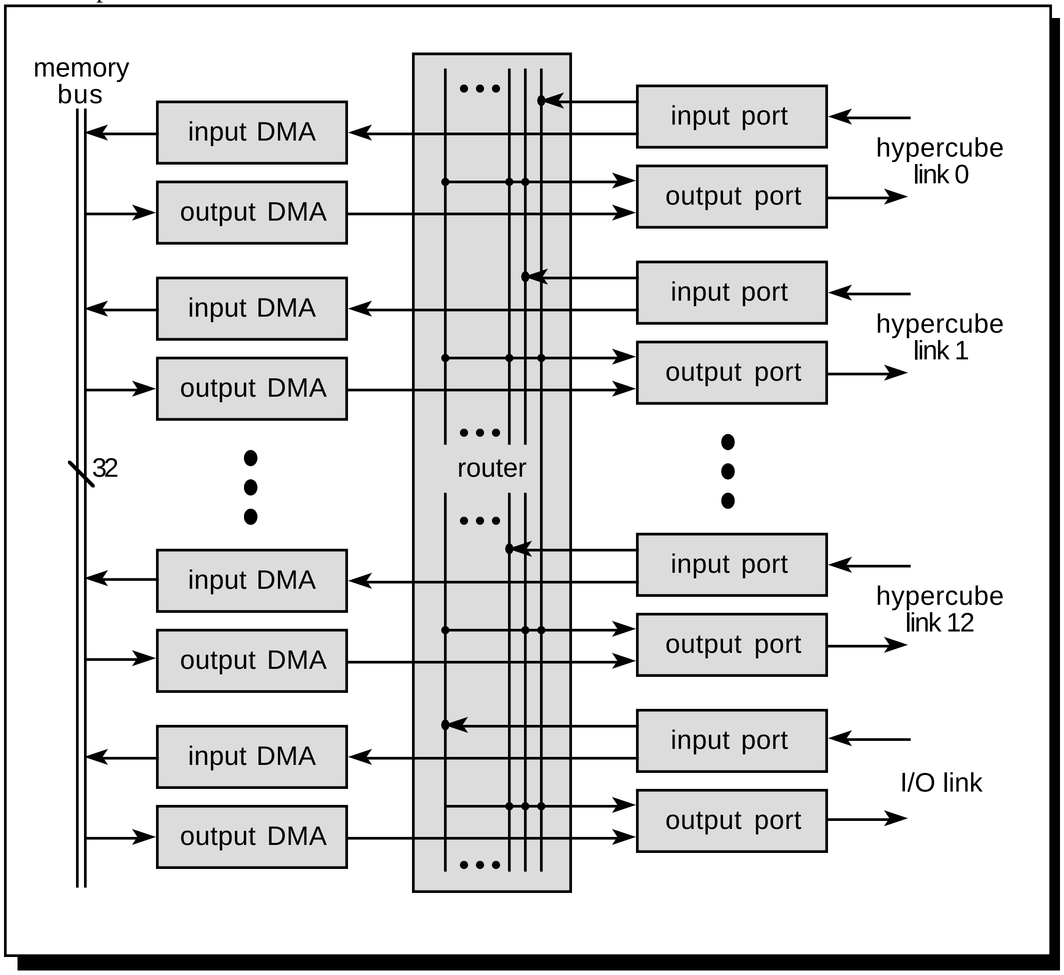


Figure 3-3: nCUBE/2 network interface organization.

[2] I will ignore the I/O port as its operation is identical to that of the other 13 except for peculiar routing details.

Network channels

The network operates on bit-serial data streams and each port consists of two unidirectional bit-serial channels operating at 20 MHz. Communication in each direction is in units of 36-bit flits (flow control units) holding 32 bits of data, a parity bit and flow-control information for the channel in the reverse direction.

Message format

The network provides inter-processor communication in units of messages. A message consists of an arbitrary number of 32-bit words, the first word being interpreted by the network as destination node address. No further header information is recognized by the hardware, thus the receiver does not know the source node address nor the message length.

Network volume

In the case of contention, i.e., when the output port is busy, the router simply blocks the incoming message. At that point, a small buffer can hold up to 2 flits until flow control reaches the up-stream router and prevents further flits from being sent. The message thus backs up through the network blocking the intermediate network links it occupies and eventually stalling the source DMA engine.

Transmission mechanism

Messages can only be sent and received using the DMA engines, no programmed I/O access to network FIFOs is provided. Each message originates in a memory buffer on the source node, is injected into the network by a DMA engine, routed through the network to the destination node, and transferred into a memory buffer by another DMA engine.

To send a message, the output DMA engine corresponding to the appropriate network port must be set-up with the message start address and length. The DMA engine then fetches the message autonomously from memory using quadword accesses, serializes each word and transmits it on the outgoing port. The DMA simply stalls if the downstream router sends negative flow-control back (e.g., stops sending acks back).

Message reception is somewhat “blind” in that an input DMA engine must be set-up with a base address before any message can be received from the corresponding network port. Any incoming message is then transferred to memory starting at that address without any message length limit being enforced by the DMA engine.

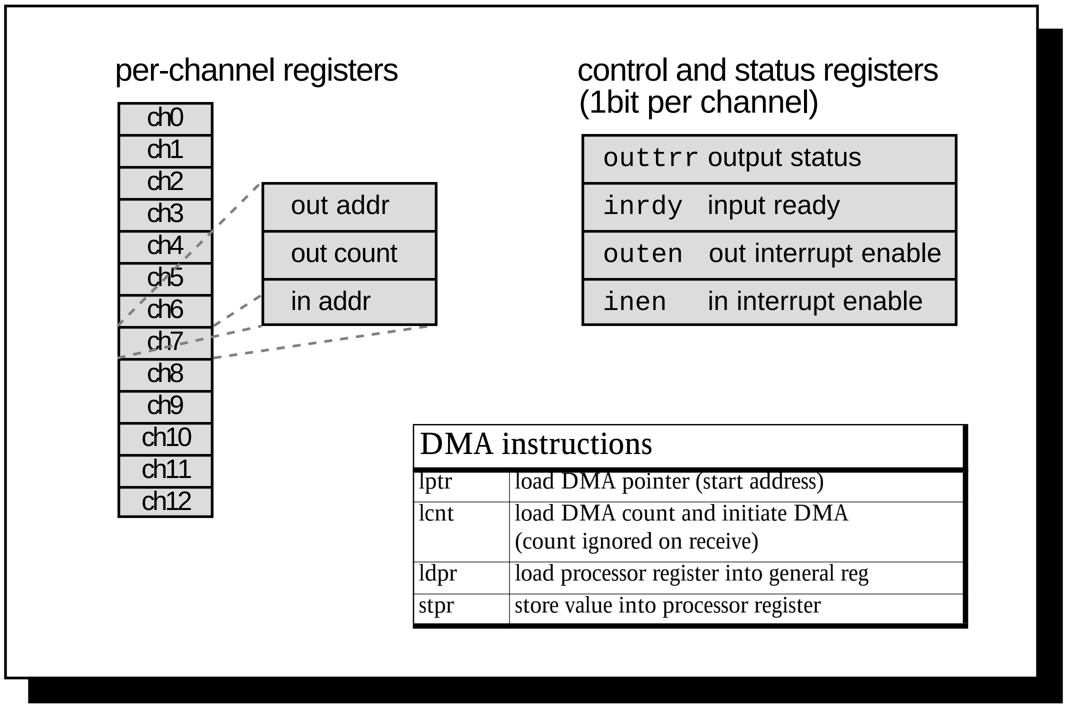
The DMA engines also support non-contiguous messages in memory allowing, for example, message headers to be stored separately from the message data. For this purpose, messages can be subdivided into segments such that the transfer of each segment corresponds to an independent DMA operation. All segments of a message use the same route through the network and arrive back-to-back at the destination.

Network status and events

The network status is mainly available though DMA completion events. When an output DMA engine completes a transmission it sets a bit in a status register and optionally generates an interrupt. Similarly, when an input DMA engine receives the last word of a message it sets a status bit and optionally generates an interrupt.

The DMA status registers are organized as bit vectors with one bit per channel. The input and output ready status bits indicate the completion of a DMA operation. The input message ready status bits indicate the completion of the DMA for the last segment of a message.

Interrupts are vectored individually for each input and output DMA channel. The interrupt enable registers hold an enable bit for each channel and, in addition, the PSW allows all input or all output interrupts to be controlled collectively.



DMA instructions	
lptr	load DMA pointer (start address)
lcnt	load DMA count and initiate DMA (count ignored on receive)
ldpr	load processor register into general reg
stpr	store value into processor register

Figure 3-4: nCUBE/2 DMA instructions and registers.

Each bi-directional DMA channel state consists of three registers: an input address to store the next word received, an output address from which to fetch the next word to send, and a send count. The control and status registers are bit-mapped, i.e., have one bit for each channel, and allow a send completion and receive completion interrupt to be enabled for each channel. The interrupts are individually vectored.

Figure 3-4 summarizes the registers and instructions controlling DMA operation.

Routing

Messages are automatically routed through the hypercube interconnect using wormhole routing: each 36-bit flit is forwarded 8 cycles after having been received. The output port is chosen using an e-cube deterministic routing algorithm which corrects the low-order node address bits first. This algorithm is deadlock-free because it monotonically orders network links.

At the originating node, messages must be injected by the DMA engine corresponding to the appropriate network port, that is, the port corresponding to the lowest bit differing in the source and destination node addresses. At the destination node the router forwards the message to the DMA engine corresponding to the input network port.

The network is reliable in that it never drops or misroutes messages. In addition, each 36-bit packet is parity-checked. Parity errors are considered fatal.

Although the on-chip router uses a fixed oblivious routing algorithm, it is possible to exert some control over the route using forwarding addresses: setting a bit in the address word causes the router at the destination node to “peel-off” the address word and to use the second word as new address. Note that forwarding messages in this manner can cause deadlock in the routing. It is mainly used to forward messages to and from I/O nodes.

Mask bits in the address word can cause address bits to be interpreted as “don’t cares” with the result that the message is broadcast in hardware to multiple processors. The problem with this hardware broadcast is that a router acquires the required output ports as soon as they become available but for-

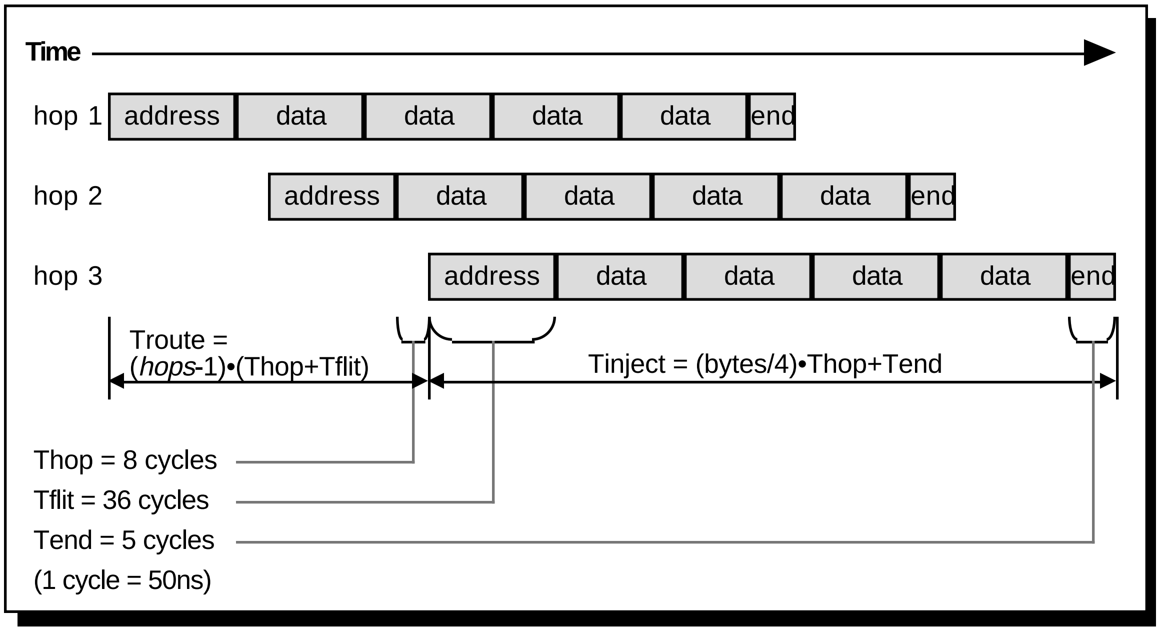


Figure 3-5: nCUBE/2 network timing.

The nCUBE/2 router forwards packets using wormhole routing along bit-serial channels. Each flit (flow control unit) is 36 bits long and holds 32 bits of data as well as flow control and a parity bit. Each message is terminated by a short end-of-message flit.

wards the message only when all output ports are available. This algorithm can cause deadlock and is only useful when the state of the network is well defined, such as at boot time or program load time.

The nCUBE/2 router forwards packets using wormhole routing along bit-serial channels. Each flit is 36 bits long and holds 32 bits of data as well as flow control and a parity bit. Messages consist of an arbitrary number of data flits and are terminated by a short end-of-message flit. As illustrated in Figure 3-5, the latency through each router is 44 cycles.

Protection mechanisms

The instructions controlling the DMA engines are privileged, thus access to the network is strictly reserved to the kernel. The destination node address is not checked by the hardware and the kernel is responsible for preventing user processes to send messages to nodes outside its address range.

In addition, the DMA engines operate with physical memory addresses. User virtual addresses for message buffers must be translated by the operating system, which due to the simple segment-based memory mapping is not too problematic given that this guarantees the virtual to physical mapping to be contiguous.

Network context

The network does not support the notion of state that can be saved. Messages in transit can only be retrieved by the destination node using the normal DMA operations. The only (undocumented) special support available is that setting the input DMA address to –1 causes the DMA engine to drop the next incoming message segment.

3.1.4 Summary

The nCUBE/2 communication micro-architecture is memory-to-memory and requires all messages to be handled and checked by the kernel to enforce protection. To send a message, the user-level process must compose the message in a memory buffer and trap to the kernel. The kernel then checks the message destination and length and sets-up the appropriate outgoing DMA engine. If that DMA is busy, the kernel can either queue the message or suspend the user process. Queueing typically requires setting up an interrupt at the end of the current transfer which services the queue. If the user process is blocked, network deadlock issues must be considered carefully.

On the receiving side, the kernel must set-up all DMA engines to transfer arriving messages into a set of buffers. After a message is transferred to memory, an interrupt is signalled by the DMA engine and the kernel can inspect the message header. At that time, the kernel typically restarts the DMA for the next message. Messages destined for user processes can then either be copied into the appropriate address space or can be mapped using one of the four data segments.

3.2 The CM-5 Micro-Architecture

The Thinking Machines Corp. CM-5 is a massively parallel processor built largely with workstation technology. The basic philosophy is that the CM-5 network replaces the backplane of a typical mini-computer. Processing nodes and I/O devices all plug into the network which interconnects them, just as the backplane did, but with a higher and scalable aggregate bandwidth.

Figure 3-6 shows a typical CM-5 configuration which uses the network to interconnect processing nodes grouped into several reconfigurable partitions, control processors to provide standard OS services to the partitions, and smaller partitions for scalable disk arrays and high-speed network interfaces.

3.2.1 Node micro-architecture

The current CM-5 processing nodes are not as densely integrated as the nCUBE/2 but offer higher performance. Each node, depicted in Figure 3-7, is very similar to a Sun SPARCstation 2 motherboard. Instead of the typical workstation I/O interfaces the CM-5 nodes have a network interface (NI) and four DRAM memory banks each with a memory controller “augmented” by an on-chip vector unit. The Cypress 7C601 SPARC chipset runs at 33Mhz and achieves a peak of approximately 7Mflops. The vector units integrated on each memory controller operate at a peak of 32Mflops each[3]. Four processing nodes are placed on two sandwiched boards and 32 nodes, including the two lowest levels of the tree, fit into a 19" backplane.

The memory hierarchy on each node consists of the Sparc registers, a 64Kbytes direct mapped unified cache and four DRAM banks. The network interface is connected to the MBUS and is accessed using uncached load and store instructions. The timing for network interface accesses is shown in Figure 3-8: the processor takes the normal 2 to 4 cycles to execute a load or store instruction and the NI completes

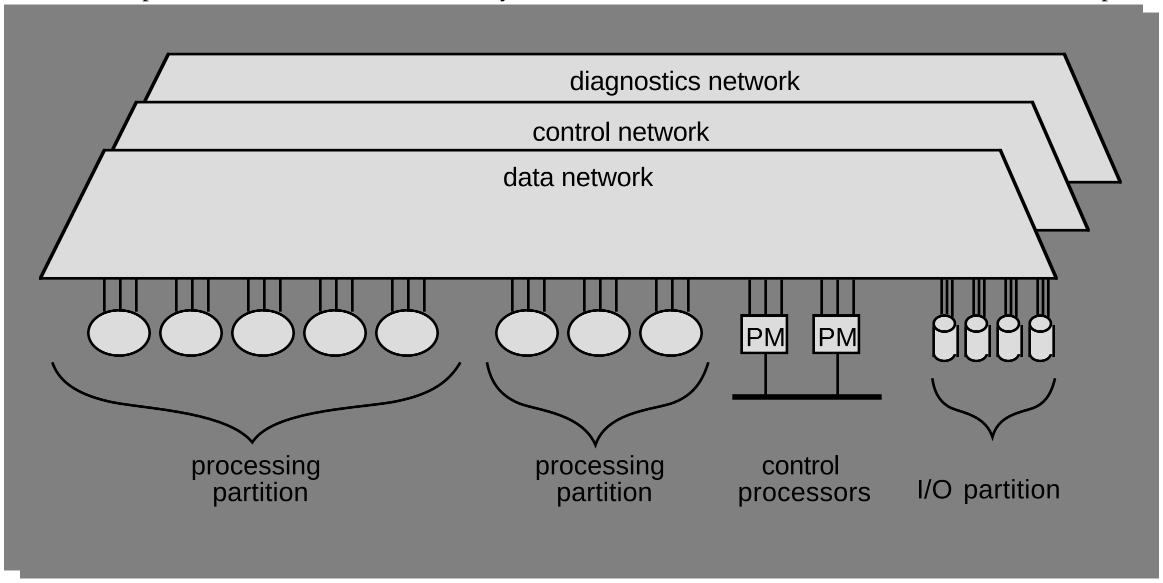


Figure 3-6: CM-5 network backplane.

The three CM-5 networks serve as the backplane of the machine. Each node, be it a an I/O node, a control node, or a processing node, connects into all three networks. The diagnostics network is a modified JTAG boundary scan network and is used to configure the machine and to diagnose faults. The control network supports collective communication, including broadcasts. The data network is the main network and carries short point-to-point data packets.

[3] The CM-5 vector units are largely ignored in this dissertation. While they are important in achieving high computational performance, their use is to a large degree orthogonal to the communication models. Note that taking the vector units into account is important when evaluating the overall balance of the machine as in Figure 1-1.

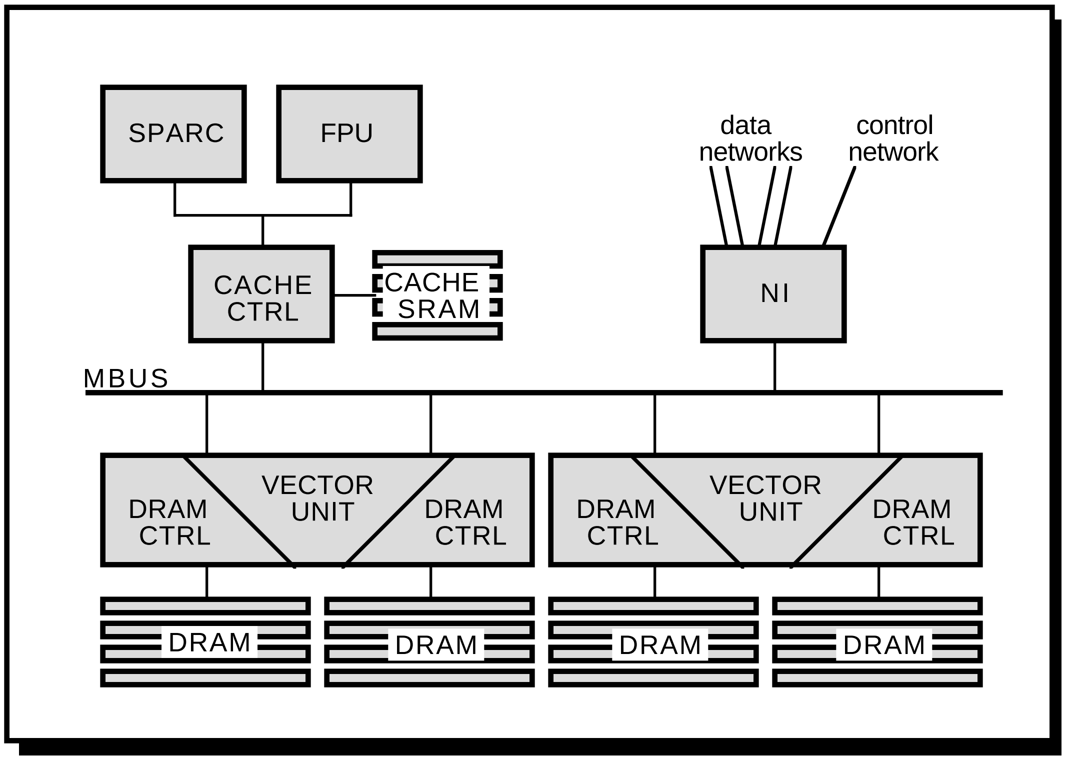


Figure 3-7: CM-5 processing node organization.

A standard Cypress Sparc chipset forms the heart of every CM-5 node. The memory system is special in that each memory controller accesses two DRAM banks and hosts an on-chip vector unit. Each physical vector unit is time-multiplexed among two logical vector units, each of which operates out of one of the DRAM banks. The network interface is attached to the memory bus and links the node into the data, control and diagnostics networks.

an MBUS transaction in 4 to 5 cycles. Note that while stores can be pipelined using the four-entry write buffer, the processor is stalled during the MBUS access cycles executing a load.

The following table summarizes the CM-5 node memory hierarchy:

level	read	write	transfer
registers	0 cyc	0 cyc	32 bits
cache, write buffer†	2/3 cyc	3/4 cyc	32/64 bits
NI	7/8 cyc	7/8 cyc	32/64 bits
DRAM	—	18‡/10 cyc	32/64 bits
	22 cyc	—	32 bytes

†. Single/double-word accesses.
‡. Writes of less than 64 bits require a read-modify-write.

3.2.2 Instruction set architecture summary

A description of the SPARC processor architecture and instruction set is omitted and the reader is referred to [PH90, ROS90].

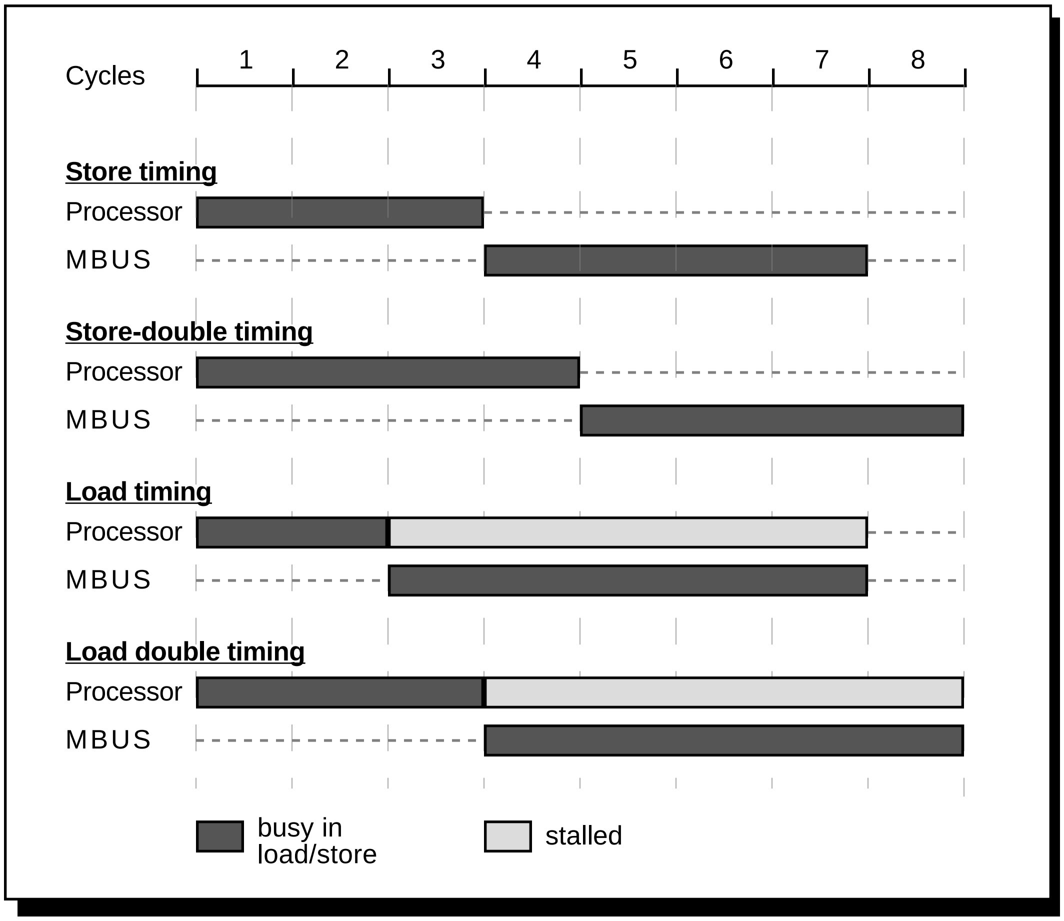


Figure 3-8: CM-5 network interface access timing.

The network interface must be accessed across the 64-bit memory bus (MBUS). Stores to the NI complete in three cycles (four for double words) due to the write-buffer. An MBUS write takes four cycles during which the processor continues instruction execution. Loads from the NI see the full MBUS latency and take seven cycles (eight for double words), five of which are due to the MBUS read cycle.

Calling convention

The CM-5 implementation of Active messages integrates quite closely with the Sparc register windows and calling convention. In preparation for the discussion of the Active Messages implementation in Section 4.3, the following paragraphs summarize the calling convention to the extent relevant. Further information can be found in the Sparc architecture specification[SPA91].

The Sparc uses a circular register file subdivided into overlapping register windows [Kat83]. Each register window consists of 24 registers, of which eight overlap with the next window, eight overlap with the previous window, and eight are private. In addition, 7 global registers are shared among all register windows and register zero is hard-wired to the value zero.

The registers in the current window are denoted i0 through i7 for those overlapping with the caller's window, o0 through o7 for those overlapping with potential callees, l0 through l7 for the private ones, and g0 through g7 for register zero and the global registers.

According to the calling convention the first six words of arguments are passed in registers i0 through i5 (as seen by the callee) and the return address is passed in register i7. Additional argument words are passed on the stack. Note that floating-point values are passed in the integer registers and that complex data types are never passed in registers.

3.2.3 Communication micro-architecture

The CM-5 contains a total of three networks. The data network provides high performance point-to-point communication among all nodes in a CM-5 system. Active Messages uses exclusively the data network which is described in more detail in the subsections below.

The control network supports cooperative communication functions among groups of nodes, such as barrier, broadcast, or integer reduction. The control network plays a number of roles, from accelerating data parallel computations to supporting system functions. Data parallel programming models benefit from the fast synchronization of all processors and from the global broadcast, reduce, and scan operations provided by the control network. Because the control network requires the participation of all processors in a partition[4] it is quite specific to this programming model. Its hardware cost is small, however, in relation to the data network.

In addition, the control network is used by the operating system to coordinate the kernels on all nodes of a partition. The control network prioritizes operating system operations and allows the partition manager to broadcast commands and interrupts to all nodes in a partition. This, in effect, provides the operating system with a limited but highly useful communication path external to the data network.

The third network in the machine, the diagnostic network, is used to bootstrap, test, and monitor the machine. It provides a scan path through most components of the machine.

Network topology

The network uses a 4-ary fat tree topology illustrated in Figure 3-9. Internal nodes in a 4-ary fat tree are connected to four child nodes and four parent nodes. This keeps the aggregate bandwidth constant from one level to the next of the tree. The CM-5 network deviates from a normal fat tree in that the

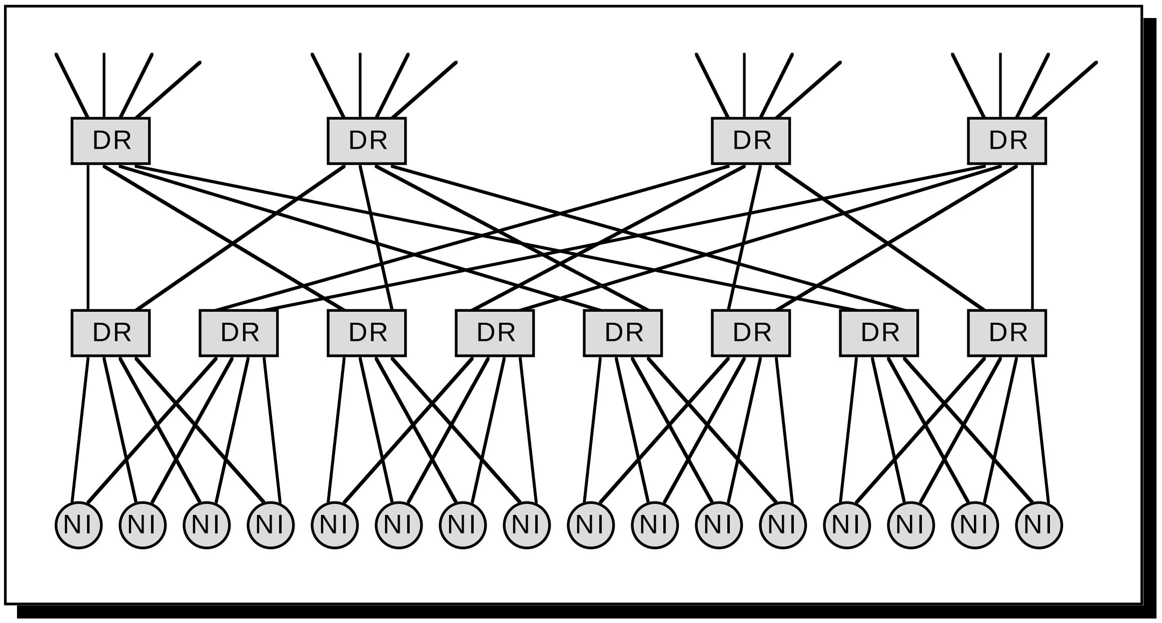


Figure 3-9: CM-5 fat tree data network topology.

The CM-5 data network uses a four-ary incomplete fat-tree topology. In general, each router is connected to four parent anc four child routers, however, each network interface at the leaves of the tree and each of the lowest level routers is connected tc only two parent routers.

[4] While individual processors can abstain from a control network operation, it is not possible to use the control network among processor subsets without interference.

two lowest levels of the tree are incomplete: the leaf nodes and the lowest internal nodes have only two, instead of four, parent nodes. In addition, the CM-5 data network consists of two independent identical fat trees, informally called the “left” and “right” networks.

At the heart of the data network micro-architecture are two chips: the network interface (NI) and the data router (DR). As shown in Figure 3-9, a DR is placed at every internal node of the fat tree data network and connects four child nodes to four parent nodes (except at the lowest level where it connects to only two). At the leaves of the tree the NI connects processing and I/O nodes to two routers in each of the two data networks[5].

Network channels

Each network link contains 4 data wires in each direction and transmits 4-bit flits at 40 MHz for a raw 20 Mbytes/s bandwidth per link. Each message data word as well as the CRC code are transmitted in 8 flits, and the message length and tag use 1 flit each. The length of the destination address depends on the number of levels the message must climb in the tree. As a result, the longest message in a 1024-node machine uses 48 flits and takes 1.2µs to be injected into the network. The net link bandwidth for such messages is a little over 16 Mbytes/s.

Messages

The network provides inter-processor communication in units of messages. A message consists of a 32-bit destination node address followed by up to five 32-bit data words. In addition, each message carries a 4-bit tag which is typically used to distinguish different types of messages. On reception the destination node address is stripped from the message. The message tag and the message length can be queried, however, the source node address is not known.

Transmission mechanism

On the processing nodes, the NI is attached to the 64-bit MBUS. The NI provides access to the network via four memory mapped FIFOs. The processor must perform a sequence of stores to the FIFO in order to send a message and similarly a sequence of loads to receive one.

The network interface is an MBUS slave device and the processor has to transfer all data explicitly into and out of the NI. As shown in Figure 3-10 the NI contains an input and an output FIFO for each data network. To send a message the processor performs a series of single or doubleword stores into the output FIFO corresponding to the desired network. The first store is to a special `send-first` register transfers the destination node address and, in the case of a doubleword store, the first data word. In addition the low-order address bits encode the message length and tag. Additional data words are stored to the send-data register and after the last word is stored the message is, in principle, launched into the network. However, if at any point the output FIFO is full, the network interface simply drops the message and it must be retransmitted by the program.

To receive the message the processor can read the message tag and length from the status register and load the message data from the appropriate network input FIFO. Note that the destination address is stripped from the message and cannot be read.

Network status and events

Because the network interface can drop a message being injected into the output FIFO, the program must check the `send-ok` bit in the status register after the injection and retransmit the message if necessary. If the status register indicates that the message has been sent then transmission is ensured and the message is guaranteed to eventually reach the destination node.The arrival of a message is indicated by a bit in the status register and can, in addition, be signalled by an interrupt. The status register layout is shown in Figure 3-11.

[5]. The NI also connects into the control network and into a diagnostics network.

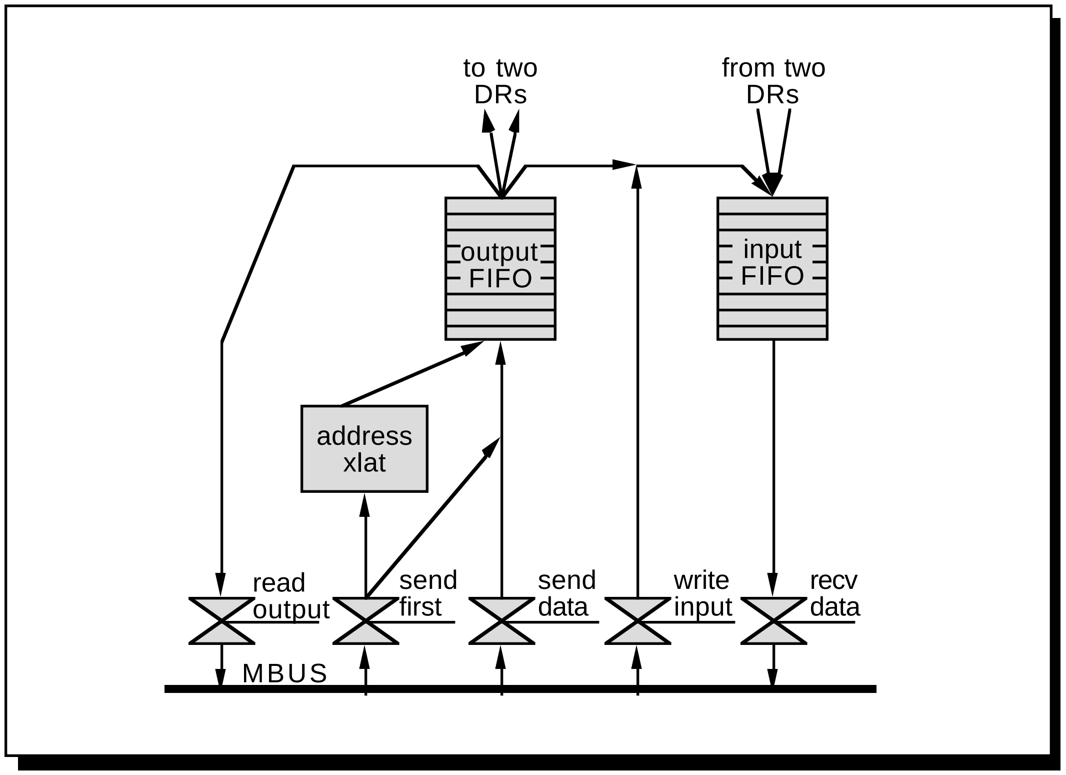


Figure 3-10: CM-5 network interface organization.

For each data network (and control network) the network interface provides an input and an output FIFO. Send-ing a message involves pushing the message a word (or double-word) at a time into the output FIFO. The first worc pushed is the destination node address and is translated from a logical address within the partition into a physica node address. The translation enforces protection boundaries and allows faulty nodes to be mapped out. To receive a message must be read a word (or double-word) at a time from the input FIFO. Note that the network interface loops messages to the local node back into the input FIFO.

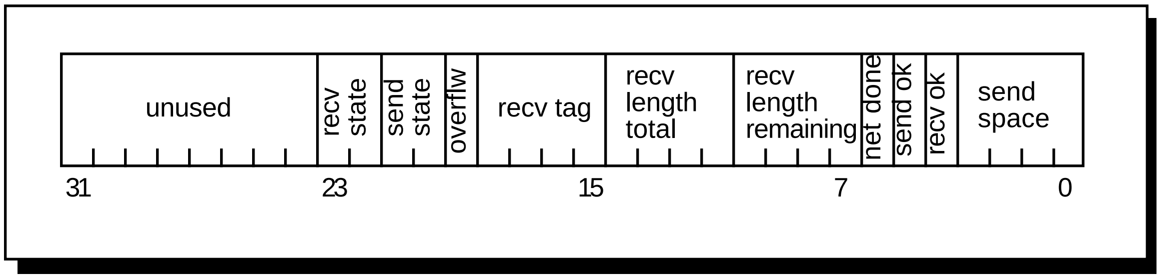


Figure 3-11: CM-5 network interface status registers.

The CM-5 network interface status register contains numerous fields giving information about the status of the input anc output FIFOs. However, only the send_ok, receive_ok, receive_tag, and possibly receive_length fields are generally used. The send_ok bit indicates whether the last message pushed into the output FIFO was sent successfully. The receive_ok indicates whether a message is present in the receive FIFO. When receive_ok is set, the receive_tag and receive_length fields indicate the tag and the length, respectively, of the message. Note that the send_space indicates the current space in the outgoing FIFC but does not give any guarantee that a send attempt would succeed.

Routing

The CM-5 network uses cut-through routing with randomized decisions among the multiple possible paths. Each message injected into the network is routed to one of the common parent nodes of the source and the destination leaf nodes. On the path up the tree each DR routes the message to one of the four possible up-links choosing randomly among the idle ones. Once the message reaches the common parent node only a single path down leads to the destination.

Measurements show that each router takes 8 cycles to route a message if the network is idle, thus the maximal latency (without contention) on a 1024-node machine is 1.8µs. The bisection bandwidth through the root of the tree is guaranteed to be at least 5 Mbytes/s (of data payload) per processor per data network, calculating this figure is not straightforward and requires careful analysis of the routing and queueing details.

Due to the high bisection bandwidth of the fat tree and due to the randomized routing which avoids hot-spots within the network the network topology can be ignored to a large degree in the implementation of algorithms that avoid destination hot spots. However, the constriction in the two lowest levels of the tree favors communication patterns with some degree of locality.

The randomized routing among the multiple equivalent paths has a major drawback however: messages injected into the network one after another can reach the destination out of order by experiencing varying delays due to contention. When breaking up large messages into small ones one therefore cannot rely on the message arrival order for reassembly.

Network volume

Given that the message injection time is quite close to the routing time through an unloaded network (1.2µs vs. 1.8µs) the network volume is quite low when operated below saturation. Each processor, even if sending messages back-to-back can barely complete injecting a second message when the first one reaches its destination. Under contention, however, the input buffers on each link in the DRs absorb the tails of blocked messages. Each input buffer can hold approximately one message and in a 1024-node machine these buffers can hold roughly 10 messages per processor on average.

Protection mechanisms

An often overlooked innovation in the CM-5 is the ability to provide direct protected user-level access to the network interface and to time-share the network among multiple parallel processes.

A novel feature of the CM-5 NI is that it allows controlled user-level access to the network: the network input and output FIFOs and various status registers can be mapped into the user process without compromising system security. An address translation map ensures that the user process can only send messages to nodes within the partition and special control over the internal operation allows the kernel to context switch from one process to the next.

When the user process sends a message the NI performs an address translation on the destination node address which allows the kernel to limit the possible destination nodes to those in the current partition[6]. The kernel can circumvent the address translation to send messages to control processors or I/O nodes.

Because the NI contains only a single output FIFO per data network the kernel must be careful when sending a message. If a partial message has been pushed into the FIFO by the user process the kernel can flush the partial message by setting a `send-lock` bit in a privileged control register. Flushing the user message clears the `send-ok` flag which eventually causes the user process to retransmit the lost message. While the implementation could have provided two distinct user and kernel FIFOs it was judged better to provide a single FIFO of twice the size which improves the average case and penalizes the exceptional one.

[6] The address map also allows spare nodes to be mapped into a partition to replace failed nodes.

Message arrival presents a similar problem due to the single input FIFO. All messages arrive into the same input FIFO and can be read by the user process. In order to prevent the user process from intercepting messages, the kernel must ensure that all its messages cause an interrupt on arrival so that it can read the message before the user process could. The generation of interrupts can be controlled individually for each message tag. The kernel thus reserves a set of tags for its own use by setting a mask register preventing the user from using these tags and forces all reserved tags to cause an interrupt on arrival. In the current NI, unfortunately the user process can detect the arrival of a kernel message due to the latency of the interrupt. This means that the user process must check the message tag in order to avoid reading the, by then empty, input FIFO which causes an exception.

Network context

The time-sharing supported in the CM-5 is limited to gang-scheduling parallel processes within each partition, i.e., all processors in the partition always switch to the same parallel process at the same time. Time-sharing the network is conceptually quite simple: in addition to saving and restoring the processor state the kernels on all nodes cooperate in saving and restoring the entire network state at each context switch.

In order to save the state of the network the data routers are put into a so-called “all-fall-down” (AFD) mode in which each message in transition is routed to the “closest” processing node instead of its destination. The AFD mode prevents that, in the worst case, a single node has to empty the entire network, which would take a few seconds on a large machine and would require reserving a large worst-case buffer on each node. With the AFD mode all nodes participate in saving the network state.

To start the next process, each kernel injects the previously saved messages, sending them to their original destination. Given that the messages cannot be re-injected into the same DR that originally held them[7], the kernel may not be able to inject all saved messages. In such a case the user process must be restarted in order to receive messages and allow the remaining messages to be eventually injected. The user process can be prevented from sending during this phase to avoid more and more messages from accumulating in the kernel’s buffer at every context switch.

3.2.4 Summary

The CM-5 uses a register-to-register communication architecture in which the processor is involved in all message transfers. The network interface itself is memory mapped and contains protection mechanisms so it can be accessed directly by user processes. To send a message, the user process stores the destination address followed by the message data into an outgoing network interface FIFO. After storing the last word, a send-ok status must be checked to determine whether the network has accepted the message.

Message arrival can be detected either by polling a status register or via an interrupt. In both cases the message is received by loading the message data from an incoming network interface FIFO. Note that while the FIFO is directly accessible from user space, the interrupt is dispatched via the kernel and cannot be enabled/disabled by the user process.

[7]. For example, a message in transit from node 1023 to node 0 might “fall down” from a root node of the tree onto node 16. When node 16 re-injects the message, it will simply travel two levels up the tree and back down to node 0; it will not go through the root again. In general, after an all-fall-down, messages are closer to their destination and will not travel as high up the tree when they are re-injected, thus more messages than before the all-fall-down will content for the buffers in the lower level of the tree.

3.3 Conclusion

The two communication micro-architectures presented in this chapter are at first sight quite different. The nCUBE/2 has a memory-to-memory micro-architecture with DMA support for unlimited-size messages while the CM-5 register-to-register micro-architecture places a stringent 5-word limit on messages. A consequence of this difference is that message transmission and reception are semi-autonomous on the nCUBE/2 while the network interface is a pure slave device on the CM-5. This means that the network is a little more decoupled from the processor on the nCUBE/2 but it is also more difficult to control. For example, incoming messages are transferred into memory autonomously, but cannot easily be stored directly into the right locations.

At the system level, the two micro-architectures differ in their support for virtualizing the network. The nCUBE/2 places all responsibility onto the kernel which must multiplex the network among messages from user processes and its own messages. Enforcing protection on message send and receive relies on inspection by the kernel. In addition, the hardware does not readily allow to virtualize the network state. There is no support to save the network state other than receiving all message in transit by normal means, there is no way to stop an ongoing DMA transfer, and there is no communication path reserved for the operating system.

The CM-5 micro-architecture allows the network to be virtualized as long as parallel processes are gang-scheduled. The hardware enforces protection boundaries by limiting message destinations and by distinguishing user and kernel messages (using the 4-bit tag). In addition, the all-fall-down mode allows the operating system to context swap the network and the network interface FIFOs can be saved and restored as well. The control network supports special kernel broadcasts which can interrupt all processors. Albeit limited, this provides the operating system with a communication path external to the data network used by the user processes.

While these differences are interesting, the common elements in both micro-architectures are really more important. In both machines the message format is simply the destination address followed by a number of data words on which the hardware imposes no particular interpretation.

Message arrival is indicated by a flag in a status register and optionally by an interrupt. This allows both polled and interrupt-driven communication models to be implemented. However, the interrupts are dispatched into the kernel and neither micro-architecture provides any special support for forming critical code sections or data structure updates that are atomic relative to communication interrupts.

The network routing algorithm in both machines is deadlock-free as long as processors keep receiving messages. In particular, both networks require processors to accept an arbitrary number of incoming messages when attempting to inject a message. This property is shared by almost all networks used in multiprocessors.

Four key issues

The two communication micro-architectures discussed in this chapter pose a number of challenges to the design of communication architectures. Most of these challenges, however, revolve around the same four key issues:

- efficient *data transfer* in and out of the network,
- *synchronization* of message arrival with the computation,
- handling of *send failure* due to contention without deadlock, and
- *network virtualization*.

On the nCUBE/2 the key to efficient data transfer is minimizing the number of data copies between the application data structures and the buffers accessed by the DMA engines while on the CM-5 it is the ordering of load and store instructions accessing memory and the network interface.

Efficient synchronization of communication and computation on the nCUBE/2 boils down to minimizing the number of kernel traps and interrupts, and coordinating the action of interrupt handlers with the computation. Polling is not a real option because each poll requires an expensive kernel trap. Interrupts on the CM-5 pose the same challenge, however, polls can occur in user mode and offer a real alternative which solves atomicity issues trivially at the cost of requiring periodic polls to be inserted into the entire program code.

On both machines a send may fail until an arbitrary number of messages has been received. On the nCUBE/2 either receives must be handled autonomously by the kernel or control must be handed back temporarily to user level. The situation is simpler on the CM-5 in that the kernel is not involved. However, because the network interface simply drops messages it cannot inject into the network, the user-level program must be prepared to receive incoming messages and retransmit the outgoing one.

Virtualizing the nCUBE/2 network is challenging in that the micro-architecture provides no support. To minimize message handling overhead, protection enforcement must be kept simple. In addition, to prevent the user from clogging the network, careful buffer resource management is required to ensure that all messages in transit can be absorbed. The CM-5 hardware does provide support but requires the operating system to use strict gang-scheduling of parallel processes and to space partition the machine quite statically.

While this discussion of the four key issues has focused on the two micro-architectures presented in this chapter, the situation is very similar on almost all other communication micro-architectures. It appears appropriate to declare these to be *the* four key issues to be addressed in any efficient, versatile, and incremental communication architecture.