The Connection Machine[1] Message Router

Brewster Kahle, Bradley C. Kuszmaul[2], and W. Daniel Hillis
April, 1989
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142

Abstract: The Connection Machine message router is described. A Connection Machine computer system with 65536 processors physically fits within a 2 meter cube. The Connection Machine interconnection network is designed to deliver ten thousand to ten million short (about 32 bit) messages at a time with a throughput of about $2 \cdot 10^8$ messages per second. The network is wired as a variant of a hypercube. The routing of messages through the network is performed on-line, i.e., the destination addresses of the messages are not known in advance. The routing algorithm, which is supported in hardware, is adaptive and combines the advantages of circuit-switched with packet-switched systems to achieve near optimal use of the wires in the network. This paper describes the implementation and performance of the message router used in the Connection Machine Model 1.

1 Introduction

Large scale parallel computers, such as the Connection Machine computer [Hil85], need good message routing systems in order to support a wide range of parallel algorithms. Such routing systems must support a wide variety of communication needs, such as those required to implement shared memory [Sch80], or such as those required to implement explicit communication between data elements [HS86].

A fully configured Connection Machine computer has 65536 processors and physically fits within a two meter cube. The system runs synchronously, and all of the processors execute the same instruction stream, (the Connection Machine computer is a Single Instruction Multiple Data (SIMD) computer[?]). In keeping with the synchronous SIMD design of the system, when a routing instruction executes, all messages are delivered before the next instruction is executed. This paper describes the message routing system for the Connection Machine CM1 computer, although in Section 4 the additional features present in the model CM2 are briefly described . The Connection Machine computer and its routing system are further described in [?,Hil85].

A *routing pattern* is a set of messages M, where each message has a source and a destination. The *router* in a parallel machine must deliver each message from its source to its destination. Some parallel algorithms use routing patterns which can be computed *off-line*, i.e., the pattern itself is part of the algorithm, and is not affected by the data which is being processed. Such off-line patterns include FFT, parallel-prefix [Ble87], and nearest neighbor communication in a grid. Many algorithms, on the other hand, require *on-line* calculation of the routing pattern, i.e., the destination for any given message is not known until the program is actually running. Any off-line routing pattern can be treated as an on-line routing pattern (where the calculation of the pattern is trivial), and so a parallel computer which efficiently supports on-line message routing is at least as powerful as one which only supports off-line routing. Often networks which support off-line routing are called *permutation networks* [LPV81,Lee87].

However, performing on-line routings can be more complex than performing off-line routings, since e.g., nodes in the network must make decisions about how to route messages using only local information (there is no time to compute much global information). For off-line routings it is possible to perform a lot of global computation in order to determine how to set the switches in the network. It is thus conceivable that an off-line routing strategy would make better use of the network resources than an on-line routing strategy, but it fact, the routing algorithm used in the Connection Machine computer uses the wires in the network nearly optimally, and no routing algorithm (on-line or off-line) could do much better. The most expensive network

[1]Connection Machine is a trademark of Thinking Machines Corporation
[2]Bradley C. Kuszmaul is currently a graduate student at the Massachusetts Institute of Technology

resource in the Connection Machine computer is the wires, since the nodes are implemented in relatively inexpensive VLSI.

Typical message loads consist of ten thousand to ten million short (32 to 64 bit) messages which need to be delivered as quickly as possible. This is in contrast to some other systems which are designed to handle at most a few thousand messages at once. We believe that this is an important design assumption which can help the reader understand our design decisions.

The actual performance achieved by the Connection Machine message router is about $2 \cdot 10^8$ (two hundred million) 32 bit messages per second for typical message loads. Another way to measure the system performance is as a fraction of some theoretical maximum. The Connection Machine typically achieves 90% to 95% wire usage; i.e., 90% to 95% of the wires are productively transmitting a bit on any given clock cycle during the execution of a routing instruction. Of course, this later measure is not an absolute measure, but a relative measure, which should hold constant as the implementation technology improves.

The rest of this paper is organized as follows: Section 2 reviews some of the interconnection networks used in other parallel computers. Section 3 describes the router of the Connection Machine computer. Section 4 evaluates the performance of the Connection Machine routing network and concludes.

2 Previous Work

In order to understand what is good and bad about the Connection Machine router it is necessary to understand the interconnection networks of other parallel computers.

The ILLIAC-IV [BDM*72], the MPP [Bat80], and the DAP [Red73] have nearest neighbor two-dimensional grid routing. For certain routing patterns (such as FFT) it is possible to construct a sequence of nearest neighbor operations to perform the routing movement required, but in practice moving data effectively on machines with no router can sometimes be very difficult [Hor82].

Machines such as the TRAC [SUK*], the BBN butterfly [?], the Monarch [Ret86], the ultracomputer [DGK86] and the RP3 [al85] all have on-line routing networks. Often a distinction is made between shared memory architectures and message passing architectures. This distinction is artificial and has more to do with the way the programmers think about the machine than with the hardware. Shared memory means that the hardware has been specialized to handle messages which look like memory requests (e.g. `READ` and `WRITE` messages). Such specialization could conceivably give better performance or lower cost, but we do not believe that this is the case because the overhead of handling general messages in the Connection Machine computer is effectively zero. The interconnection networks of the some of these computers are multistage cubes (i.e. butterflies), which are almost the same as hypercubes [KS82,WF80]. (Section 4 will discuss some of the engineering tradeoffs of multi-stage cube network vs. the variant on a hypercube that is used in the Connection Machine router.)

The Cosmic Cube [Sei85] and its offspring support nearest neighbor hypercube routing. The software can support an on-line routing algorithm by explicitly shipping messages containing addresses one hop at a time. Apparently the software approach to on-line routing is relatively expensive, since on-line routing hardware support has been recently announced for the Intel IPsC [Intel].

The toroidal routing network described in [DS86] is a two dimensional grid. The design of the toroidal routing network, like the design of the Connection Machine, is predicated on the assumption that wires are the most expensive part of the routing network. The analysis in [Dal87] argues that a low dimensional grid or torus is better than a high dimensional hypercube. It is not clear whether that analysis applies to a machine such as the Connection Machine computer. Section 4 compares the Toroidal router network with the Connection Machine router network in more detail.

Telephone switching networks [Ben62,Clo53] have been used to do message routing on a different scale than the Connection Machine computer. Typically telephone switching networks handle many fewer messages each of which is much longer than the messages presented to switching networks in parallel computers. Recently the digital switching networks are starting to look like hybrids between circuit-switched and packet-switched networks [ITT,Bell77]. These newer networks nonetheless operate in a different problem domain

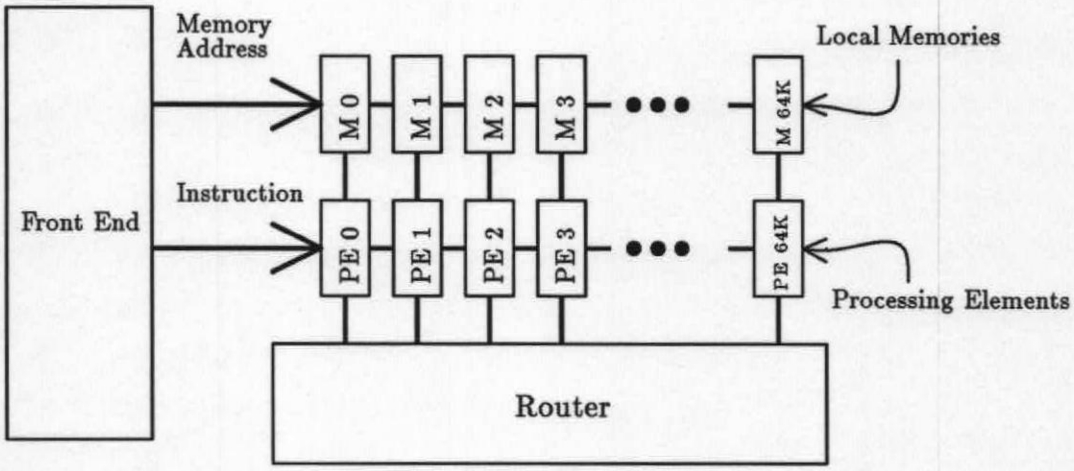

Figure 1: Abstractly, the Connection Machine processors have local memories and a connection to the routing network.

from the other routing networks discussed in this section because, e.g., the network is physically much larger (which means that the switching elements can be more expensive), and the network has real-time requirements (which justifies underusing some of the network resources, such as the expensive switches).

3 Connection Machine Router Implementation

The Connection Machine computer is a Single Instruction Multiple Data (SIMD) computer. The interconnection network is a hypercube with one chip at each vertex and sixteen processors per chip. The synchronous nature of the machine may have a lot to do with its success as a buildable machine. For example, the router hardware is synchronous, which helps control the complexity of the design. This is important because the routing algorithm is relatively complex. In the following subsections, the router is described in a top-down fashion, starting with an block level description of the router and proceeding to each of the components.

3.1 Organization of the Router Hardware

The Connection Machine router is integrated with the processing elements, and is organized as 4096 identical VLSI chips, each containing sixteen processors and some specialized hardware to support routing. Figure 1 shows an abstract view of the Connection Machine computer. Processors have local memories, and a connection to a routing network. In the implementation, part of the routing network is contained in each chip. Each node of the network is implemented on one of the 4096 VLSI chips, which are connected together as a hypercube (see Figure 2).

Figure 3 shows a block diagram of one of the nodes of the routing network. The implementation of a node consists of a VLSI chip and some off chip memory. Logically, each node consists of sixteen processors, and a routing switch. Messages originate at the processors, and end up at the processors. Messages can be stored in buffers, which are implemented in the off-chip RAM.

The details of how data flows through the routing system are as follows. The *injector* combines messages from the buffers and the processors, and sends them to the *heart.* The *injector* sends acknowledgement to those processors which succeed in injecting a message, and the other processors will have to try again later. The messages then move through the *heart*, which successively performs the actual passing of messages across each of the cube dimensions. Messages coming out of the heart are fed into the *ejector*. The ejector removes messages that are at their destination from the system (delivering them to their destination processor),

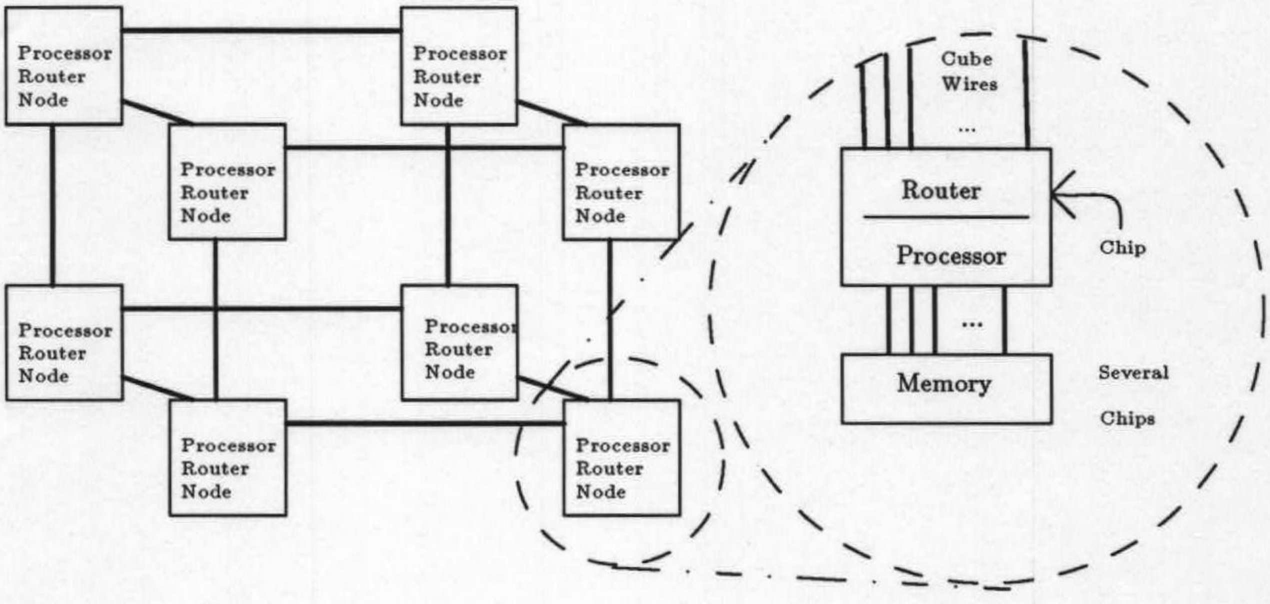

Figure 2: In the Implementation of the Connection Machine computer, the routing network is integrated with, and distributed among the processing elements. The VLSI chips correspond to the nodes of the network, and are wired as a hypercube. Eight nodes of the system are shown.

and places any undelivered messages in the buffers. Then the process repeats (e.g., the processors then get another chance to fill, through the injector, any slots left by empty buffers).

The time it takes to inject messages, route them through the heart once, and eject them is called a *petit cycle* (see Figure 4). It is possible that not all of the messages in a routing pattern can be delivered in one petit cycle, and so the petit cycle is repeated enough times to deliver all messages. (In fact it is possible that not all of the messages will even be injected into the router during one petit cycle.) The time period to deliver all the messages in a message set is called a *delivery cycle.* The number of petit cycles in a delivery cycle depends on the routing pattern.

Note that the only flow control in the system is in the injector. The heart and ejector always do something with all of their messages, and so there is always space for the next batch which come through.

The router is often described as if messages were atomic values which move from one part of the router to another, but in fact this is not so; Messages are pipelined through the various stages of the router in a bit-serial fashion. The pipelining details are discussed in Section 3.4.2. If the messages were atomic values in an unpipelined system, we would need buffering at each stage of system (i.e. the injector, the twelve dimensions of the heart, and the ejector). In reality, the buffers are distributed through the pipeline, but the system is are described as though at each stage of system, the set of buffered messages is modified and sent to the next stage of the system.

The message, as a bitstream being pipelined through the router, consists of the following bits: (See Figure 5)

`MSGP` (1) The first bit of the bitstream says whether the bitstream is a message or not. In our hardware pipeline, there are bits moving from one stage to the next in every bitstream whether or not there is a message in that bitstream or not. The rest of the bitstream is ignored if `MSGP` is zero.

`CUBE-ADDRESS` (12) The next twelve bits of the bitstream are the *relative* address of the destination chip with respect to the chip in which the bitstream is being processed. The relative of chip X with respect to chip Y is gotten by taking the bitwise exclusive or of X and Y as binary integers (written as $X \otimes Y$).

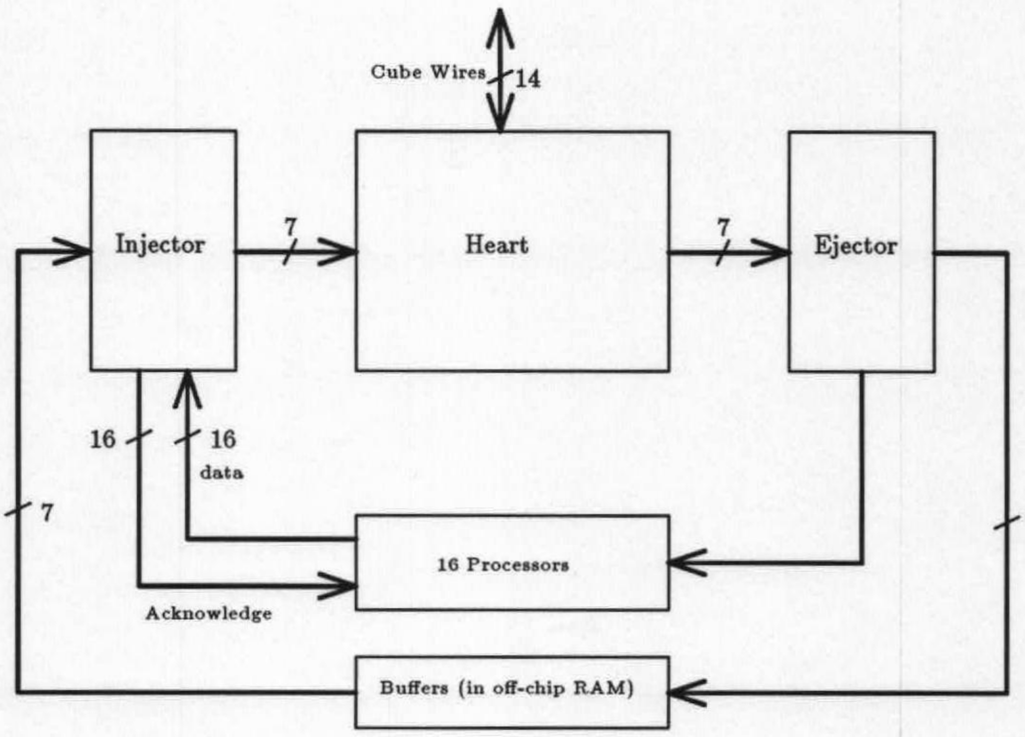

Figure 3: Block diagram of one node of the routing network.

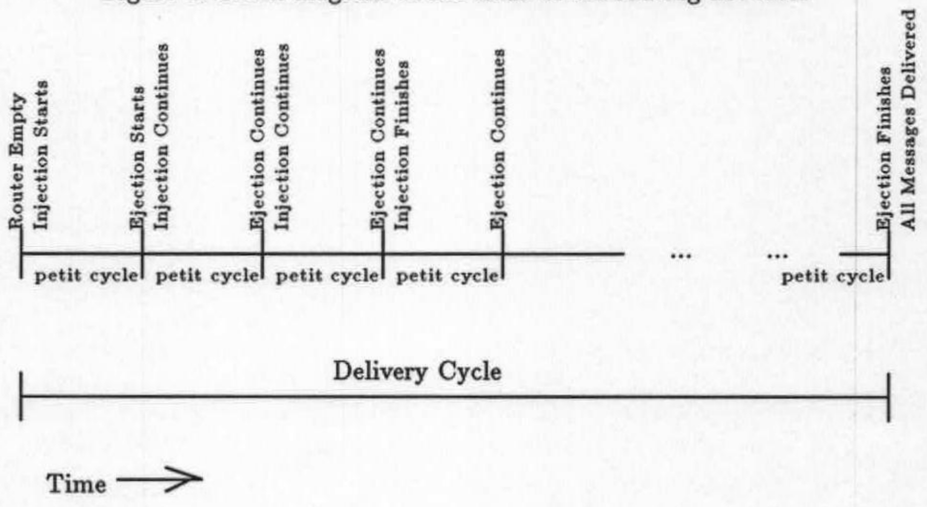

Figure 4: A delivery cycle is the amount of time it takes to deliver all messages. A petit cycle is the amount of time it takes to run data through the data path in a node. A dimension cycle is the time it takes to cross each dimension of the cube. (Note that dimension cycles are overlapped.)

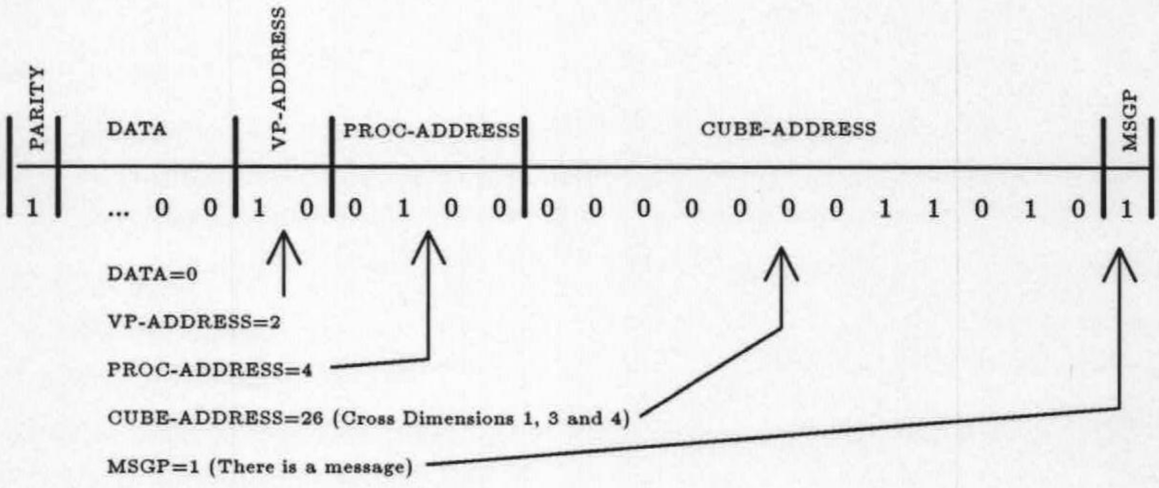

Figure 5: The Message format.

Relative addressing is used so that, for the purposes of routing, all the chips in the machine can think that they are chip zero. This has the advantage that the ejector can test for zero to determine if a message is at its destination and the heart can test for one in the *i*th bit of the `CUBE-ADDRESS` to determine if a message needs to hop across the *i*th cube dimension. Using relative addresses makes message passing slightly more complex in that when a message travels from chip X to chip $X \otimes 2^i$ across dimension i, the *i*th bit of the relative address of the destination changes.

`PROC-ADDRESS` (4) The next four bits say which processor on the destination chip should receive this message.

`VP-ADDRESS` (v) The next v bits (v is under software control) specify the *virtual processor* offset with the physical processor specified by the `CUBE-ADDRESS` and `PROC-ADDRESS`. The Connection Machine computer is designed to allow the user to program the machine as though there are more processors than are physically present in the machine [HS86]. This is done by time-sharing each physical processor among several virtual processors. This is important in many applications (for example, the ability to assign one processor to each transistor in a 200,000 transistor circuit greatly simplifies the programming of the machine, but the Connection Machine computer has only 64K processors in it, so virtual processors are used).

`DATA` (d) The next d bits (d is under software control) are the data of the message. The meaning of the data is entirely under software control, and for example the data could contain a CRC code to protect against multiple bit errors in the transmission of data (such a code would have to be interpreted by the software). Another example of how the data can be used is that the data can contain a local memory address and some data to be written, and the message could be interpreted as a request to write into memory at the destination processor.

`PARITY` (1) The next bit is a parity bit which covers the `CUBE-ADDRESS`, the `PROC-ADDRESS` and the `DATA`. This can be used to detect all single bit errors in router network. Note that since the `CUBE-ADDRESS` is modified as a message crosses a cube wire, the parity bit must be inverted when a message crosses a cube wire.

3.2 The Injector

The injector (see Figure 6) is a *concentrator* [CL86] which has as input the previously buffered messages (shown coming from the left) and a set of up to sixteen messages from the processors (shown coming from the

Figure 6: The Injector concentrates buffered messages with messages from the processors.

bottom). The injector can send up to 7 messages to the heart. First the injector tries to fill up those 7 slots with messages from buffers. Then the injector places as many of the uninjected messages (from processors) as possible into the remaining empty slots. Finally the resulting set of messages is sent to the heart (shown leaving at the right in Figure 6).

The injector is really just a switch which is set up during the injection phase of each petit cycle. For each buffer position to the left of the injector and for each processor one only needs to know whether there is a message or not in order to set the switches. The `MSGP` bit serves this purpose, and thus at just the time that the `MSGP` bits are all at the injector, the switch is set. A two-dimensional priority chain is used to determine the switch settings at any of the $7 \cdot 16 = 112$ sites in the injector.

3.3 The Ejector

The ejector (see Figure 7) removes from the network the messages that are at their destinations and gives them to processors. Messages arrive from the heart (shown on the left) and are either sent to the correct destination processor (out the bottom) or they stay in the same row and go to the injector (as buffered messages). Note that all sixteen processors are connected to the ejector, and that the ejector implements a cross-bar switch which can switch the bitstream from any buffer to any processor. If two bitstreams are switched to the same processor then the two bitstreams are OR'ed together bitwise. There is a mode which prevents the OR'ing together of bitstreams by only allowing one message to be ejected per petit cycle. That mode slows the system down by an amount which depends on the routing pattern (for example, random routing patterns typically are slowed down by a factor of 2 to 2.5 when in this mode).

The ejector determines that a message is at its destination node by looking at the 12 bits of the `CUBE-ADDRESS`. (A simple finite state machine can determine if all 12 bits are zero). The ejector determines the destination processor by looking at the 4 bits of the `PROC-ADDRESS`. The `VP-ADDRESS` is interpreted by software.

3.4 The Heart

The heart (see Figure 8) receives messages from the injector, routes them across each of the cube dimensions (sending up to one message out across each cube wire, and receiving up to one message in across each cube wire), and then gives them to the ejector. There are twelve columns in the heart, one for each cube dimension. In each column there are seven sites.

The algorithm implemented by the heart is first described as though the system were not pipelined (in Section 3.4.1), and then the details of the pipelining are revealed (in Section 3.4.2). Briefly, each of the sites in the heart (shown as a small square in Figure 8) is the same, and together they implement the functionality of the heart.

Figure 7: The ejector implements a cross-bar between the messages coming from the heart and the processors. Unejected messages are sent to the buffers.

Figure 8: The heart performs the switching of messages across cube wires.

3.4.1 If the Heart Were Not Pipelined

This section describes the routing algorithm as though the router were not pipelined. In effect, this section specifies the path that each message will take through the routing network, but this section does say how the bits in a message follow the path.

At any given time, the messages in the router are either all in the injector, all in the ejector, or all in one column of the heart. Messages move from column i to column $i+1$ during a period of time called a *dimension-cycle*. When messages are in column i, the ith bit of the `CUBE-ADDRESS` along with the `MSGP` bit of each message is used to determine how to route the messages. The heart chooses a message to send across cube dimension i, and sends the other messages to the next column. Any message which jumps across cube dimension i from chip x will land in column $i+1$ of the heart of chip $x \otimes 2^i$ (where $a \otimes b$ is the bitwise logical exclusive or of two integers). Correspondingly, chip $x \otimes 2^i$ may send a message to chip x, which will receive the message.

All of the messages which do not cross cube wires move to column $i+1$, and the messages are "packed" into the low numbered rows (i.e., the messages move from high numbered rows to low numbered rows if there are any empty low numbered rows.) The message, if any, that arrives across dimension i, is placed in the highest numbered row of column $i+1$.

A message with `MSGP` equal to one, and with a one in the ith bit of the `CUBE-ADDRESS` *wants* to cross dimension i. If there is such a message, then the one in the lowest numbered row (the closest to the top in Figure 8) is removed from the row and sent out cube dimension i.

A given router chip makes the pessimistic assumption that a message will always arriving over the ith cube-wire (from the processor at relative address 2^i) during dimension cycle i. This assumption is pessimistic because it means that we need to make space for that message even if no such message actually arrives. Thus if all seven rows actually contain a message we will have to route one of them across dimension i in order to make space for that message. If none of the messages want to cross dimension i but all seven rows are full then a message is mis-routed. We call this mis-routing *desperation-routing* because it is a last ditch attempt to avoid running out of places to put messages by making some messages travel *away* from their destination. If a message needs to be desperation routed, the heart always picks the message in the highest numberedrow (the row at the bottom of Figure 8).

Thus, when making the routing decision for dimension i at a chip, there are three interesting cases (see Figure 9):

1. There is a message at the chip which wants to cross dimension i. In this case, the message in the lowest numbered row, that wants to jump, is allowed to jump across dimensions i, and all the messages in higher numbered rows move to the next lower numbered row. (Shown in Part (a) of Figure 9.)

2. There are strictly fewer than seven messages at the chip, and none of them want to jump across dimension i. In this case no message jumps across dimension i from this chip. The messages are all packed into the lowest numbered rows. (Shown in Part (b) of Figure 9.)

3. There are exactly seven messages at the chip, and none of them want to jump across dimension i. In this case, the message in the highest numbered row is desperation routed across dimension i in order to create space for a message which may be simultaneously arriving across dimension i. (Shown in Part (b) of Figure 9.)

Note that in each case, the router has arranged things so that row six (the highest numbered row) always contains a free space for an incoming message.

This routing algorithm could conceivably fail to terminate since there is no obvious guarantee that, over time, any message will make any net progress towards its destination. In practice we have never found a message pattern which fails to terminate, although there is a known class of routing patterns in which a 64K Connection Machine router would fail to deliver messages if there were fewer than seven rows in the heart. This class of routing patterns was constructed by Washington Taylor in 1984 while he was employed at Thinking Machines.

Figure 9: Three interesting cases for routing behavior.

Should I give an example of one of those routing patterns?

Even if the Connection Machine router were to livelock, a simple timeout mechanism would force the router to empty, and the routing could be retried in two steps (with half the messages being delivered in each step).

Figure 10 shows the path that messages would take thorough the heart for a few interesting cases. Messages are shown entering the heart from the injector on the left, and leaving the heart to the ejector on the right. Messages also arrive across the cube wires (shown at the bottom) and leave across cube wires (shown at the top). Note that the messages arriving across the cube wires are not the same messages which left across the cube wires, but that the messages are going to identical chips when the cross the cube wires.

3.4.2 Pipelining the Heart

If the Connection Machine router were not pipelined, only one cube dimension would be busy sending messages at a time, and we would use at most one twelfth of the wires. That would be an unacceptable situation, since we are trying to optimize for wire utilization. However, the Connection Machine router actually does perform pipelining which allows every wire to transmit a bit every clock cycle.

At a given time step, dimension twelve is be sending one bit of every message message, while dimension eleven is sending another bit, and dimension ten is sending another. This style of pipelining is sometimes called *wormhole routing* [Dal87]. Figure 11 shows how a message could be spread through eight chips of a Connection Machine computer.

The right way to measure time on a machine like the Connection Machine computer is in units of *bit-times.* The amount of time it takes to trasfer one bit across one cube dimension is called a bit-time. Every bit-time, every router node sends a total of 12 bits of data, one across each of the 12 cube dimensions.

There are some tricky details in the exact mechanism that allows the system to be pipelined. To understand those details, one needs to understand the ways that the pipelining could be implemented wrong. The following section describes some of the 'wrong' ways to pipeline the Connection Machine routing algorithm, and concludes with the actual implementation, which in comparison is 'right'.

Note that for a given column of the heart, say column number i, only two bits of each message are needed in order to make the routing decision: The `MSGP` bit and the ith `CUBE-ADDRESS` bit. Thus, column i may make its routing decision no sooner than when `CUBE-ADDRESS[i]` arrives.

Figure 10: The paths that messages take through the heart are shown for a few interesting cases. Case (a) shows the situation where every message wants to cross every cube wire. Case (b) shows the situation where a message must desperation route in column 0, and where no message crosses a cube wire in column 1.

Figure 11: A message can be spread through a machine, and different parts of the message can be crossing different dimensions at the same time.

3.4.3 The Wrong Solution

The 'wrong' pipelined implementation works as follows: Look at a given column of the Heart, (say column number i). Column i receives the message as shown in Figure 5. Thus, exactly $i+1$ bit-times after the `MSGP` arrives, column i has enough information to make its routing decision. The `MSGP` can then be sent to column $i+1$ during step $i+2$ bit-times after the `MSGP` arrives at column i.

The amount of storage needed in the heart can be determined as follows. Column i needs $i+2$ bits of storage for each message, since it has to hold onto the `MSGP` and $i+1$ bits of the `CUBE-ADDRESS` (remember that columns are numbered from zero). Thus to total amount of storage required is

$$S_{\text{heart}} = \sum_{i=0}^{c-1} b(i+2) = b(c^2+3c)/2,$$

where c is the number of dimensions (equal to the number of columns), and b is the number of rows in the heart. Setting $c = 12$ and $b = 7$ gives 630 bits of storage in the heart.

The time it takes for a single message to traverse the heart can be determined as follows. Suppose a message contains d bits of data, and the heart has c columns and b rows, and there are v bits of `VP-ADDRESS`. Then a message is $l = 6 + c + v + d$ bits long. In our system $c = 12$ and typically $v = 0$ and $d = 32$, giving $l = 6 + 12 + 0 + 32 = 50$. Thus from the time that the `MSGP` arrives at a given point to the time that the last bit of the message arrives at the same point is l bit-times. However, in column i, the message incurs $i+2$ bit-times of delay, gibving a total delay through the heart of

$$T_{\text{heart}} = l + \sum_{i=0}^{c-1}(i+2) = l + (c^2+3c)/2.$$

Thus, for our typical case $T_{\text{heart}} = 140$, 50 of which is accounted for by the length of the message. For a given message, the next petit-cycle can not start until the `MSGP` emerges from the heart, which takes 90 bit-times. Thus the petit-cycle is 90 bit-times long, and any given wire is only used for 50 of those bit-times.

It might be possible to start a second batch of messages through the heart, and make the petit-cycle 100 bit-times long, alternating between the first batch and the second batch of messages. This has the following disadvantages:

- More than twice as many messages are needed in order to get the same efficiency from the router. This becomes especially important when there are only a few messages in the system.
- As the length of the data changes, d, changes, which means l changes. Therea are applications which use only a few bits of data, and in that case the number of batches required would be much larger, i.e., if l becomes much smaller than $(c^2+3c)/2$ the number of batches required goes up.

This is not to say that such a scheme is unworkable, but would certainly require some additional engineering to make it practical and efficient.

3.4.4 Dropping Address Bits

The second solution is not 'wrong'; rather it, but does not apply directly to the Connection Machine router. This solution is to throw away address bits as they are consumed by the router. Throwing away address bits as they are consumed by the router imposes some serious constraints on the routing algorithm. Namely, the algorithm must be *oblivious*, rather than *adaptive.*

An *adaptive* algorithm is one in which the path that a given messages takes is affected by the local loading conditions of the network. Often there are several "shortest" paths from the message's source to its destination; An adaptive algorithm might choose which shortest path to use based on local information. The routing algorithm used by the Connection Machine system is adaptive.

An *oblivious* algorithm is one in which the path that a message take is uniquely determined without looking at the network load. Some oblivious algorithms choose the path based solely on source and destination, although more complex choices might be made (e.g. by randomly picking one of the shortest paths at the time that the message is injected into the network). In particular, the E-cube routing algorithm[DS87] (described in the next paragraph) is oblivious.

Many wormhole routers use an oblivious routing strategy. On a hypercube, an oblivious strategy might take the form of the following rule:

> A message does not travel across dimension i if there is some $j < i$ such that the message needs to travel across dimension j.

This rule means that, given a source and destination, there is a unique path from that source to that destination, and messages traveling from that source to that destination will folow that unique path no matter what else is going on.

There are some disadvantages to oblivious routing, some of which appear in [Kus87], but the rule has an interesting advantage: Once a message starts considering dimension i, the router does not need the address bits for dimensions less than i (because those address bits have already been used). Thus such a wormhole router can throw away address bits as messages proceed through the router. Note also that fewer bits travel across high numbered cube dimensions than across low numbered cube dimensions (because almost all of the address bits need to be shipped across low numbered dimensions, but almost none of the address bits need to be shipped across high numbered dimenions).

Note then that column i of such an oblivious heart would recieve `MSGP` and then *on the very next bit-time* the ith bit of `CUBE-ADDRESS` would arrive.

Thus the total amount of storage required for each message in each column of the heart is only 2 bits (for the `MSGP` and the address bit) giving total storage required (in bits) of

$$S_{\text{heart}} = 2bc = 168.$$

The delay through a column is simply 2 bit-times, giving a total delay through the heart of

$$T_{\text{heart}} = (l - c) + 2c$$

bit-times, which for our typical case is 62 bit-times. (Note that the length of the message coming out of the heart is c bits less than l, since the `CUBE-ADDRESS` has been completely stripped off.)

The Connection Machine router is adaptive , and so it is not possible to throw address bits away as messages progress through the router.

3.4.5 The Actual Solution

The solution actually used in the Connection Machine router is a combination of the two 'wrong' solutions. We modify the solution which drops address bits as follows: Column i, rather than throwing away address bit i, keeps address bit i, until all of the other address bits have been forwarded to column $i+1$, then the ith address bit is transmitted to column $i+1$, then the data is transmitted.

The `CUBE-ADDRESS` is thus 'rotated' as it travels through the heart. When it leaves column i, the low order i bits of the `CUBE-ADDRESS` have been rotated to the high order i positions.

Each message needs two bits of storage in each column of the heart (one for the the `MSGP`, one for the saved `CUBE-ADDRESS`), for total storage requirements in bits of

$$s_{\text{heart}} = 2bc = 168.$$

The delay through the heart is 2 bit times for each column, and no bits have been stripped off, giving, in bit-times

$$T_{\text{heart}} = l + 2c,$$

Figure 12: The number of petit cycles needed to route messages as a function of the virtual processor ratio. The solid line shows the actual values, while the dashed line shows a theoretical lower bound.

which in our typical case is 74 bit-times.

Note, that two subsequent petit-cycles can be pipelined, since the first bit which comes out of the heart is the low order address bit, and the message has been restored to its original form. Thus, if a delivery cycle takes p petit-cycles, then the total time (in bit-times) is

$$T_{\text{delivery-cycle}} = \begin{cases} pl + 2c & \text{if } 2c \leq l \\ l + 2cp & \text{if } 2c > l \end{cases} .$$

For typical cases where $l = 50$, and $p = 12$, that gives a total time of 624 bit-times for a delivery cycle. The condition $2c \leq l$ is the condition that says that the pipeline stays full, and all the wires send a bit of data every bit-time (except during the pipeline startup, which uses c bit-times at the of beginning the delivery-cycle) and shut-down (which uses c bit-times at the end of the delivery-cycle). Note $2c \leq l$ is true whenever $v + d \geq 6$ (so for example, 6 bits of data is enough to keep the pipeline full).

Thus, a naive pipelined implementation of the router would result in a substantial loss of bandwidth across the wires, and would require substantially more on-chip storage space as compared to the actual implementation of the Connection Machine router.

4 Evaluation and Conclusions

4.1 High Wire Utilization

The Connection Machine router system achieves very high wire utilization. Figure 12 shows the number of petit-cycles needed to route random message patterns for various virtual processor ratios. The solid line shows the number actually used by the Connection Machine router, while the dotted line shows a theoretical lower bound on the amount of time needed.

The lower bound is derived as follows: Suppose that there are v messages per processor. Then there are $2^{16}v$ messages in the machine. Since the messages are taking a random routing pattern, about half the messages (about $2^{15}v$) need to cross each cube dimension, and about a fourth of the messages (about $2^{14}v$) need to cross any given cube dimension in any given dimension. In one petit cycle, each chip can send at most one message across each cube dimension, and there are 2^{12} chips, so it will take about

$$\frac{2^{15}v}{2^{12}} = 8v$$

petit cycles to route the messages. In fact, a slightly higher lower bound can be derived since in order to route the messages in time $8v$ or less, it must be the case that no more than $2^{14}v$ of the messages want to cross any given dimension in any given direction. The probability, P, that in any given dimension and direction across that dimension, strictly more than a fourth of the messages will want to cross that dimension is about one half. More precisely, the probability is

$$P = \frac{1 - \frac{\binom{m/2}{m/4}}{2^{m/2}}}{2},$$

(where $m = 2^{16}v$), which, using Stirling's approximation, gives

$$\begin{aligned} P &\simeq \frac{1 - \frac{2}{\sqrt{2\pi 2^{15} v}}}{2} \\ &\simeq \frac{1}{2}. \end{aligned}$$

The number of petit cycles could be $8v$ or less only if no more than half of the messages cross each dimension in each direction. Since there are twelve dimensions and two directions for each dimension, the probability that the number of petit cycles is $8v$ or less is 2^{-24}. Thus we expect the number of petit cycles to be strictly greater than $8v$ (which when $v = 1$, means that it will probably take at least nine petit cycles).

Note that this lower bound completely ignores the fact that two messages may need to use the same cube wire in dimension i (and one may be delayed) even though somewhere else in the machine there is a free cube wire in dimension i, and if our mathematics could account for this fact, the lower bound would likely be even higher (which would mean that the actual hardware implementation would look even better).

Thus, for a virtual processor ratio of one, the hardware is only about a third worse then the optimistic lower bound given above, and as the virtual processor ratio increases, the router's behavior approaches the lower bound. By this measure, the Connection Machine router achieves almost ideal wire utilization.

Part of the difference between the Connection Machine router and many other routers is that the Connection Machine router is designed to be used in a SIMD machine, where most other routers are designed to be used in MIMD machines. For example, the load presented to the router on a MIMD machine is more uniformly distributed over time than the loaded presented to the Connection Machine router, which must cope with large numbers of messages all at the same time. Thus, it is important for the Connection Machine router to optimize for the amount of time it takes to deliver large numbers of messages, and so the design attempts to maximize the throughput of the router. The MIMD routers, on the other hand, usually are very concerned with the time it takes for a single message to be delivered when the network is lightly loaded.

The analysis given in [Dal87] indicates that a bit-serial hypercube's latency is substantially higher than networks of lower dimension with wider channels, and so the Connection Machine router has more latency under lightly loaded conditions when compared to routers such as the Torus Router[DS86].

On the other hand, the analysis given in [Dal87] indicates that at most half the theoretical throughput of the toroidal network can actually be used, and further simulations done at Thinking Machines Corporation has confirmed that the toroidal network has low throughput as compared to the actual bandwidth of the wires [Kus87].

Thus, there is a clear tradeoff between the design of the Connection Machine router and the design of routers such as the Torus Router. The Connection Machine router is optimized for heavy message loads, rather than for light message loads. In retrospect, it appears that this design decision was the correct one for the Connection Machine computer.

4.2 Packaging Technology

The Connection Machine computer makes good use of the packaging technology compared to some other layouts (such as is used in the standard multi-stage cube network layout). We can think of the Connection Machine router system as being a multi-stage cube system (see Figure 13) where each of the columns of the heart forms one layer of the multi-stage cube. In the usual multi-stage cube layout, the switches inside the

Figure 13: The Connection Machine router network can be thought of as a multi-stage cube network, where the choice of packaging is made by putting a whole row of the network on one chip (shown on the left). The usual multi-stage cube network layout tries to put pieces of several rows on the same package, and offers no systematic way of using the fact that on-package connectivity is higher than off-package connectivity (shown on the right).

largest possible *square* are put into one package (such as a chip or a board). Such a layout ignores the fact that interconnection within one package is much less expensive than interconnection between packages. The Connection Machine computer integrates all the switches in one row of the multi-stage cube network, and then uses the fact that the bandwidth from one stage to the next is much higher for those messages which do not make a "diagonal" jump. In the case of the Connection Machine computer, the "on chip wires" are called "rows of the heart".

Thus, the Connection Machine router has 7 messages to pick from when trying to find a message to send across a cube wire. This results in much higher wire utilization than standard multistage-cube routers which have only 2 messages to pick from when trying to send a message across a cube dimension. This increased flexibility can be traced directly to the fact that the Connection Machine router packages its multistage-cube differently than the standard package.

4.3 Technological considerations

The low level protocol for transfering bits across cube wires uses what is now very conservative technology. The wires in the router are several meters long, and the whole wire is allowed to settle to the value being driven at one end before the value is sampled at the other end of the wire. This means that the wires run relatively slowly, compared with, say the Torus Routing Chip[DS86].

The Connection Machine router is implemented in semicustom ASIC[3] technology, rather than in full custom layout VLSI chips, which puts it at an additional speed disadvantage when compared to some of the other more recently implemented routers. The speed disadvantage of ASIC technology is more than balanced by the relatively fast design time of ASIC parts, and by the fact that it is possible to obtain second sources for ASIC chips. Both of these advantages help make the Connection Machine successful as a product.

[3] Application Specific Integrated Chips (ASIC) are typically built using gate array or standard cell technologies.

4.4 Circuit Switched vs. Packet Switched

The Connection Machine router system combines the best characteristics of circuit switched networks and packet switched networks. The Connection Machine router operates as circuit switched system within a petit cycle, but as a packet switched system between petit cycles. The advantages of a circuit switched network are that a low percentage of the resources are spent actually setting switches, and the latency for a given bit, once the switch is set, is low. The advantages of packet switched are that the system can adaptively route messages, improving the performance of the router [Kus87].

4.5 The CM1 vs. the CM2

Thinking Machines Thinking Machines announced and has been delivering the CM2 Connection Machine system since 1987, but this paper describes the CM1 Connection Machine routing system. There are many important improvements in the CM2 over the CM1. The CM2 has 512 Megabytes of error corrected RAM, 2048 floating point units, a parallel I/O subsystem (e.g., for graphics and disks). The features which interact directly the message router fall in the following three areas: Indirect addressing, combining routing, and technology. A full description CM2 router will appear in a later paper, and a description of the CM2 data processor architecture appears in ??.

The CM1 does not have indirect addressing, and this results in a performance penalty when running with high virtual processor ratios. (This is because on a given petit cycles, messages can only be ejected to one bank of virtual processors. Thus, if the virtual processor ratio is v, then it takes on the order of v^2 petit cycles just to eject messages once they have arrived. The CM2 has substantially more processor memory than does the CM1. This means that users typically run their CM2 programs with much higher virtual-processor ratios, in which case the quadratic time cost would be substantial. But the CM2 also has indirect addressing, which allows messages addressed to different virtual processor banks to be ejected in the same petit-cycle. This means that even with very high virtual-processor ratios, the CM2 sustains very high wire utilization.

The CM2 has a mechanism which allows messages to be combined in the router if they are addressed to the same location. This is important if the message passing network is being used to implement a shared memory concurrent read, concurrent write (CRCW) [?] programming model. It is possible that many different virtual processors will try to access the same location in memory. The router detects messages which are destined for the same virtual-processor, and combines them. This works for both *reads* and *writes*, but not for operations such as fetch-and-add. Fetch-and-add can be implemented in software on the CM1 or the CM2 using *sort*, *send*, and *scan* primitives.

The CM2 is substantially faster and more robust than a CM1 just because of technological advances such as increased silicon density and clock speed.

Thinking Machines Corporation is continuing to improve the Connection Machine family of computers.

4.6 Summary

The Connection Machine computer is a massively parallel supercomputer which places high demands on its router system. The Connection Machine router does a very good job of satisfying those demands, whether we measure such satisfaction in terms of wire utilization, packaging, message latency, or worst case routing times.

Acknowledgements

The address rotation scheme was invented by Lyman Hurd and Richard Feynman. Many students, faculty, and members of the staff at the MIT Artificial Intelligence Laboratory and at Thinking Machines Corporation have contributed to the success of the Connection Machine.

References

[?] ?. ?.

[al85] G. F. Pfister et al. The IBM research parallel processor prototype(RP3): introduction and architecture. In *Proceedings of the 1985 International Conference on Parallel Processing*, August 1985, pages 764–771.

[Bat80] Kenneth E. Batcher. Design of a massively parallel processor. *IEEE Transactions on Computers*, C-29(9):836–840, September 1980.

[Bell77] Bell system technical journal. February 1977.

[Ben62] V. E. Beneš. On rearrangeable three-stage connecting networks. *Bell System Technical Journal*, pages 1481–1492, September 1962.

[Ble87] Guy Blelloch. Scans as primitive parallel operations. In *Proceedings of the 1987 International Conference on Parallel Processing*, August 1987, pages 355–362.

[BDM*72] W. J. Bouknight, Stewart A. Denenberg, David E. McIntyre, J. M. Randall, Amed H. Sameh, and Daniel L. Slotnick. The ILLIAC IV system. *Proceedings of the IEEE*, 60(4):369–388, April 1972.

[Clo53] C. Clos. A study of nonblocking switching networks. *Bell Syst. Tech. J.*, pages 406–424, March 1953.

[CL86] T. H. Cormen, and C. E. Leiserson. A hyperconcentrator switch for routing bit-serial messages. In *Proceedings of the 1986 International Conference on Parallel Processing*, August 1986, pages 721–728.

[DS86] William J. Dally, and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1(3):187–196, 1986.

[DS87] William J. Dally, and Charles L. Seitz. Deadlock-free message routing in multiprocesor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[Dal87] William J. Dally. Wire-efficient VLSI multiprocessor communication networks. In *Proceedings of the 1987 Stanford Conference on Advanced Research in VLSI*, 1987, pages 391–415.

[DGK86] Susan Dickey, Allan Gottlieb, and Richard Kenner. Using VLSI to reduce serialization and memory traffic in shared memory parallel computers. In *Proceedings of the Fourth MIT Conference on Advanced Research in VLSI*, April 1986, pages 299–316.

[Hil85] W. D. Hillis. *The Connection Machine.* The MIT Press, Cambridge, MA, 1985.

[HS86] W. Daniel Hillis, and Guy L. Steele, Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, December 1986.

[Hor82] R. Michael Hord. *The ILLIAC IV: The First Supercomputer.* Computer Science Press, 1982.

[Intel] *A Technical Summary of the iPSC/2 Concurrent Supercomputer* Intel Scientific Computers.

[ITT] *System 12, ITT 1240 Digital Exchange, A Technical Description* ITT.

[KS82] Clyde P. Kruskal, and Mark Snir. *Some Results on Multistage Interconnection Networks for Multiprocessors.* Technical Report 51, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, May 1982.

[Kus87] Bradley C. Kuszmaul. Adaptive vs. oblivious message routing in large interconnection networks. September 1987. to appear.

[Lee87] Kyungsook Yoon Lee. A new Beneš network control algorithm. *IEEE Transactions on Computers*, C-36(7):768–772, July 1987.

[LPV81] G. Lev, N. Pippenger, and L. G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, 1981.

[Red73] S. F. Reddaway. DAP - A Distributed Array Processor. In *Proceedings of the First Annual Symposium on Computer Architecture*, 1973, pages 61–65.

[Ret86] R. D. Rettberg. Shared memory parallel processors: the Butterfly and the Monarch. In *Proceedings of the Fourth MIT Conference on Advanced Research in VLSI*, April 1986.

[Sch80] J. T. Schwartz. Ultracomputers. *ACM Transactions on Programming Languages and Systems*, 2(4):484–521, October 1980.

[Sei85] C. L. Seitz. The Cosmic Cube. *CACM*, 28(1):22–23, January 1985.

[SUK*] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lopovski. An overview of the texas reconfigurable array processor. In *AFIPS Conference Proceedings 1980 National Computer Conference*, Anaheim, California, May 19–22, 1980, pages 631–641.

[WF80] C.-L. Wu, and T.-Y. Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, C-29:696–702, August 1980.